Kévin
Salmon

BTS-SIO1

12 Rue du Bois Chaland
91090 Lisses
01 69 11 26 26

13 juin 2022

Note de synthèse de 1ère année au sein de Martin Brower :

Développement WEB & Salesforce

Lycée Geoffroy Saint-Hilaire
4 Avenue Geoffroy Saint-Hilaire, 91150 Etampes
Année Scolaire 2021-2022

Remerciements :

Dans le cadre de ce rapport, je souhaite remercier l'ensemble de l'entreprise Martin Brower et tout particulièrement l'équipe Salesforce.

J'ai été accueilli chaleureusement et cela m'a permis de m'épanouir dans un environnement de travail riche et intéressant.
Je remercie plus précisément Mickaël Riviere pour m'avoir fait confiance sur des projets importants et de m'avoir laissé intégrer son équipe dans le cadre de mon apprentissage.
Mais également pour m'avoir permis d'être son apprenti. Cette expérience a été très enrichissante et m'a offert la possibilité d'apprendre énormément sur le comportement à adopter en entreprise ainsi que sur la communication au sein d'une équipe. L'autonomie que Mickaël a su me laisser m'a permis de monter en compétences rapidement, sur des outils que je n'avais jamais utilisés et cela afin d'aider au mieux l'ensemble de l'équipe dans le cadre des projets à mener.

Enfin je souhaite remercier tout le reste des membres de l'équipe Salesforce pour m'avoir intégré si facilement parmi eux et pour m'aider lorsque cela été nécessaire.

Apprenti : Kévin SALMON
Maître de stage : Mickaël RIVIERE
Professeur Tuteur : Mr. Ramond

Sommaire :

I/ Introduction :

Dans le cadre de ma formation en BTS SIO (Option SLAM) Services Informatique aux Organisations, option développeur d'application et de logicielle. J'ai effectué un stage du Lundi 23 mai 2022 jusqu'au vendredi 24 juin 2022, au sein de Martin Brower à Lisses.

a) Cadre du Stage :

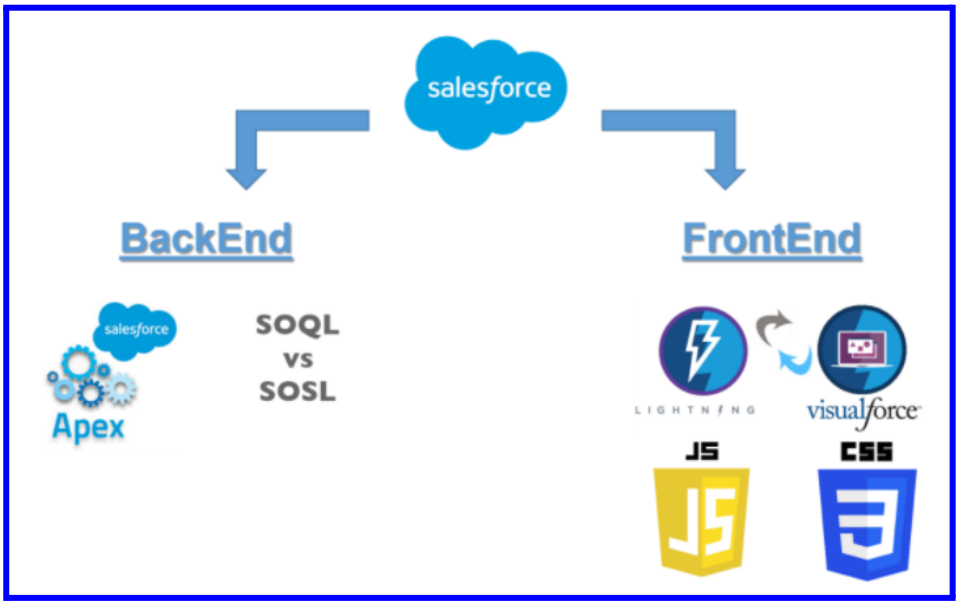
Le poste que j'occupe en tant que stagiaire, est un poste de développeur Web & Salesforce au sein de l'entreprise Martin Brower à Lisses.

Le site se situe plus précisément au 12 rue du Bois Chaland 91090 à Lisses.

Pour réaliser mes missions, plusieurs compétences et langages sont nécessaires :

- Connaissance de la plateforme Salesforce.
- Apex (langage orienté objet proche du Java).
- SOQL (Salesforce Object Query Language) un langage proche du SQL.
- Lightning et VisualForce proches du HTML.
- JavaScript et CSS.

Voici une représentation plus schématique de Salesforce :



II/ Présentation de l'entreprise :

a) Martin Brower

Martin Brower est une entreprise américaine créée en 1956, suite à une fusion entre Brower Paper Company et de la Martin Paper Company. Cette entreprise est spécialisée dans la logistique et le transport, elle collabore presque exclusivement avec l'entreprise McDonald's avec laquelle elle travaille depuis 1956. Elle était alors simplement chargée de livrer des serviettes en papier.

L'entreprise a ensuite été rachetée par le groupe américain Reyes Holdings en 1997, spécialisé dans le transport de marchandise agro-alimentaire. À la suite de ce rachat, l'activité de

l'entreprise n'a cessé de grandir conjointement avec celle de McDonald's jusqu'en 2012 avec le rachat de l'entreprise Keystone Foods spécialisée dans la logistique.
À l'occasion de ce rachat le siège social français de Martin Brower s'est alors mis en place à Lisses dans le 91, lieu où j'effectue mon apprentissage.

L'entreprise Martin Brower est donc un prestataire logistique travaillant avec Mc Donald's depuis 1956. Elle est chargée entre autres de stocker et d'acheminer, les marchandises du groupe à travers tous les restaurants dont Martin Brower s'occupe soit plus de 25 000 répartis dans 19 pays. (1 486 en France).

En tant que prestataire logistique, Martin Brower France a dégagé un chiffre d'affaires de 1 239 522 200 € en 2017. Ce chiffre d'affaires est majoritairement issu de son travail avec l'entreprise McDonald's. Cependant, bien qu'il existe un partenariat de long terme entre MB et Mcdonald's, le groupe Martin Brower essaie de diversifier ses activités au cours de ces dernières années en utilisant son expérience dans plusieurs domaines tels que la réalisation de projets (informatiques ou logistiques).

La plus grande difficulté rencontrée dans cette évolution est le fait que l'entreprise doit se rendre entièrement disponible à son principal client pour éviter de perdre cette activité au profit d'un autre concurrent.
En effet, les concurrents de Martin Brower sont nombreux ce qui oblige le groupe à être le plus innovant et attentif possible dans le but d'être leader sur le marché. Parmi les concurrents les plus redoutables dans le domaine logistique, on peut retrouver le groupe STEF et le groupe Havi qui s'occupe également du transport pour les restaurants McDonald's dans de nombreux pays.

L'entreprise Martin Brower se distingue de ses concurrents notamment par ses méthodes de travail saluées par le prix de « Top employer » obtenu lors des quatre dernières années. Ce prix récompense les excellentes conditions de travail mises à disposition des employés. En plus de ce prix le groupe possède également de nombreuses certifications (ISO 9001 : pour la capacité à fournir des produits en corrélation avec les exigences aussi bien clientes que légales, ISO 14001 : pour la prise en compte des enjeux environnementaux, ISO 22000 : concernant la sécurité alimentaire ...).

Martin Brower France compte aujourd'hui environ 900 employés à travers toute la France répartis sur 10 sites (9 entrepôts et le siège).

b) Equipe Salesforce

Dans le cadre de mon apprentissage, je fais partie du service Informatique. Le rôle du service informatique est de mettre en œuvre des applications pour que les entrepôts et le siège n'aient pas de soucis dans le bon déroulement de leur activité. Un arrêt soudain dans leur travail engendrerait des perturbations indéniables au sein des restaurants Mc Donalds.

Durant mon stage, je travaille dans l'équipe Salesforce qui est chargée de la mise en place et du maintien des outils Web ainsi que des applications développées sous Salesforce.
J'occupe un poste d'apprenti développeur Web & Salesforce visant à renforcer l'équipe dans le cadre des nombreux nouveaux projets.

Au sein de l'équipe Salesforce, tous les membres sont formés pour être opérationnels sur une majorité de projet, et cela dans l'objectif d'aider au maintien et à la correction éventuelle de bug sur toutes les applications.

En effet, l'équipe s'occupe de toutes les applications Salesforce pour tous les pays où est présent Martin Brower (il n'y a qu'une seule équipe Salesforce dans le groupe, celle du site de Lisses).

Cette équipe est habituée à avoir des apprentis, ce qui constitue un avantage concernant les méthodes de travail à adopter.

Le rôle de chacune des équipes est le suivant :

-Pilotage transverse : S'occupe du suivi global de tous les projets du service ainsi que leurs coûts financiers.

-Projets métier : Travaille sur les différentes interfaces entre les outils informatiques de l'entreprise, les projets métiers et outils d'entrepôts, ainsi que l'extraction des données avec le business intelligence.

-Projet Web et Salesforce : S'occupe du développement et du support sur les applications Web et sur la plateforme Salesforce.

-Sécurité, Infrastructure & Architecture Réseau : Travail au support de tous les utilisateurs en cas de problèmes avec les logiciels ou le matériel informatique. S'occupe également du réseau informatique de l'entreprise et des accès utilisateurs.

III/ Présentation du poste de Stage :

1. Contexte :

Les activités de l'équipe Salesforce ont connu une forte croissance au cours des dernières années. Effectivement, les besoins du groupe envers les applications Salesforce ont augmenté de manière exponentielle ce qui a nécessité un agrandissement de l'équipe. C'est donc dans l'objectif de renforcer la partie Salesforce de l'entreprise.
L'objectif principal de mon recrutement est d'accompagner en tant que stagiaire, l'équipe dans sa croissance ainsi que de travailler sur une partie des nouveaux projets (MbSync) tout en assurant la maintenance des autres applications.
Afin de la réaliser ces objectifs, il a fallu me former à la plateforme Salesforce que je n'avais jusqu'alors jamais vu.

2. Ressources :

Afin de mener à bien les missions qui m'ont été affectées, j'ai pu bénéficier de nombreuses ressources. Dès mon arrivée, on m'a attribué une place stratégique au sein de l'entreprise. De ce fait, j'étais au plus proche des membres de mon équipe et de mon tuteur qui a pu me guider au maximum lors de mes débuts. Des ressources matérielles m'ont également été confiées telles que deux écrans, un ordinateur portable ou encore un poste téléphonique afin de communiquer au mieux avec les personnes présentes sur les sites.
En plus de toutes ces ressources, on m'a également conseillé une plateforme de formation dédiée à Salesforce appelée « Trailhead ». Sur celle-ci, on peut retrouver plusieurs leçons et modules visant à apprendre les bases de la plateforme ainsi que des outils et langages mis à disposition.

3. Organisation :

L'organisation de l'équipe est effectuée par projet de manière générale. Personnellement, je travaille conjointement avec mon maître d'apprentissage Mickaël Riviere. Au début de mon apprentissage, l'ensemble de l'équipe Salesforce me planifiait l'ensemble de mes tâches de façon à ce que je sache exactement quoi faire et à quel moment.

4. Outil Salesforce

L'outil Salesforce est utilisé au sein de Martin Brower depuis 2008. Il s'agit d'un outil cloud développé par l'entreprise du même nom permettant d'assurer principalement le suivi de la relation client des entreprises l'utilisant. C'est donc un outil de CRM (Customer Relationship Management).
C'est actuellement la plateforme numéro une en terme d'outil CRM et est utilisé par plus de 150 000 entreprises à travers le monde. C'est donc un outil stratégique de gestion des relations et interactions d'une entreprise avec ses clients ou clients potentiels.
L'utilisation de Salesforce constitue un avantage. En effet, cette vaste communauté est très réactive et possède de nombreuses plateformes pour échanger et faire avancer l'outil dans son évolution. Il existe une plateforme où les utilisateurs peuvent venir déposer des idées d'évolution futures pour l'outil. Ces propositions sont alors étudiées par Salesforce de manière à coller au mieux aux besoins de ses utilisateurs.
Dans cet objectif, l'outil évolue fréquemment à travers trois grosses mises à jour annuelles en vue d'apporter de nouvelles fonctionnalités collant aux demandes des utilisateurs.
En plus d'être un outil CRM, la plateforme peut aussi être utilisée pour faire du développement. Elle permet de coder dans le langage Web le plus répandu au monde à savoir le JavaScript.
La plateforme possède également de nombreux langages propres à elle tels que « Apex », le « SOQL » ou encore le « Lightning Component ».

Qui dit développement dit souvent vente d'applications et c'est dans ce cadre que Salesforce a

mis en place une plateforme appelée l'AppExchange où l'on peut retrouver des applications développées par des utilisateurs et vérifiées en termes de sécurité par Salesforce.
Cette plateforme propose des solutions aussi bien payantes que gratuites

Salesforce donne aussi la possibilité de créer des environnements appelés « Sandbox » correspondant à des environnements de test de manière à pouvoir travailler sans perturber la production pour les utilisateurs finaux.

On peut donc dire au vu des nombreuses particularités de l'outil qu'il est très vaste et peut être plus ou moins difficile à prendre en main. Il faut dans un premier temps essayer de comprendre la logique de l'outil avant de vouloir se lancer dans le développement. Il y a deux parties clé dans l'apprentissage de Salesforce : une partie configuration et une partie développement.

La partie configuration n'est pas pour autant simple à apprendre. L'outil est tellement vaste et possède tellement de particularités que même après plusieurs années d'expérience sur la plateforme, certaines fonctionnalités nous sont inconnues. De plus, via les différentes mises à jour de l'outil, Salesforce ajoute sans cesse de nouvelles possibilités auxquelles il faut également se former. Autrement dit, il est difficile voire même impossible de pouvoir dire que l'on sait configurer parfaitement les fonctionnalités de Salesforce.

Pour comprendre la partie développement de Salesforce, il faut également comprendre la logique de composant de celle-ci. Sous Salesforce, on ne développe pas des pages Web, on développe des composants qui sont appelés dans des pages Salesforce. Cette logique n'est pas facile à comprendre directement, mais elle permet de réutiliser des composants déjà développés et permet donc d'éviter la duplication de code de manière inutile. Les composants sont codés dans des langages proches du HTML tel que le « Lightning component ». Derrière ce composant, on retrouve un contrôleur JavaScript permettant d'appeler des méthodes « Apex » (langage proche du Java) afin de communiquer du côté serveur des applications. Mais il y a aussi une présence du SOQL qui est un langage proche du SQL, pour par exemple effectuer des requêtes depuis plusieurs bases de données.

IV/ Les différentes missions et tâches réalisées :

1. Déploiement d'une Sandbox :

a.1. Contexte

Tout d'abord, pour pouvoir effectuer mon travail sur l'outil Salesforce. Martin Brower à décidé de me créer une sandbox personnelle, dans le but de me laisser faire différentes missions et des travaux sur cette dernière. Cette sandbox est donc destinée à effectuer des tests, pour ainsi répondre à des besoins exposés par l'entreprise. Une sandbox est donc un milieu de test que Salesforce fournit. Ce sont donc des espaces sécurisés pour tester et former ou expérimenter différentes configurations, de nouvelles applications ou des modifications importantes de votre configuration. On peut donc créer des copies de plusieurs environnements, et donc avoir pour conséquence plusieurs sandbox associé à des tâches respectives.

Au sein de cette sandbox, il m'est demandé de la part de l'équipe Salesforce de créer cinq contacts et cinq comptes avec l'application 'CRM FR', et d'attribuer des règles spécifiques. Une règle pour la création d'un utilisateur et une autre pour la création d'un compte.

Cette première mission représente majoritairement de la configuration de Salesforce et non de la programmation. C'est l'une des principales raisons pour laquelle j'ai débuté ce projet afin de me familiariser au mieux avec l'outil avant de commencer à développer.

Note de stage, Kévin Salmon BTS-SIO1

Pour rappel, dans le cadre de ce projet, j'ai effectué une partie du travail conjointement avec mon maître d'apprentissage et d'autres membres de l'équipe.

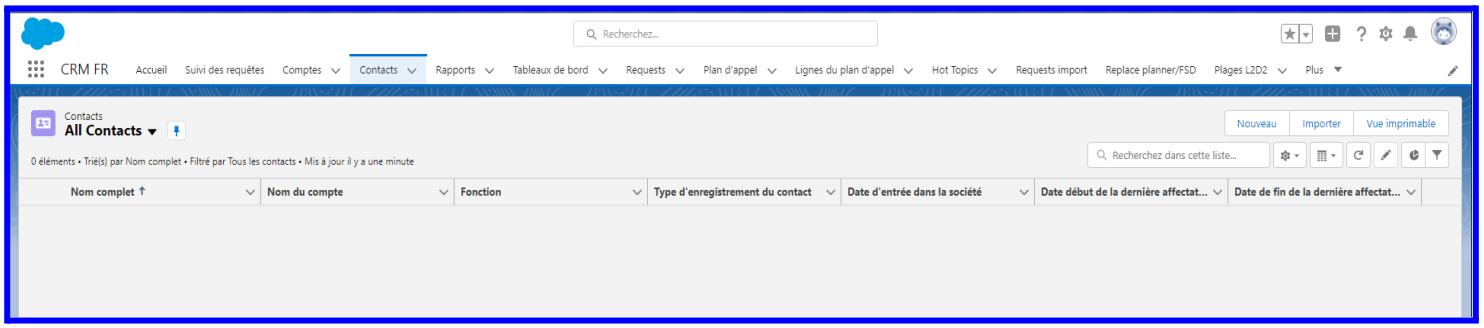
b.1. Fiche de compte

La partie la plus importante du CRM Martin Brower est représentée par les comptes restaurants qui sont associés à des contacts. Pour expliquer cela plus en détail, il s'agit de « fiches » contenant toutes les informations des restaurants aussi bien les attributs du restaurant que les contacts ou requêtes de celui-ci.

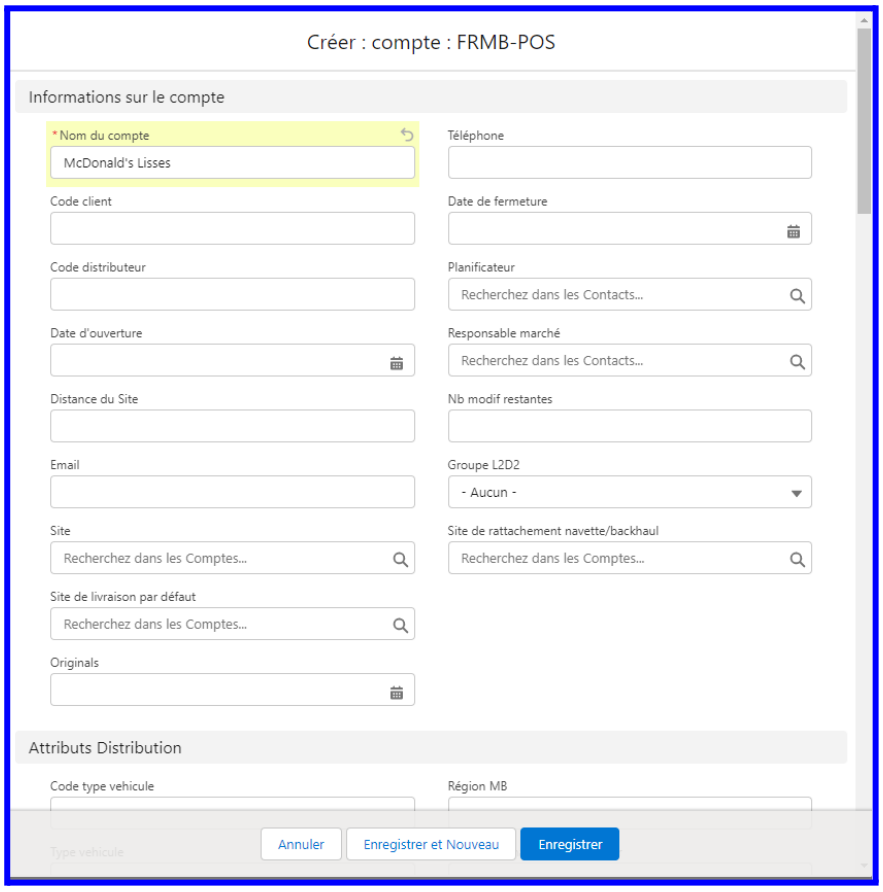
b.2. Création des contacts et des comptes

Pour créer des contacts et des comptes sur Salesforce. Il suffit d'utiliser l'application CRM FR qui permet de gérer tous les contacts et comptes.

Nous pouvons bien voir sur l'image ci-dessous qu'il n'y a encore aucun contact, ni de compte.

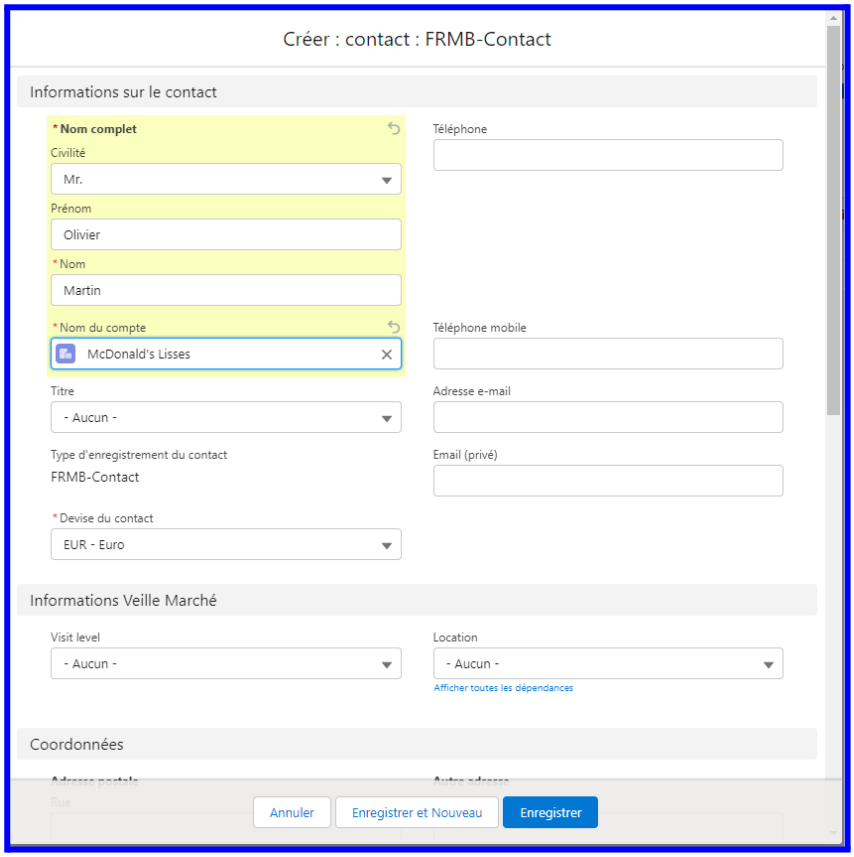


Il faut donc créer tout d'abord un nouveau compte grâce à l'onglet 'Nouveau', ce qui nous emmène sur cette page :



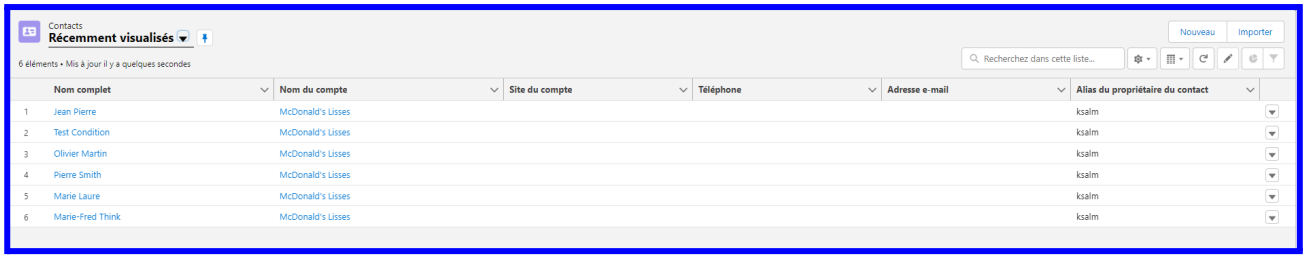
Il n'y a encore aucune restriction spécifique pour créer un contact. Il suffit simplement du nom du compte. Le compte sera donc associé à un ou plusieurs contacts. En effet, les comptes correspondent à des restaurants McDonald's et les contacts correspondent à des utilisateurs, managers, ...

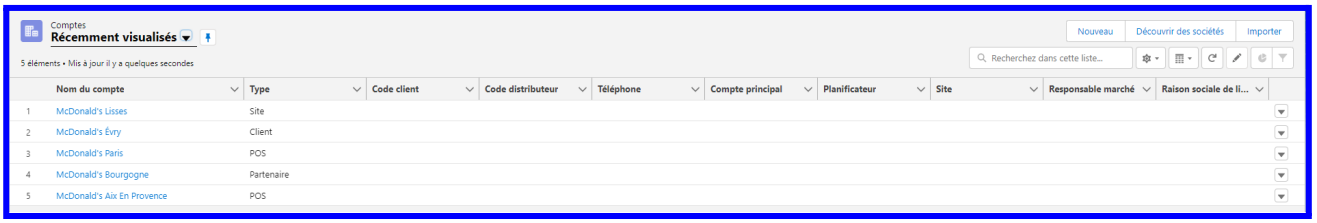
Puis il faut créer un nouveau contact grâce à l'onglet 'Nouveau', ce qui nous emmène sur cette page :



Il n'y a encore une fois, aucune restriction spécifique pour créer un contact. Il suffit simplement d'un prénom, Nom, et le nom du compte qui lui est associé. Les noms de comptes sont les comptes qui ont été créés précédemment. Un contact doit être associé à un compte.

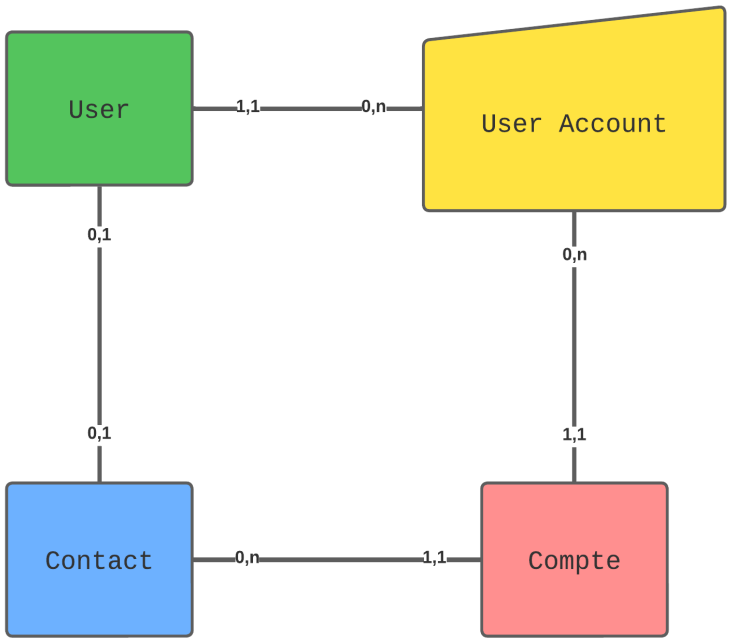
Une fois, tous les comptes et les contacts crée on se retrouve avec, une liste de tous les contacts et comptes créé :





Remarque : tous les contacts et comptes ne sont pas affichés par manque d'espace, le contact Test5UserAcc Five n'est par exemple pas affiché.

Voici un schéma résumant les liens entre les différents comptes, utilisateur, contact, User Account :



<u>Légende :</u>

0,1 : minimum 0 et maximum 1

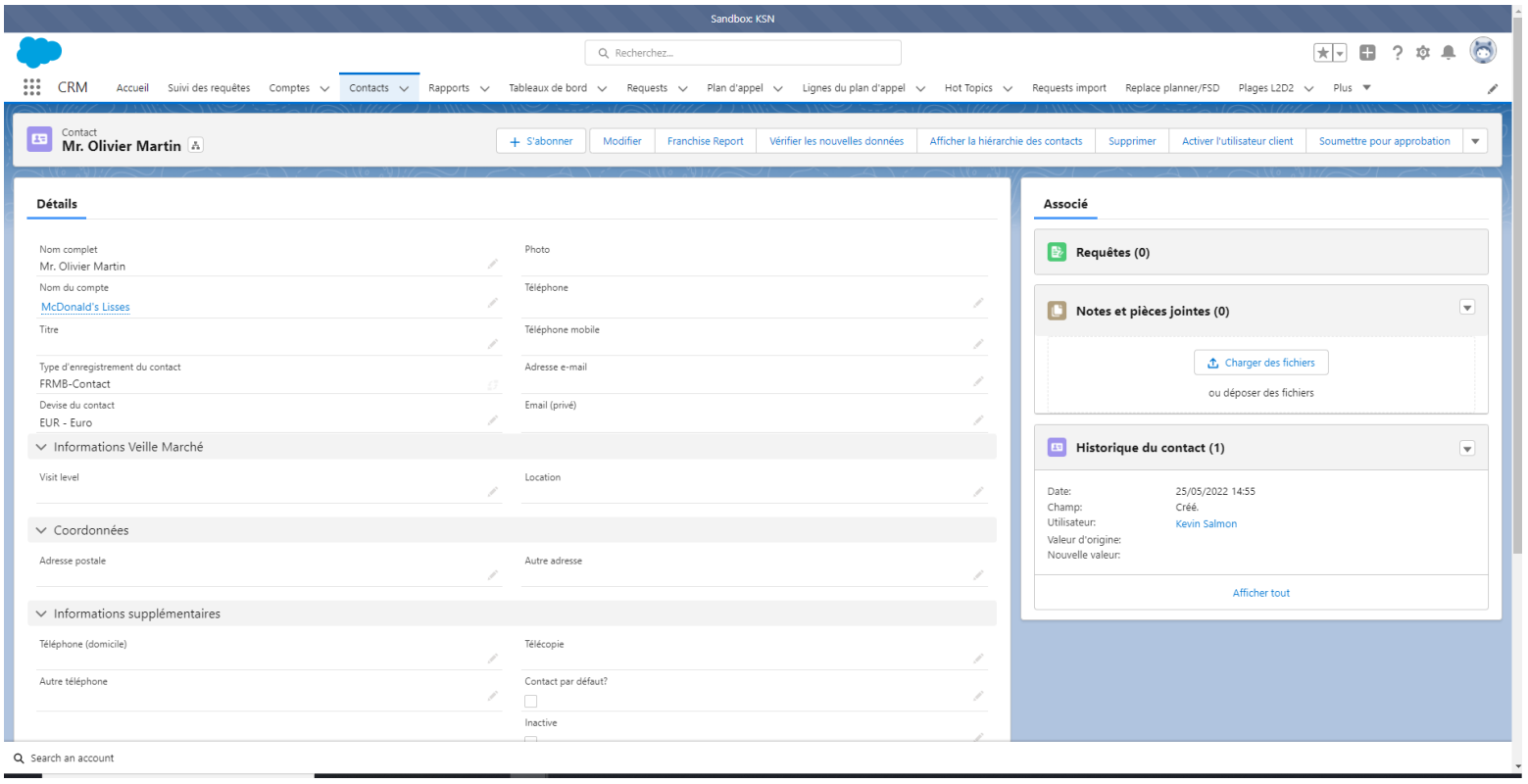
1,1 : minimum 1 et maximum 1

0,n : minimum 0 et maximum n (entier saisie)

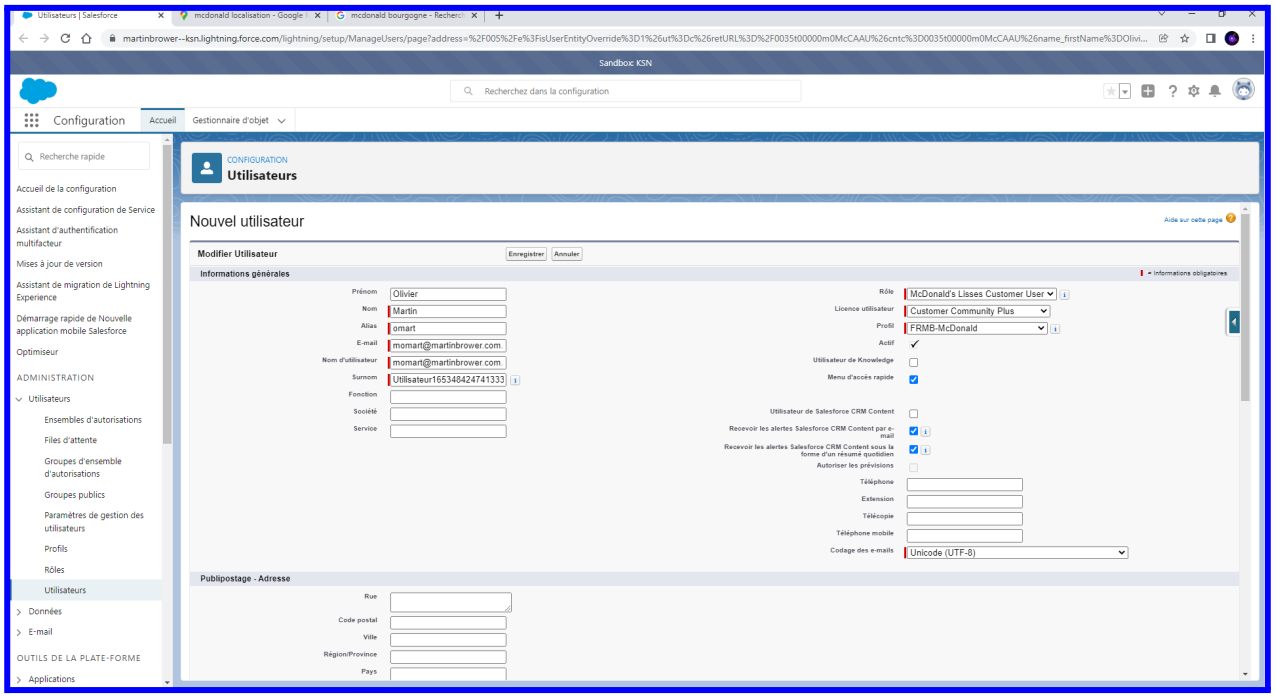
b.3. Création des utilisateurs associés aux contacts

Dans le schéma ci-dessus, deux termes sont nouveaux : User et User Account. Les utilisateurs clients sont de base désactivés pour les contacts. L'utilisateur client permet d'attribuer un user à un contact, pour qu'il disposent par exemple de plus de droits ou de plus de liberté, mais surtout lui attribuer un accès à Salesforce. Il pourrait par exemple avoir plusieurs User Account, contrairement à un simple contact.

On m'a donc demandé d'activer des utilisateurs clients pour chaque contact.



On prend donc exemple sur Olivier Martin, il suffit simplement d'appuyer sur l'onglet : ''Activer l'utilisateur client".



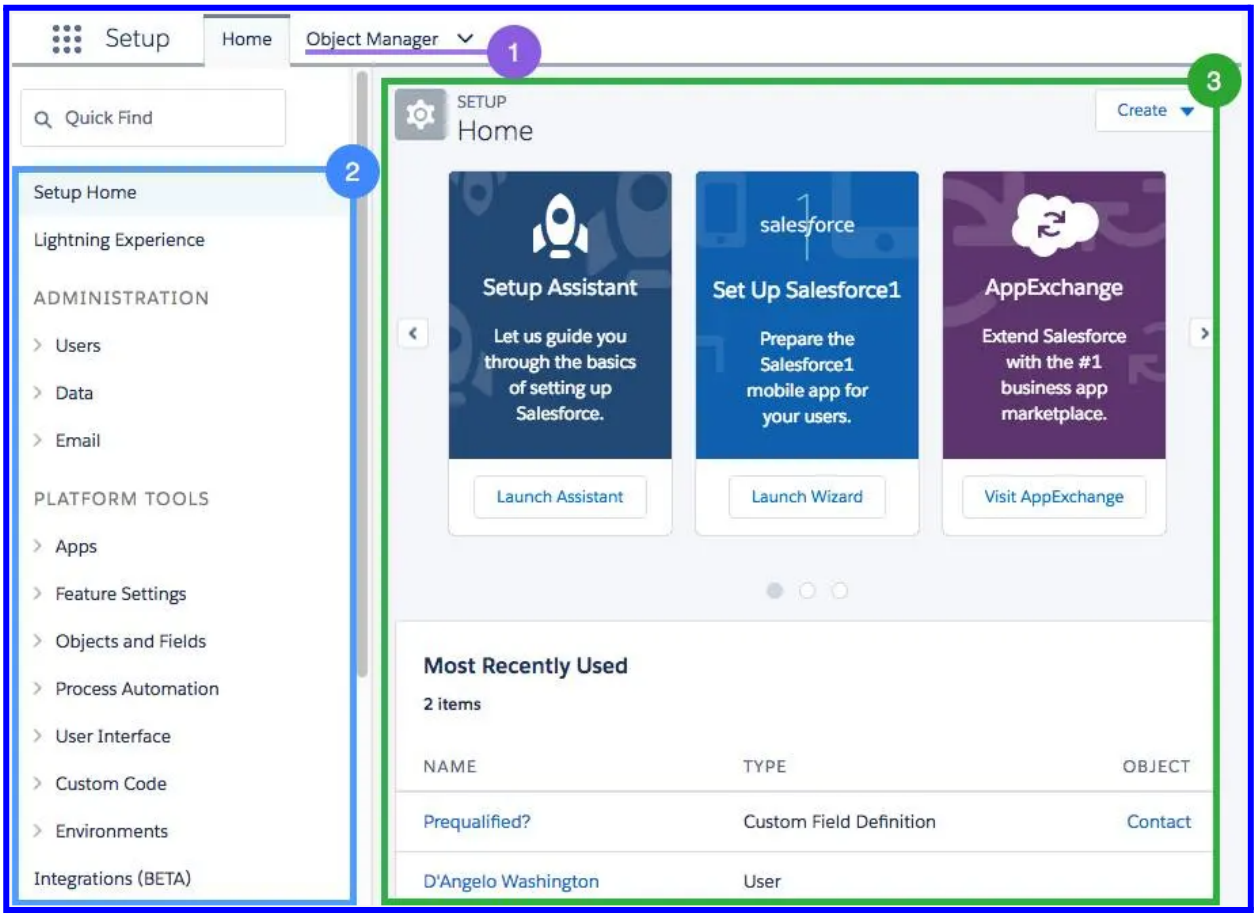
On arrive donc sur la page pour configurer l'utilisateur. Mais un problème apparaît lors de la création d'un user. En effet, il n'y à aucune restriction, pour filtrer le champ ''Pays". On ne peut donc pas avoir certaines informations précises sur la localisation de l'utilisateur. Cela pose donc problème pour l'identification de ce dernier.

b.4. Les règles de validation

C'est là qu'interviennent les règles de validations. Pour définir ces règles, cela demande un peu de codage. Une règle de validation peut contenir une formule ou une expression qui évalue si les données dans un ou plusieurs champs renvoie la valeur Vrai ou Faux. Ces règles permettent notamment de vérifier si un champ est ''null''. Les règles de validation incluent également un message d'erreur à afficher lorsque la règle renvoie la valeur TRUE (vraie) en raison d'une valeur incorrecte. Le message d'erreur est choisi au préalable par le créateur de la règle, mais il doit rester explicite et simple pour ne pas désorienter le client. Ces règles peuvent être définies avec les outils de Salesforce par défaut dans le menu "Setup" dans le coin supérieur droit de n'importe quelle page Salesforce :

- Le menu Setup est organisé en catégories basées sur des objectifs : Administrer, Créer, Déployer, Surveiller et Commander.

Représentation de la fenêtre Setup :

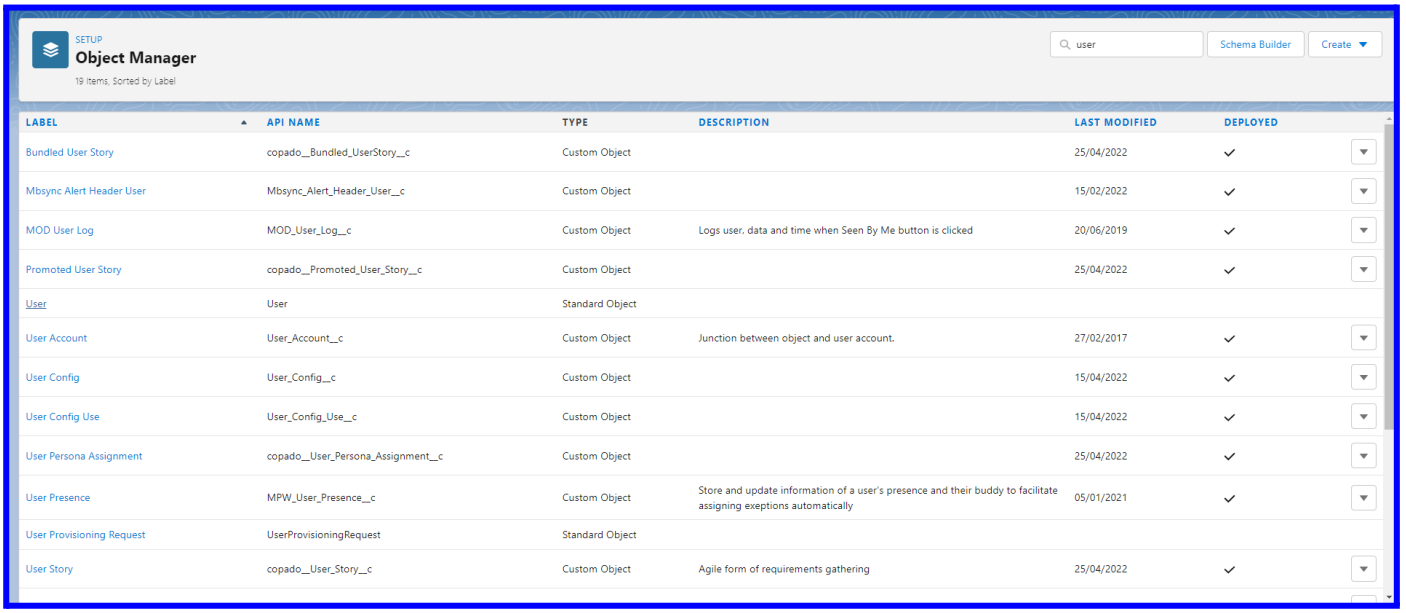


1. Gestionnaire d'objet : le gestionnaire d'objet permet de visualiser et de personnaliser les objets standard et personnalisés dans votre organisation. C'est d'ailleurs dans cet onglet qu'il faut se rendre pour les règles de validations.

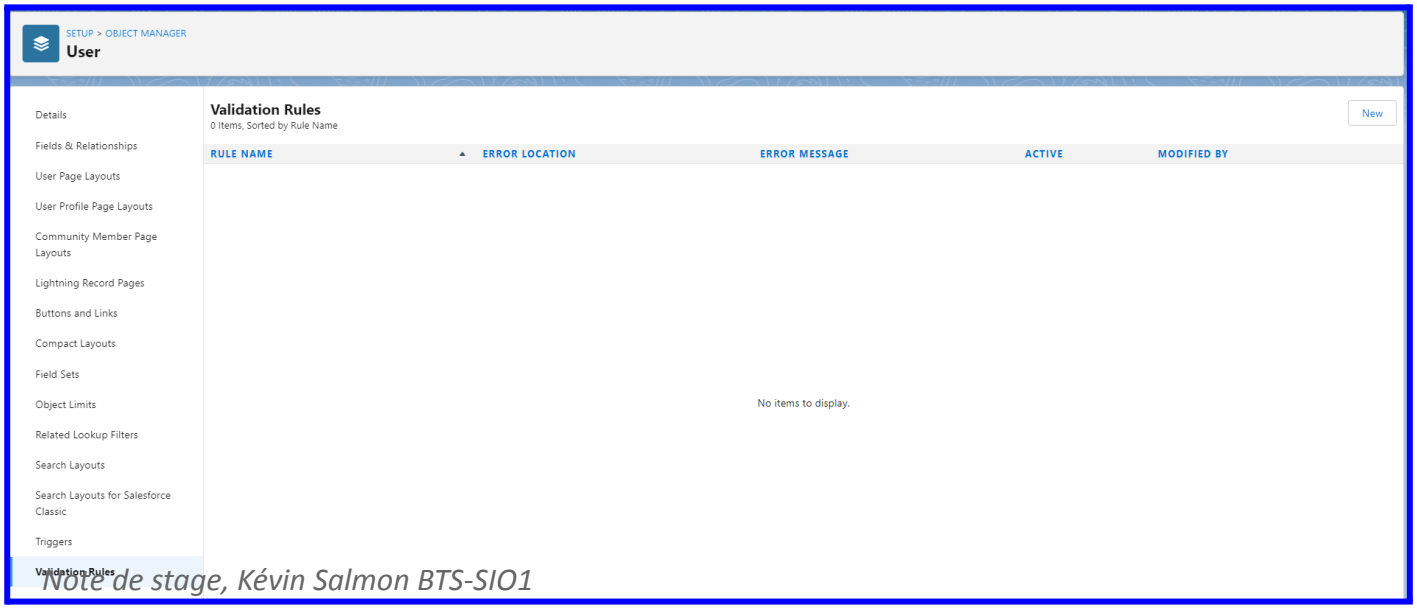
2. Menu de configuration : le menu inclut des liens rapides vers diverses pages qui permettent d'accomplir diverses tâches, de la gestion de vos utilisateurs à la modification des paramètres de sécurité.

3. Fenêtre principale : cela représente simplement la fenêtre principale, pour visualiser les éléments sur lesquels vous travaillez.

Une fois sur le gestionnaire d'objet, il faut rechercher l'élément sur lequel on veut instaurer des règles. Dans notre cas on recherche donc l'utilisateur pour la première règle, et le compte pour la deuxième règle.



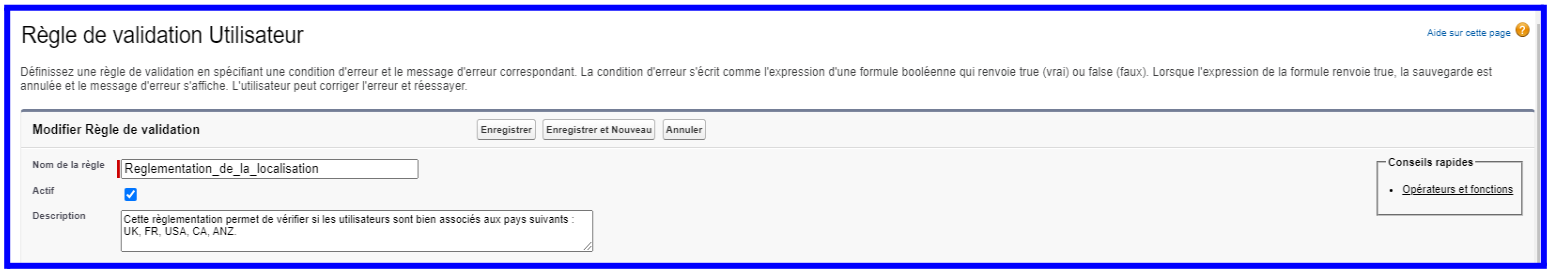
Une fois l'objet sélectionné, on arrive sur cette page où on peut voir en bas à droite les "validation rules".



Dans ce cas, j'ai utilisé ces règles pour répondre aux attentes de l'équipe. Une règle pour vérifier la localisation d'un utilisateur lors de sa création, et une autre règle qui permet de vérifier les champs : "Company ,Type, External_key", lors de la création d'un compte. Le but sera donc d'éviter les champs importants, vides. Et d'avoir une certaine mise en place de plusieurs restrictions pour définir des droits et des accès spécifiques.

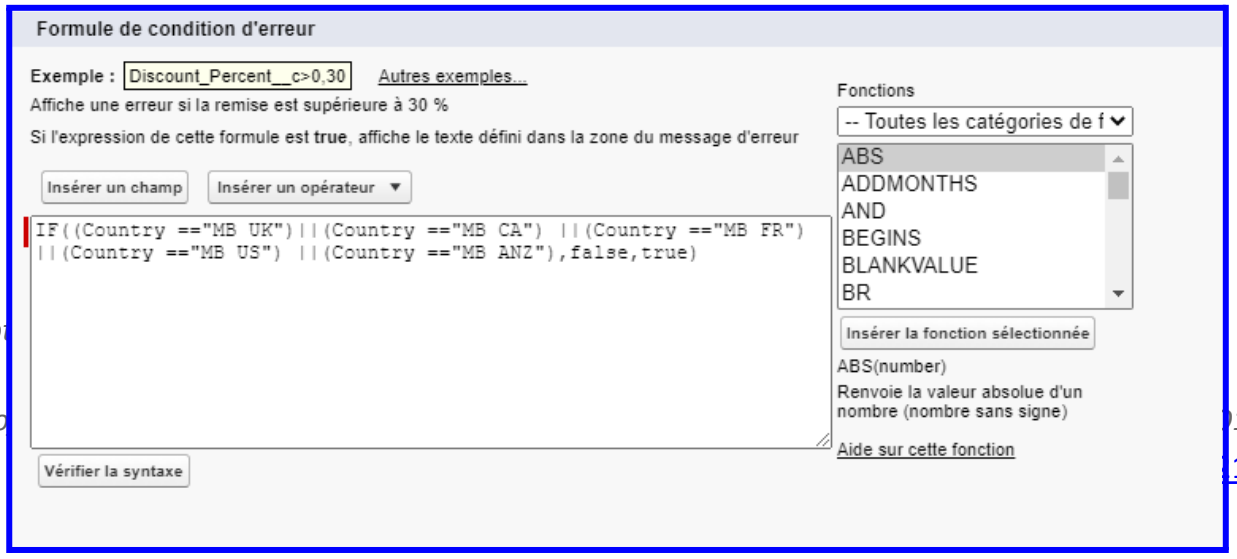
a) Première Règle :

Cela se présente donc comme ceci :



Une première partie où l'on donne le nom de la règle et sa description.

La deuxième partie qui est beaucoup plus orientée sur le codage et donc pouvoir écrire la méthode de la règle.

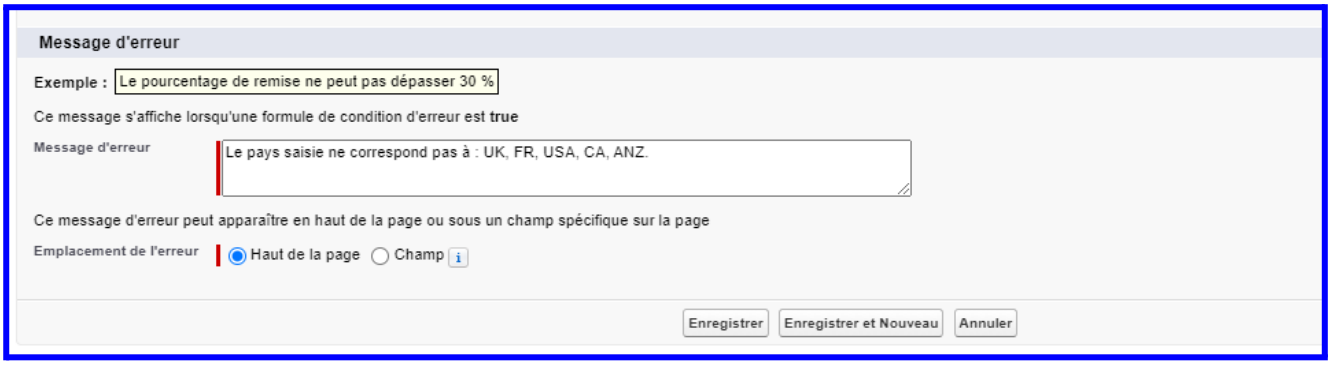


Not

Pro *01*

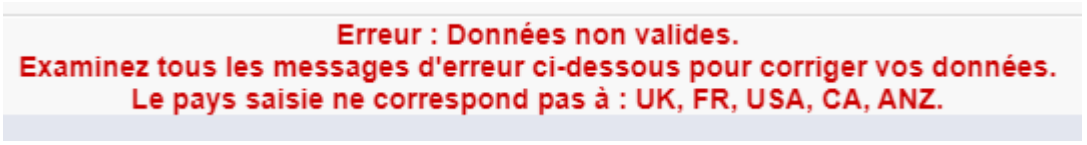
1

On m’a donc demandé pour la première règle de définir obligatoirement le pays d’un utilisateur. On prend donc le champ ‘’Country” qui doit être égale à : “MB UK, MB CA, MB FR, MB US, MB ANZ”. Si la condition est remplie, alors le résultat va être true, ce qui veut dire que le message d'erreur de la validation rules va apparaître, sinon non.



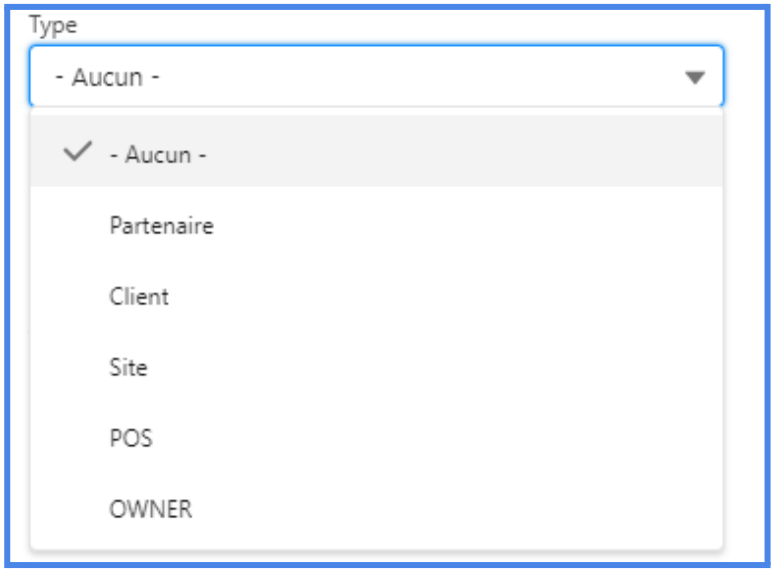
Sachant que le message d'erreur s’affiche quand la méthode renvoie True.

Si la règle n’est pas respectée, cela va donc afficher un message d'erreur :

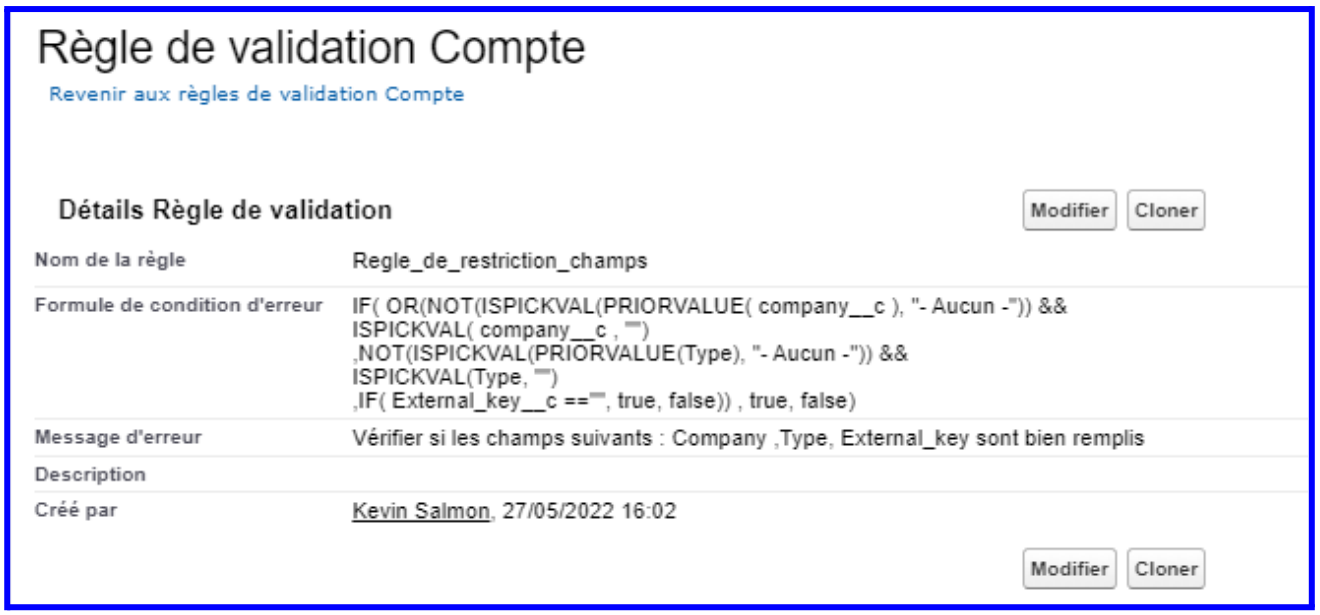


b) Deuxième règle :

Pour la deuxième règle, elle permet de vérifier si les champs : ”Company ,Type, External_key”, lors de la création d’un compte sont bien remplis. Petite particularité par rapport à la première règle, les champs ”Company ,Type” sont des listes déroulantes de ce type :



Pour ce faire, on utilise également le menu “Setup”.



Règle de validation Compte

Revenir aux règles de validation Compte

Détails Règle de validation — Modifier | Cloner

Nom de la règle	Regle_de_restriction_champs
Formule de condition d'erreur	IF(OR(NOT(ISPICKVAL(PRIORVALUE(company__c), "- Aucun -")) && ISPICKVAL(company__c , "") ,NOT(ISPICKVAL(PRIORVALUE(Type), "- Aucun -")) && ISPICKVAL(Type, "") ,IF(External_key__c =="", true, false)) , true, false)
Message d'erreur	Vérifier si les champs suivants : Company ,Type, External_key sont bien remplis
Description	
Créé par	Kevin Salmon, 27/05/2022 16:02

Modifier | Cloner

C’est donc une règle de restriction de champs, utilisant ce programme :

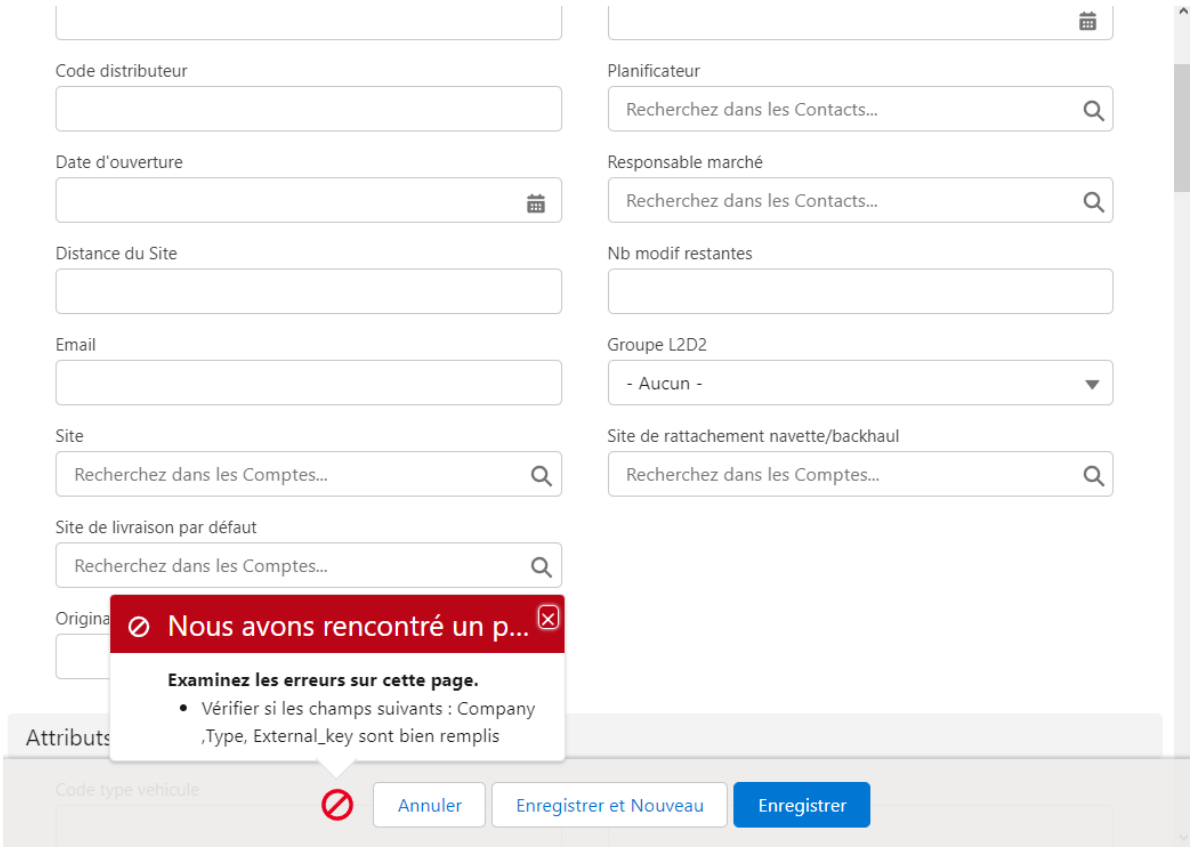
1. IF(OR(NOT(ISPICKVAL(PRIORVALUE(company__c), "- Aucun -")) &&
2. ISPICKVAL(company__c , "") , NOT(ISPICKVAL(PRIORVALUE(Type), "- Aucun -")) &&
3. ISPICKVAL(Type, ""), IF(External_key__c =="", true, false)) , true, false)

La règle de validation se déclenche si la valeur précédente de company__c n’était pas « - Aucun -» et que la valeur actuelle l’est. Car la valeur par défaut est « - Aucun -». En résumé, elles empêchent l'utilisateur de laisser sur “- Aucun -” la liste de sélection. C’est pour cela qu’on utilise ISPICKVAl, car c’est une fonction qui fonctionne seulement avec une liste de sélection. (ligne 1-2-3.)

Et pour le champ “External_key__c”, c’est exactement le même procédé que pour la première règle.

Il y a la présence du OR, car au départ, c'étaient trois règles séparées, mais grâce à la condition OR, elles sont réunies en une seule et même règle.

Si la règle n’est pas respectée, cela va donc afficher un message d'erreur :



c.1. Difficulté rencontrée

La principale difficulté que j'ai rencontrée durant le déroulement de cette mission est la prise en main de l'outil. Je pensais que cela serait beaucoup plus intuitif et rapide à prendre en main. Malgré la plateforme de formation de Salesforce et l'aide de mon maître d'apprentissage, il m'a tout de même fallu être extrêmement minutieux lors des configurations du CRM, et cela, afin que tout se passe correctement une fois déployé. Surtout, lors de la création des différentes règles, car ce n'est pas un langage qui ressemble à ce que j'ai pu étudier. Il m'a donc fallu chercher des solutions, notamment pour gérer le contenu d'une liste déroulante qui demandait une fonction précise. Cependant, à travers cette difficulté, j'ai pu être responsabilisé davantage par mon maître d'apprentissage afin de gagner en assurance dans la réalisation du projet.

d.1. Conclusion Déploiement d'une Sandbox :

Le déploiement d'une Sandbox a été la première mission à laquelle j'ai pris part durant ce stage d'apprentissage. Le choix de cette mission ne s'est pas fait de manière anodine. En effet, mon tuteur souhaitait que j'accompagne un projet de ses prémisses à son déploiement final en production.
De plus, travailler sur le CRM est une excellente manière de se former sur la plateforme Salesforce. Avant de vouloir développer sur cette plateforme, il faut comprendre son fonctionnement. Cette mission concerne en grande majorité la configuration sur Salesforce, ce qui pousse à étudier le fonctionnement de Salesforce et une partie de ses particularités. Ce projet m'a donc aidé à apprendre le fonctionnement en partie de la plateforme que je ne connaissais pas avant le début de mon apprentissage.

Pour conclure cette mission, je peux dire que cela a représenté une bonne initiation aux méthodes de travail de l'entreprise. Elle m'a aussi permis de devenir petit à petit plus autonome, et cela, afin de pouvoir travailler sur d'autres projets tout en gérant moi-même mon temps de travail.

2. Comment récupérer au préalable les comptes utilisateurs d'un contact choisi ? :

a.1. Contexte

Cette mission est un prolongement plus détaillé que la première mission. En effet, j'ai maintenant une sandbox, comportant 5 comptes et 5 contacts associés à un seul et même compte, mais j'ai aussi deux restrictions pour avoir notamment plus de champs de caractère remplis et assurer la traçabilité des utilisateurs et comptes.

Mais un problème se pose. En effet, l'équipe Salesforce m'expose des besoins et des attentes sur : *" Comment récupérer au préalable les comptes utilisateurs d'un contact choisi ? "*. Ils me

demandent donc plusieurs solutions fonctionnelles, qui pourraient répondre à cette problématique.
Car l'équipe aimerait améliorer la sélection d'user account pour créer un nouveau contact.
Dans ce cas, ils voudraient qu'on utilise directement un modèle d'un user disposant de plusieurs user account pour récupérer directement les user account et pouvoir les transmettre aux nouveau contact, au lieu de sélectionner les user account un par un.
C'est donc une demande d'optimisation de leur outil Salesforce.

Cette deuxième mission représente majoritairement des solutions dites fonctionnelles, puis par la suite après ma vidéoconférence des solutions techniques. Une solution fonctionnelle est une solution qui n'est pas totalement précise au niveau du code par exemple.

C'est donc une solution qui fonctionne dans la réalité, c'est-à-dire une solution réalisable ou non et capable de répondre aux besoins demandés. Elle diffère d'une solution technique, qui est beaucoup plus dans la précision, et qui permet d'assurer la faisabilité d'une solution.

Pour réaliser cette mission, on me demande d'utiliser le support que je souhaite pour pouvoir présenter mes solutions fonctionnelles à l'oral, à l'équipe SalesForce. Puis ensuite de les concrétiser avec une maquette et les coder. Pour que mes solutions soient donc maintenant techniques et non plus fonctionnelles.

b.1. Présentation de mes solutions :

J'ai donc présenté mes solutions dans un diaporama, elles sont au nombre de deux.

Ma première solution était d'utiliser des langages de programmation, proposés par Salesforces. J'ai choisi SOQL (variant du SQL) et le langage Apex (variant de Java). J'ai choisi cela car selon moi, grâce au SOQL, notamment, on pourra directement exécuter des requêtes précises depuis les bases de données, contenant les Comptes, Contacts,... Et donc, sélectionner l'élément dont on aura besoin.

Ma deuxième solution était d'utiliser les outils proposés par Salesforce. Je voulais également utiliser les règles de validation comme pour la mission 1. Mais ce ne fut pas très concluant, mais je leur ai quand même suggéré d'améliorer la problématique en y ajoutant des règles de validation. Pour notamment avoir une récupération plus précise.

Revenons donc à mon diaporama, j'ai tout d'abord expliqué l'existant et l'inexistant, et ce que j'envisageais d'apporter comme solution. La présentation s'est faite à la fois en vidéoconférence et en réel, car toute l'équipe n'était pas présente dans l'entreprise.

Durant cette vidéoconférence, je me suis donc mis d'accord avec mon maître de stage, sur les solutions exposer que je pourrais mettre en place pour répondre à cette problématique. Une seule des deux solutions à était retenue. Nous partons donc sur la solution utilisant le SOQL et le langage Apex, qui sont deux langages de programmation présents sur Salesforce.

Pour plus de détails, voici le lien de ma vidéoconférence sous Genially :
https://view.genial.ly/629082223a832e0011750221/dossier-copie-mobile-app-dossier

b.2. Le langage SOQL :

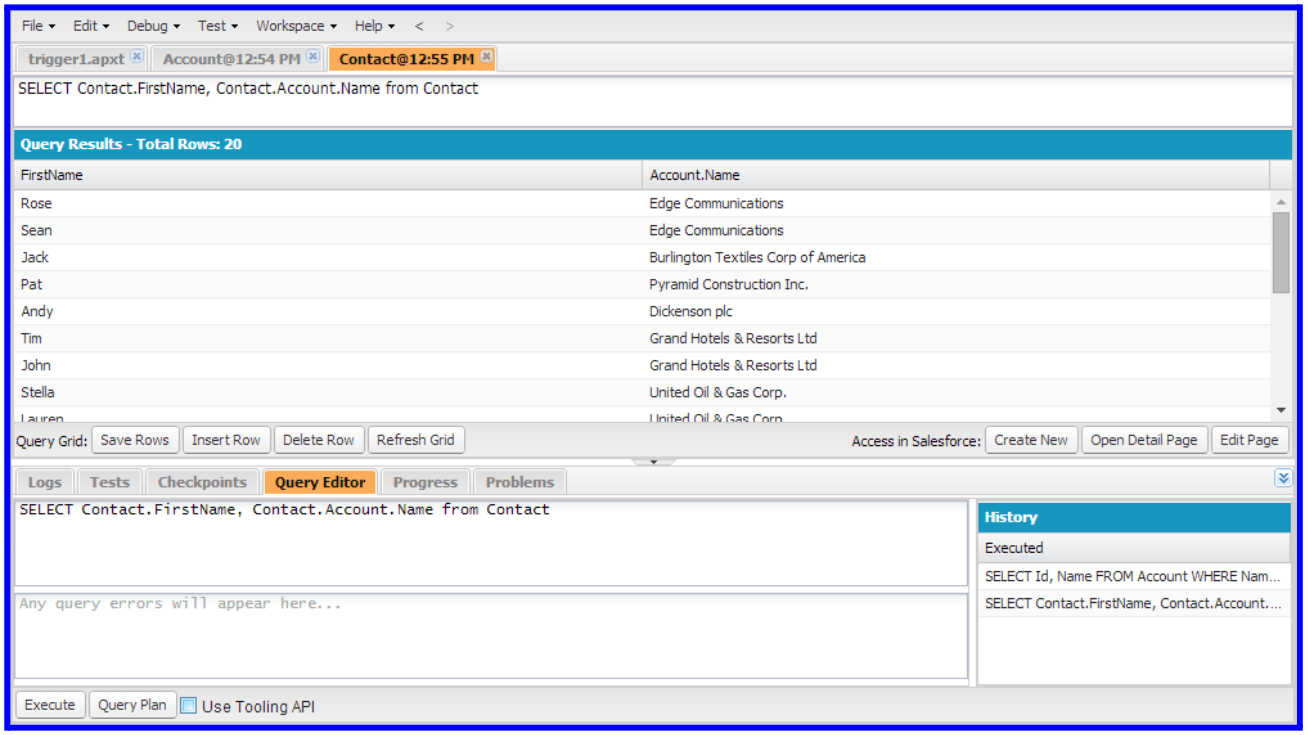
Avant de présenter le travail effectué sur la solution choisie, il est important de définir le langage SOQL, car il est primordial pour ma solution.
Le langage SOQL (Salesforce Object Query Language), est utilisé pour lire des enregistrements de données, directement dans des bases de données notamment.

Ce langage est un équivalent du SQL, car ils disposent également de requêtes comme le “SELECT” mais aussi plein d'autres mots-clefs. Mais ils sont différents dans leurs utilisations. En effet, le SQL récupère des données de une ou plusieurs tables dans des bases de données. Alors que le SOQL est utilisé pour récupérer des données d’un objet en particulier et de ses objets liés, comme par exemple “Account,User,...” C’est donc une variation du SQL propre à Salesforce.

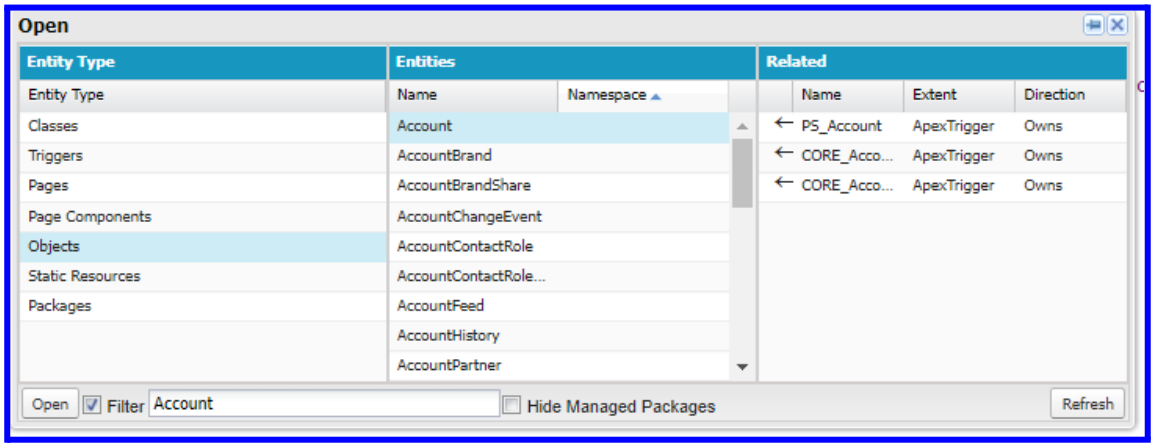
Pour utiliser le SOQl, il suffit de se rendre dans la “developer console”, qui est un environnement de développement intégré avec une collection d'outils que vous pouvez utiliser pour créer, déboguer et tester des applications dans votre organisation Salesforce.

Il suffit ensuite de se rendre dans l’éditeur de requête de la console développeur, pour y entrer directement les requêtes SOQL et pouvoir les exécuter.

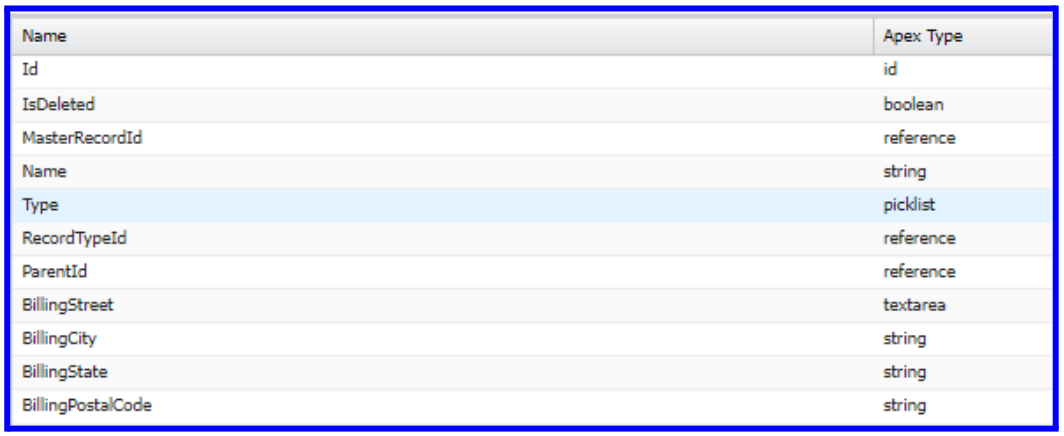
Cela se présente comme ceci :



Remarque : Pour mieux voir les ressources que j’ai à disposition dans la “developer console”, il suffit d’aller dans l’onglet “File”, puis dans “Open”. Cela permet de voir toutes les classes Apex, Pages, Objects, … J’ai donc utilisé cela pour savoir, où était stocké notamment, les Accounts, Les User,...



Je peux donc ouvrir la ressource “Account”. Cette étape n’est pas du tout obligatoire, mais cela me permet notamment de savoir quel champs il y à dans un “Account”.



Name	Apex Type
Id	id
IsDeleted	boolean
MasterRecordId	reference
Name	string
Type	picklist
RecordTypeId	reference
ParentId	reference
BillingStreet	textarea
BillingCity	string
BillingState	string
BillingPostalCode	string

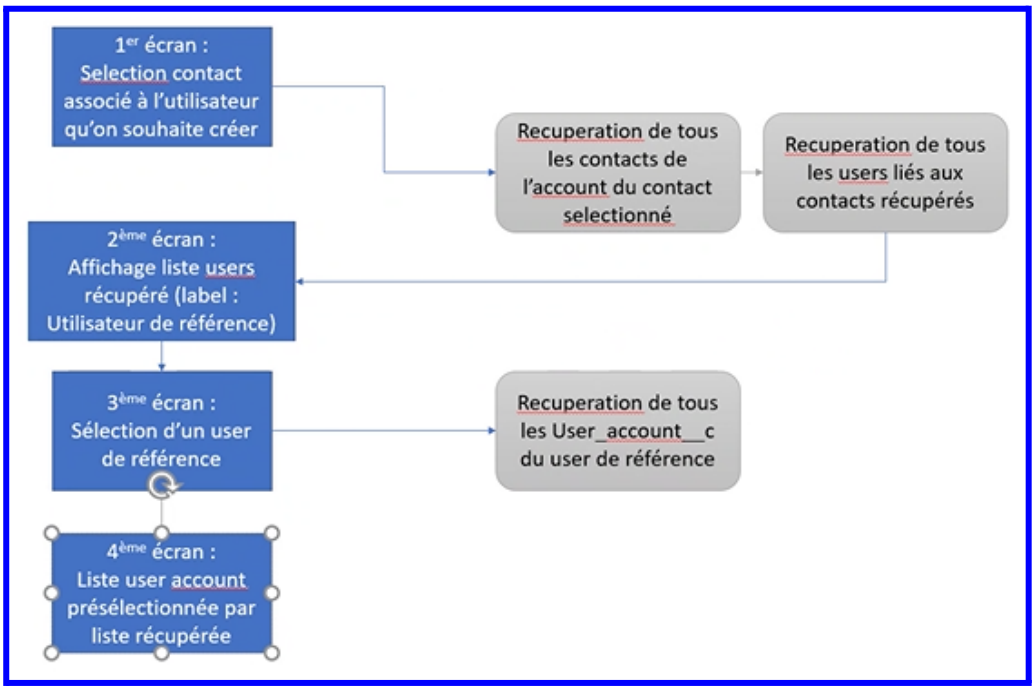
Je peux donc savoir quel champ me serait intéressant d’utiliser en requête SOQL, pour obtenir le résultat souhaité.

b.3. Maquettage :

Ma vidéoconférence est donc finie, et ma première solution à été choisie.
Mon maître de stage m’a donc demandé de faire une maquette, pour indiquer les actions que j'allais mettre en place, phase par phase, que ce soit au niveau du visuel et du codage avec

notamment des requêtes SOQL que je dois faire pour aller chercher l'information. Il faut donc mettre des captures d'écrans à chaque étape, pour montrer à quoi cela pourrait ressembler.

Représentation éphémère des consignes de la maquette :



Cette maquette ci-dessus me présente donc de manière générale les étapes que je dois effectuer pour poursuivre ma mission.

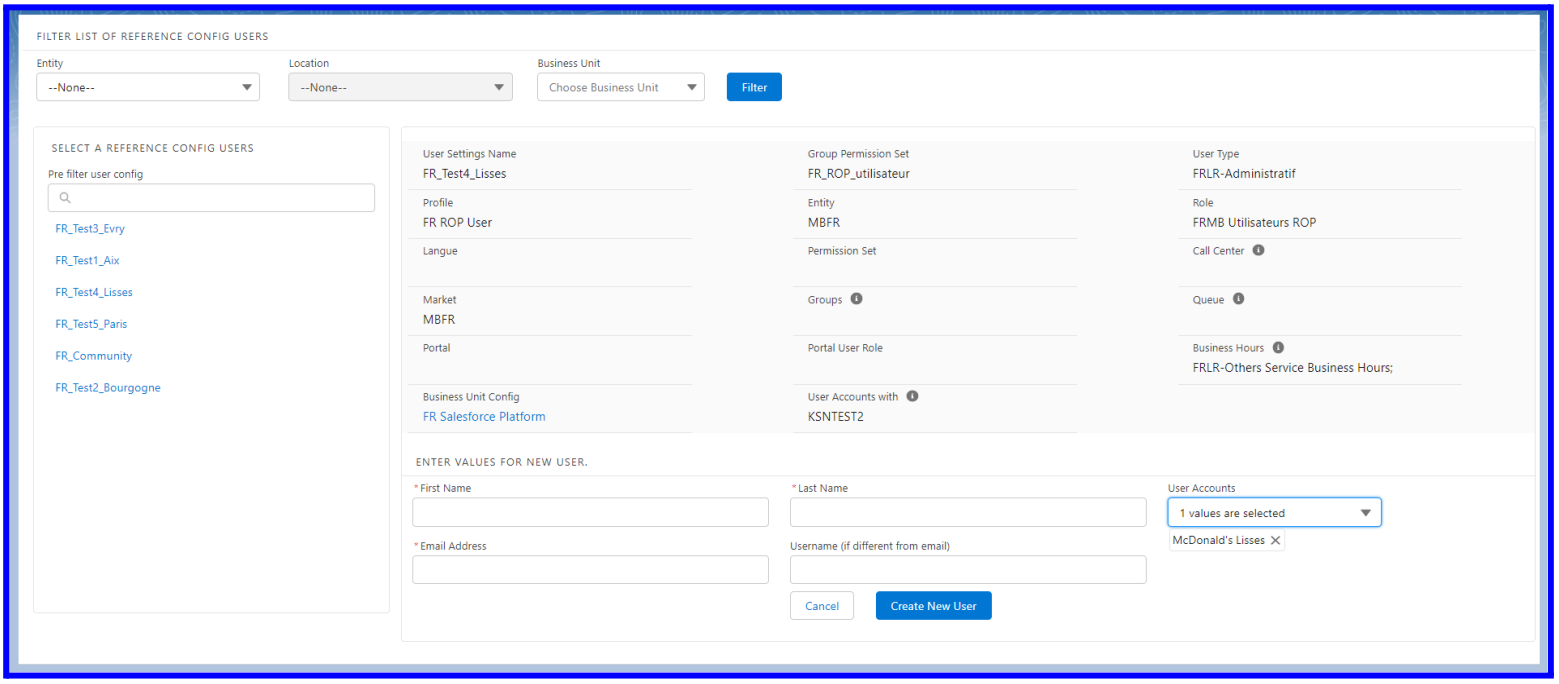
Tout d'abord, pour effectuer la première tâche qui est demandée, il faut que j'utilise dans Salesforce, l'application "User Manager Config'', pour mieux analyser les besoins qu'il me demande.

L'application User Manager permet donc :

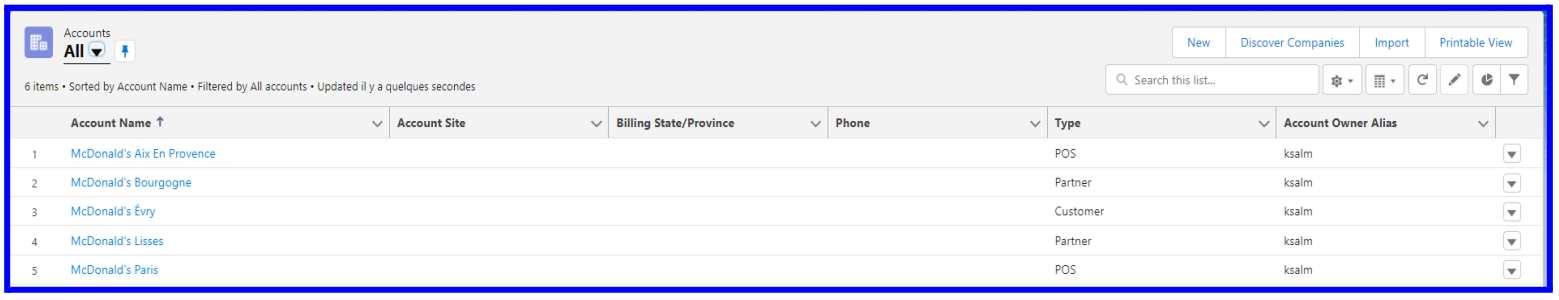
- D'avoir une gestion du profil et les droits d'accès de chaque utilisateur
- D'assurer le suivi des actions menées par chaque utilisateur

- De n’autoriser les utilisateurs qu’à consulter leurs propres transactions
- De gérer les droits d’accès au personnel temporaire.
- De configurer plusieurs utilisateurs sous un seul est même compte

On arrive donc sur cette page :



Sur cette page, on peut créer de base un nouvel user avec un ou plusieurs user account sélectionné. Un user account correspond à un restaurant. Dans le cas ci-dessus Mcdonald's Lisses est un user account, mais on peut en sélectionner d’autre parmi les comptes créés :



Un user account est donc par conséquent un restaurant associé à un ou plusieurs utilisateurs, et un utilisateurs peut aussi avoir 0 ou plusieurs user account.

Le problème qui se pose, c'est d'imaginer qu'on veuille créer un nouveau contact pour des besoins quelconque. Ce contact aurait comme nom par exemple : Nicolas Dupont. Nicolas Dupont sera donc associé obligatoirement à un compte (un restaurant), ce compte serait par exemple McDonald's Lisses. Nicolas Dupont serait donc un contact associé au compte McDonald's Lisses. Pour qu'un contact dispose de plus de droit et de liberté, on lui crée un utilisateur associé. Nicolas Dupont aura donc un utilisateur et depuis cette utilisateur, il pourra avoir un ou plusieurs user account. Mais imaginons qu'on souhaite que Nicolas Dupont soit associé à plus de 100 user account précis. Cela poserait donc un problème de vitesse d'exécution car à l'heure actuelle, on doit sélectionner les user account manuellement, depuis la liste User Account sur la capture d'écran. C'est pour cela que l'équipe Salesforce, m'a donné comme problématique: *"Comment récupérer au préalable les comptes utilisateurs d'un contact choisi ? : "* pour cette mission. Car on souhaite en résumé, utiliser un modèle d'un user ayant lui des user account, pour directement copier ses user account à lui et pouvoir les intégrer directement aux nouveaux contacts.

J'ai donc commencé par faire les requêtes SOQL, pour mieux récupérer les informations souhaitées.

Pour ce faire, on souhaite d'abord récupérer tous les contacts d'un compte associés au contact choisi au départ. Par exemple, nous choisissons comme contact : "Jean Pierre". Ce contact est associé au compte McDonald's Lisses. Nous souhaitons alors récupérer tous les comptes associés à McDonald's Lisses.

J'ai donc utiliser cette requête SOQL : SELECT Name, (SELECT Name FROM Contacts) FROM Account WHERE Id IN (SELECT AccountId FROM Contact WHERE ID='0035t00000m0MgSAAU')

L'id : '0035t00000m0MgSAAU' correspond à l'id du contact Jean Pierre.

Qui affiche ce résultat :

Query Results - Total Rows: 1

Name	Contacts
McDonald's Lissess	[{"Name":"Olivier Martin"},{"Name":"Jean Pierre"},{"Name":"Marie Laure"},{"Name":"Pierre Smith"},{"Name":"Marie-Fred Think"},{"Name":"Test Condit...

Cela affiche bien tous les contacts associé au comptes : “McDonald’s Lisses”

On souhaite maintenant récupérer tous ceux qui ont des utilisateurs parmi la liste des contacts renvoyés. On souhaite seulement les utilisateurs, car seuls les utilisateurs peuvent avoir un ou plusieurs user accounts.

J’ai donc utiliser cette requête SOQL :

```
SELECT Name, (SELECT Name FROM Users) FROM Account WHERE Id IN (SELECT AccountId FROM Contact WHERE ID='0035t00000m0MgSAAU')
```

L’id : '0035t00000m0MgSAAU' correspond aussi à l’id du contact Jean Pierre.

Qui affiche ce résultat :

Query Results - Total Rows: 1

Name	Contacts
McDonald's Lissess	[{"Name":"Olivier Martin"},{"Name":"Jean Pierre"},{"Name":"Marie Laure"},{"Name":"Pierre Smith"},{"Name":"Marie-Fred Think"},{"Name":"Test Condit...

À première vue, cela semble afficher le même résultat, et c’est tout à fait normal. Car lors de la création des contacts associés à McDonald’s Lisses, j’ai défini comme quoi tous les contacts disposaient d’un utilisateur.
J’ai quand même fait un test en créant un contact n’ayant pas un utilisateur associé, au cas où cela ne fonctionnerait pas. Et cela fonctionne bien, car il n'apparaît pas dans la liste.

Nous avons donc maintenant la liste de tous les utilisateurs associés au comptes du contact choisi.

Il faut donc maintenant récupérer tous les UserAccounts (s'il y en a) de l'utilisateur référence choisi parmi la liste précédente.

J'ai donc utilisé cette requête SOQL : select User__c,User__r.name FROM User_Account__c

Qui affiche ce résultat :

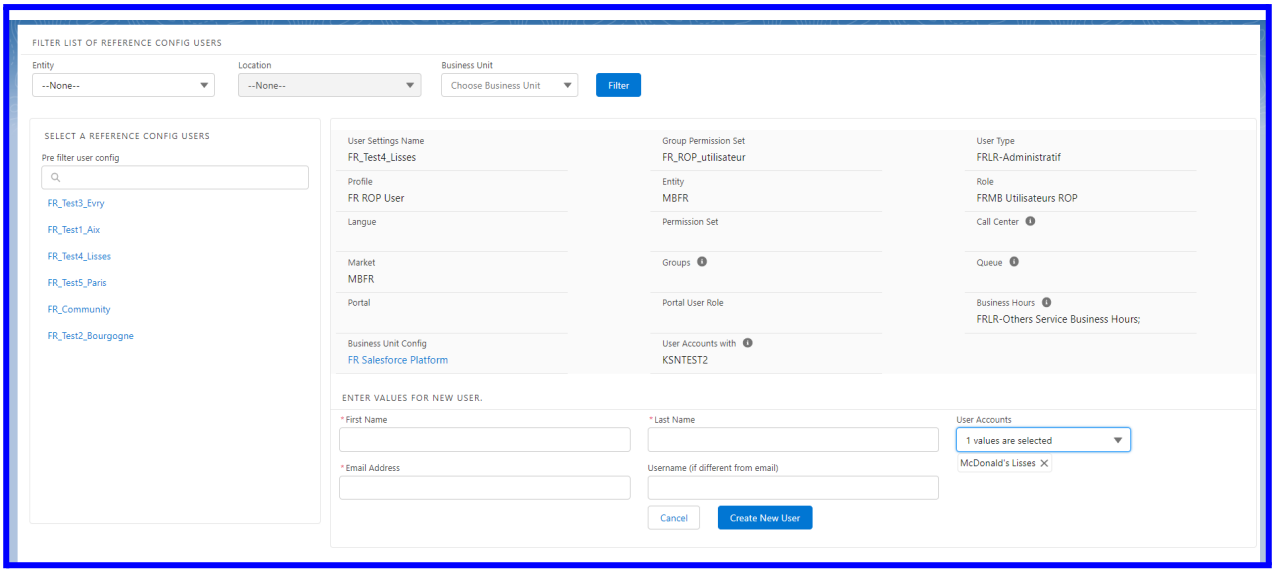
Query Results - Total Rows: 5

User__c	User__r.Name
0055t000001D4oaAAC	Test5UserAcc Five
0055t000001D4oaAAC	Test5UserAcc Five
0055t000001D4oaAAC	Test5UserAcc Five
0055t000001D4oaAAC	Test5UserAcc Five
0055t000001D4oaAAC	Test5UserAcc Five

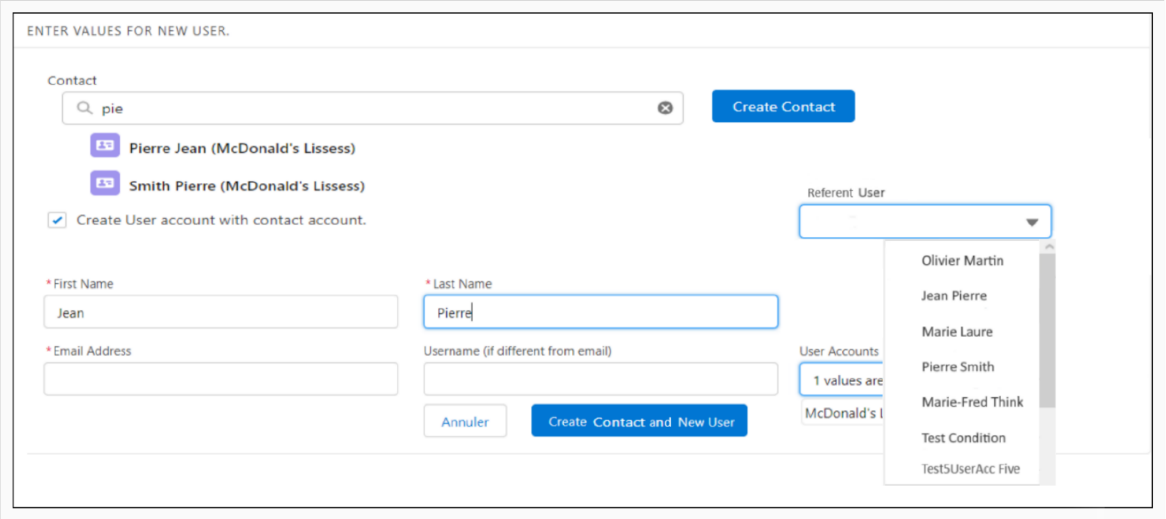
Cela affiche donc cinq fois la même chose, ce qui veut dire que l'utilisateur Test5userAcc Five disposent de 5 User Account. Pour récupérer le nom du user account, se sera fait plus tard de manière automatique.
J'ai donc maintenant écrit et testé toutes les requêtes SOQL imbriquées.

Une fois toutes les requêtes SOQL écrites, nous pouvons commencer à modifier sur paint ou un autre outil similaire, l'affichage de l'écran qui sera impacté par ces requêtes, pour essayer de donner le visuel que cela pourra avoir sur la page de base.

Ce sera donc cette écran :



Comme dit précédemment, nous souhaitons récupérer la liste de tous les utilisateurs, associée au compte du contact sélectionner. J'ai donc imaginé ce visuel pour correspondre à la demande :

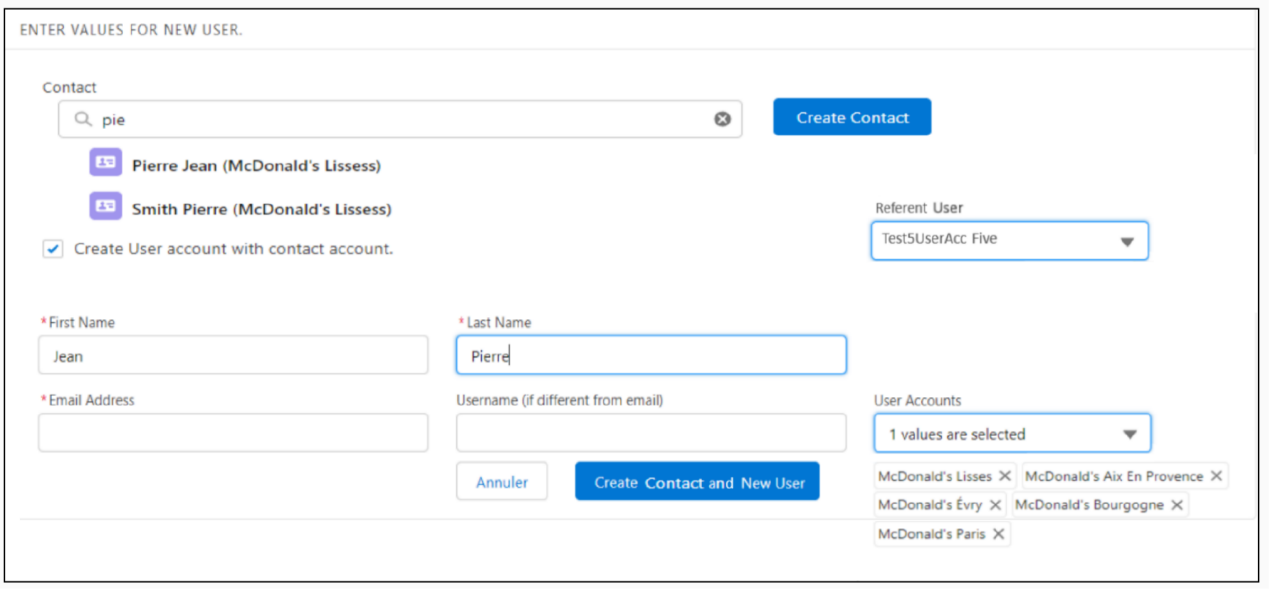


Cette image ci-dessus, est donc un visuel imaginé par moi-même, il n'est en rien un visuel totalement définitif, c'est seulement une maquette de ce que ça pourrait donner.

J'ai donc imaginé une liste, qui apparaîtrait dans le champ Referent User, cette liste montre tous les utilisateurs associé au compte du contact sélectionner. Nous pouvons voir d'ailleurs, que cela

correspond bien avec le résultat de la deuxième requête SOQL exécuter. J'ai appelé cette liste, Referent User, car le but étant de choisir un utilisateur de référence. J'ai mis le titre en anglais, car l'outil est totalement en anglais, et il pourrait donc y avoir confusion.

Nous passons maintenant à l'étape suivante :



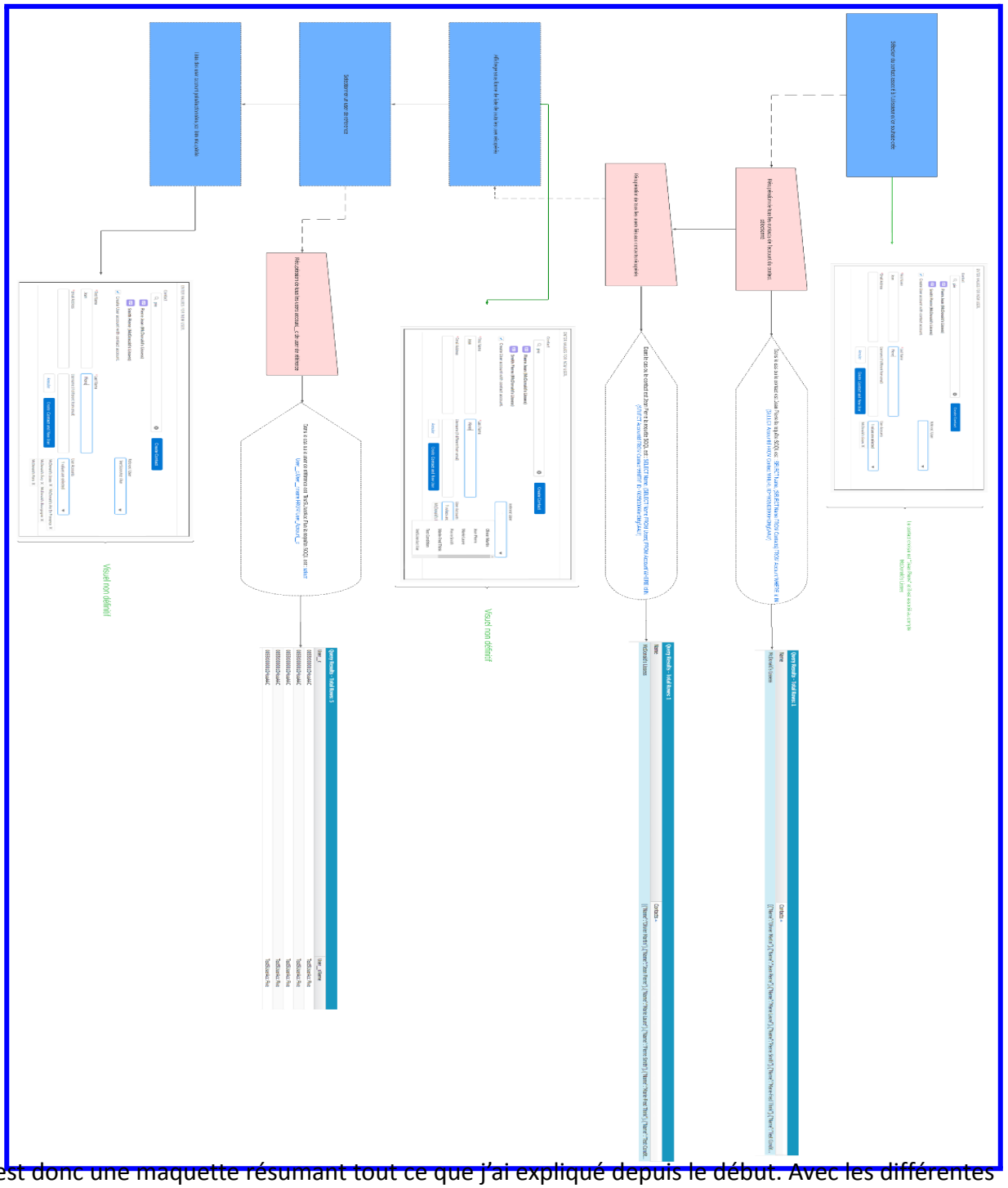
Donc pour cette image, j'ai aussi imaginé le visuel, quand on sélectionne justement l'utilisateur référence. Dans ce cas, j'ai imaginé qu'on utilisait l'utilisateur : "Test5User Five". Car c'était pour le moment, le seul disposant de plusieurs user account, et aussi car j'avais pris exemple sur ce contact, lors de la requête SOQL.

Donc là, une fois qu'on à sélectionner le contact "Test5User Five", nous pouvons voir que le champ user account c'est actualiser, en mettant les user account de "Test5User Five". Les user account sont donc, McDonald's Lisses, McDonald's Évry, McDonald's Bourgogne, McDonald's Paris et McDonald's Aix En Provence. Nous pouvons donc maintenant créer, un contact et un utilisateur, disposant des mêmes User Account que "Test5User Five". Il n'y à donc plus besoin de sélectionner manuellement les user account, lors de la création d'un nouveau contact et utilisateur.

J'ai donc maintenant toutes les captures d'écrans qui pourraient montrer à quoi cela va ressembler, et aussi toutes les requêtes SOQL correspondantes.

Je peux donc maintenant faire une maquette avec un diagramme, pour réunir toutes les informations dont je dispose. À savoir, que j'ai fait plusieurs versions de la maquette, pour avoir le meilleur résultat possible, et répondre aux demandes de l'équipe.

Voici la version finale de la maquette :



C'est donc une maquette résumant tout ce que j'ai expliqué depuis le début. Avec les différentes requêtes SOQL, les différents affichages imaginés et les résultats des requêtes SOQL.

Voici le lien de la maquette :
https://drive.google.com/file/d/1C5Gy5_938MeOw4l_mNMcY3QCDO1FdED4/view?usp=sharing

b.4. Renouvellement de la solution :

Une fois la maquette finie, je me suis mis d'accord avec mon maître d'apprentissage et l'équipe Salesforce pour voir si la maquette correspondait à leurs attentes. La maquette à donc été validée, et j'ai pu continuer le projet.

Mais, il y à un problème qui se pose, en théorie vu que je suis sur ma sandbox, il y à peu de Contacts, Account, … Cela ne pose donc pas de problème pour exécuter les commandes.
Mais si je mettais en place les requêtes de types imbriqués que j'ai mise en place, directement dans la sandbox de l'entreprise, cela pourrait fonctionner aujourd'hui, mais pas demain.
Car déjà une requête imbriquée n'est pas conseillée, car elle demande beaucoup d'informations en même temps en une seule requête.
Mais aussi, le plus gros problème, c'est tout simplement Salesforce. Car les serveurs de l'entreprise stockant des données sont gérées directement par Salesforce. Salesforce fait donc office en quelque sorte de drive. En échange de payer l'application Salesforce, cette dernière fournit tous les outils Salesforces et un espace de stockage mais avec certaines limitations.
Ces limitations sont mises en place par Salesforce pour assurer une certaine performance et flexibilité pour l'entreprise.
Avec plus de 8000 contacts, accounts, …, il faut donc penser à bien gérer les ressources, pour ne pas subir ses limitations imposées par Salesforce.
C'est d'ailleurs pour cela que Salesforce dans les requêtes SOQL, interdisent le caractère "*", pour éviter de sélectionner tous les champs d'un objet et donc être bridé par la plateforme.
La limite est actuellement à 50 000 au niveau des ressources.

Avec l'exécution de mes requêtes nous sommes arrivés, vers 46 450 sur 50 0000.
Cela semble être une bonne marge, mais pas du tout. Il y à sans cesse de nouveaux contacts, accounts, …, qui sont créés au quotidien. Cela risque donc de poser un sérieux problème dans le futur.

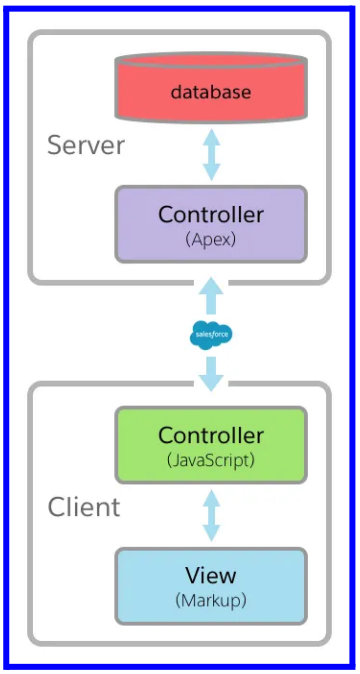
J'ai donc réfléchi avec le responsable de la plateforme des données, à une solution pour résoudre ce problème.
Avant toute chose, je lui ai présenté mon projet, mes objectifs, ma maquette, …
Il m'a donc conseillé de lier le langage Apex et SOQL en même temps. Cela va permettre notamment de mieux récupérer les données, mais aussi pouvoir stocker les requêtes SOQL dans des variables, et pouvoir donc réduire les informations sélectionner des requêtes SOQL.

b.5. Le langage Apex :

Le langage Apex est un langage propre à Salesforce, c'est un langage de programmation orienté objet. Il ressemble beaucoup au langage Java. Cela permet notamment de contrôler des pages de Salesforce, ou stocker sous forme de liste des requêtes. On peut y créer notamment des événements, sur une page web. Ce langage correspond donc à la logique d'une page Salesforce.

L'avantage de ce langage, c'est qu'il est facile d'utiliser, car il utilise déjà des procédés de Java bien connus, tels que la syntaxe des variables et des expressions. Il peut être utilisé avec SOQL, pour avoir un côté plus axé sur les données. Il propose des procédures pour stocker certaines requêtes. Ce langage est également facile à exécuter, car on est pas obligé de créer une classe Apex pour pouvoir exécuter le programme.

Pour utiliser Apex, il suffit de se rendre aussi dans la "developer console".
Il y à deux moyens pour exécuter un code Apex. Le premier moyen est sans doute le plus compliqué, car il faut créer une classe pour ensuite mettre en place le code sur une page web, pour voir s' il y à eu un résultat sur cette dernière. La plupart du temps, il n'y à pas besoin de créer une classe, car il y a déjà toutes les classes correspondant à chaque page web de Salesforce. En général, on crée une nouvelle classe, quand on veut faire également une nouvelle page sur Salesforce.
Donc si on veut simplement exécuter le code, sans vouloir créer une classe ou le mettre dans une classe déjà existante.
On peut utiliser le menu "Debug" pour ouvrir une page annexe qui exécute le code.



Voici un schéma résumant la place du langage Apex dans Salesforce. Nous pouvons bien voir qu'il est côté serveur. Il est donc du côté Backend avec SOQL. C'est-à-dire le côté non visible par le client.

b.5. Lien entre Apex et SOQL :

Nous avons donc réfléchi, étape par étape, au code Apex, exactement comme pour les requêtes SOQL.

Nous avons donc commencé par créer une liste sur Apex.
Les listes peuvent contenir un ou plusieurs objets. Les listes sont donc synonymes avec les tableaux.

La liste se nomme "selectedcontacts" car on vient sélectionner tous les contacts associés aux comptes du contact choisis. Mais pour ce faire il faut déjà sélectionner le contact qui va être choisi, on n'a donc aussi créé une liste qui se nomme : "selectedcontact".

```
1  list<contact> selectedcontact = [select accountid from contact where id='0035t00000m0MgSAAU' ];
2
3  ID acctidselectedcontact= selectedcontact[0].accountid;
4  System.debug('acctidselectedcontact '+acctidselectedcontact);
5
6  list<contact> selectedcontacts = [select id,name from contact where accountid=:acctidselectedcontact];
7
8
9  System.debug('selectedcontacts '+selectedcontacts);
```

La ligne 3 permet de reprendre la première la requête et de luis ajouter l'account id.

Ensuite pour afficher le résultat, il suffit d'utiliser "System.debug('...')".

Remarque : pour créer une liste il faut utiliser cette commande : list<*> nomdelaliste =

* Il faut mettre à l'intérieur contact, si le from de la requête est contact. Il faut donc mettre l'objet sur lequel on va récupérer l'information.

```
10
11
12  list<user> selectedusers =[select id,name from user where contactId in :selectedcontacts];
13
14
15
16  System.debug('selectedusers '+selectedusers);
17
```

On crée ensuite, une liste qui se nomme “selectedusers” pour sélectionner tous les utilisateurs associés à la liste “selectedcontacts”.

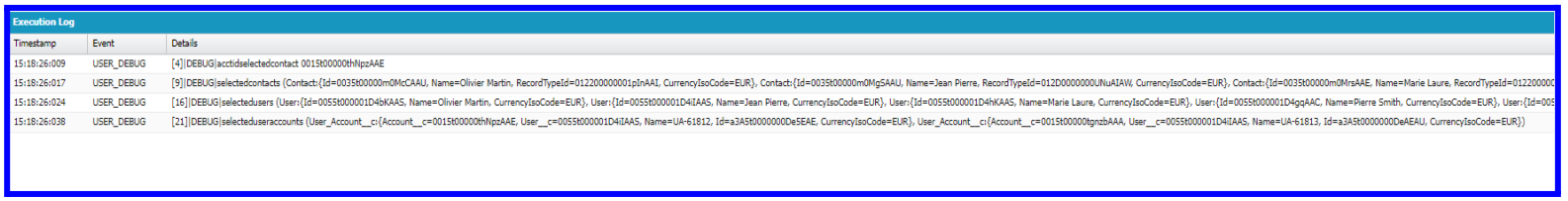
Remarque : On est pas obligé de sélectionner le nom, c’est surtout utile pour celui qui va faire le code, ou le regarder, ou vérifier si le code fonctionne, car des noms sont évidemment plus parlant qu’une id.

```
19  list<user_account__c> selecteduseraccounts=[Select account__r.name,account__c,User__r.name,name  from user_account__c where User__c in : selectedusers];
20
21  System.debug('selecteduseraccounts '+selecteduseraccounts);
```

On vient ensuite sélectionner tous les user accounts, et l’enregistrer aussi dans une liste qui se nomme : “selecteduseraccounts”.

On n’a donc, en résumé, optimiser toutes les requêtes SOQL mises en place par moi précédemment. En enregistrant notamment les requêtes à chaque fois.
Le but étant de condenser mon travail, pour assurer une meilleure flexibilité.

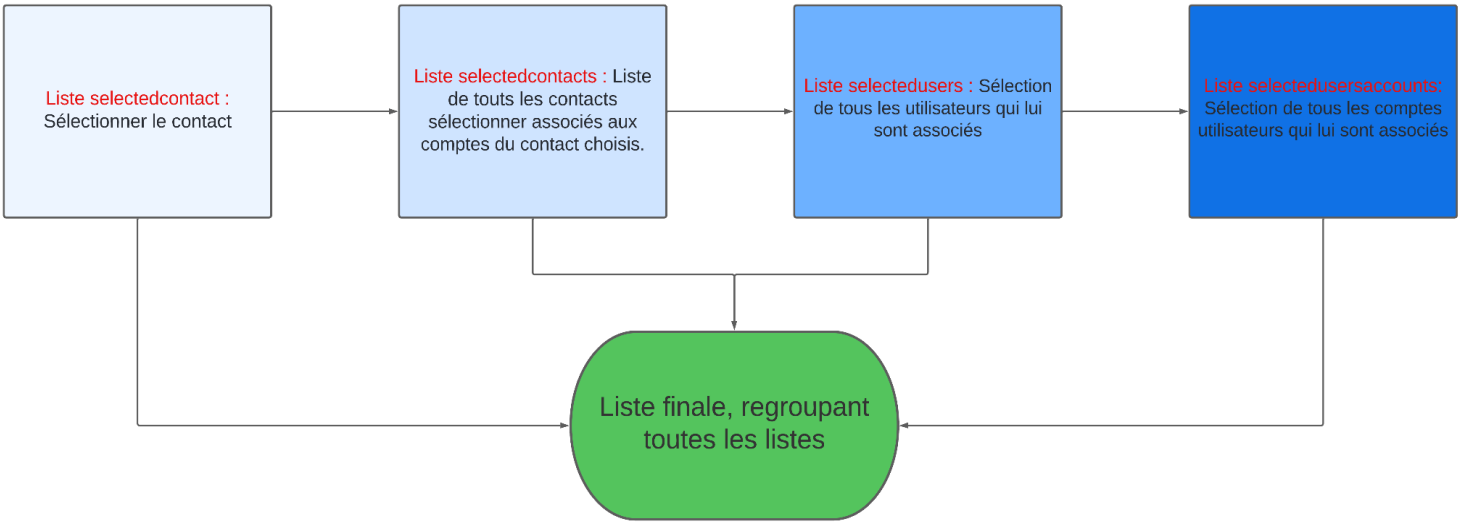
Nous allons donc maintenant exécuter le programme :



Execution Log

Timestamp	Event	Details
15:18:26:009	USER_DEBUG	[4]|DEBUG|acctidselectedcontact 0015t00000thNpzAAE
15:18:26:017	USER_DEBUG	[9]|DEBUG|selectedcontacts (Contact:{Id=0035t00000m0McCAAU, Name=Olivier Martin, RecordTypeId=012200000001pInAAI, CurrencyIsoCode=EUR}, Contact:{Id=0035t00000m0MgSAAU, Name=Jean Pierre, RecordTypeId=012D0000000UNuAIAW, CurrencyIsoCode=EUR}, Contact:{Id=0035t00000m0MrsAAE, Name=Marie Laure, RecordTypeId=012200000
15:18:26:024	USER_DEBUG	[16]|DEBUG|selectedusers (User:{Id=0055t000001D4bKAAS, Name=Olivier Martin, CurrencyIsoCode=EUR}, User:{Id=0055t000001D4iIAAS, Name=Jean Pierre, CurrencyIsoCode=EUR}, User:{Id=0055t000001D4hKAAS, Name=Marie Laure, CurrencyIsoCode=EUR}, User:{Id=0055t000001D4gqAAC, Name=Pierre Smith, CurrencyIsoCode=EUR}, User:{Id=005
15:18:26:038	USER_DEBUG	[21]|DEBUG|selecteduseraccounts (User_Account__c:{Account__c=0015t00000thNpzAAE, User__c=0055t000001D4iIAAS, Name=UA-61812, Id=a3A5t0000000De5EAE, CurrencyIsoCode=EUR}, User_Account__c:{Account__c=0015t00000tgnzbAAA, User__c=0055t000001D4iIAAS, Name=UA-61813, Id=a3A5t0000000DeAEAU, CurrencyIsoCode=EUR})

Cela fonctionne donc bien, et donne le même résultat qu'avec seulement les requêtes SOQL.



On se retrouve donc avec quatre listes différentes codées en Apex et qui contient des requêtes SOQL. Le but envisagé serait de réunir ces quatres liste en une seule seulement pour avoir qu'un seul affichage au final.

b.6. Visual Studio Code :

Visual Studio Code est un éditeur de code gratuit, multiplateforme, ultra rapide et léger développé par Microsoft pour Windows, Linux et OS X. Il est compatible avec de nombreux langages de programmation : (C/C++, Java, PHP, Python, Javascript, …).

Visual Studio Code intègre la plupart des fonctionnalités de base d'un éditeur de texte, dont la coloration syntaxique personnalisable, un système de plugins…

C'est donc un éditeur de code incontournable, pour les développeurs de Salesforce.

L'éditeur propose cependant plusieurs fonctions avancées qui sont intéressante, dont :

- Minimap : prévisualisation de tout le fichier dans une barre latérale ou du code simplement,
- Marque-page au sein même des fichiers,
- Sauvegarde automatique,
- Personnalisation des raccourcis clavier.

Lien :

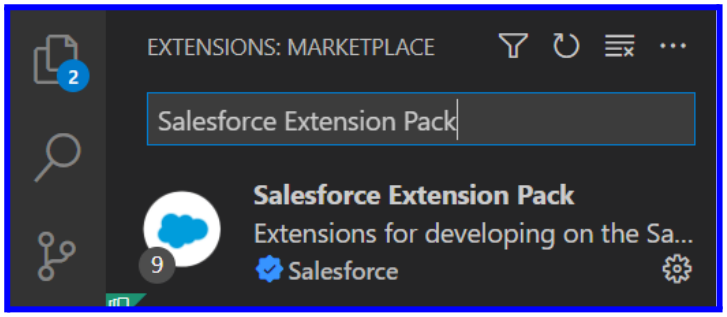
Installation de Visual Studio Code : Visual Studio Code - Code Editing. Redefined

L'avantage de ce logiciel, c'est qu'on puisse ajouter une ou plusieurs extensions, parmis le marketplace proposer par le logiciel : Lien du Marketplace
On peut donc installer, ou mettre à jour certains modules spécifiques.

Et on va justement utiliser ces extensions, pour intégrer l'outil Salesforce dans le logiciel.

Pour configurer Visual Studio Code avec Salesforce, il faut tout d'abord installer l'interface de ligne de commande (cli) : https://developer.salesforce.com/tools/sfdxcli
Cela va permettre d'avoir toutes les commandes de types "sfdx …".

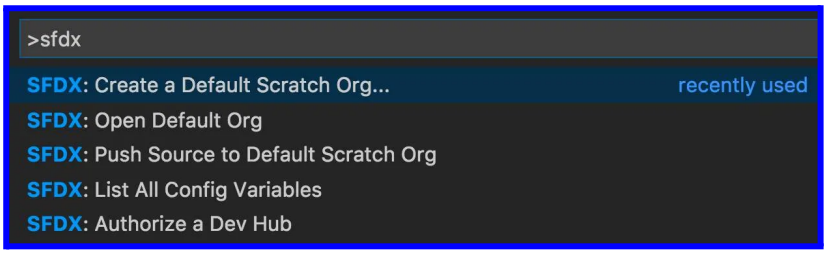
Ensuite, il faut se rendre dans Visual Studio Code et installer le pack d'extensions Salesforce. Il faut donc rechercher l'extension : "Salesforce Extension Pack"



Remarque : Assurez-vous d'avoir activé l'autorisation de modifications des extensions Salesforce. Car au départ cette option est souvent désactivée pour des raisons de sécurité.

Il faut maintenant tester l'extension, pour voir si cela est bien configuré. Pour ce faire nous ouvrons la palette de commande de Visual Studio Code en appuyant sur **Ctrl+Maj+P** (Windows) ou **Cmd+Maj+P** (macOs).

Il faut ensuite écrire sfdx :



Affichant donc toutes les commandes disponibles de l'extension.

Il faut donc créer un nouveau projet, c'est dans ce projet qu'il y aura toutes les pages Salesforces, ou travaux effectués.

Une fois fait, il faut exécuter cette commande dans le terminal de Visual Studio Code pour faire lien avec Salesforce :
sfdx auth:web:login --setalias "nom_de_la_sandbox"--instanceurl "url_de_la_page_Salesforce"--setdefaultusername

Une fois la commande exécutée cela ramène sur la page de connexion Salesforce. Une fois connecté, Visual Studio Code sera donc en lien avec la Sandbox utilisée.

Si cela ne fonctionne pas il faut :

-Dans Visual Studio Code, ouvrir la palette de commandes en appuyant sur Ctrl+Maj+P (Windows) ou sur Cmd+Maj+P (macOS).

-Saisir SFDX puis sélectionnez SFDX: Authorize an Org.

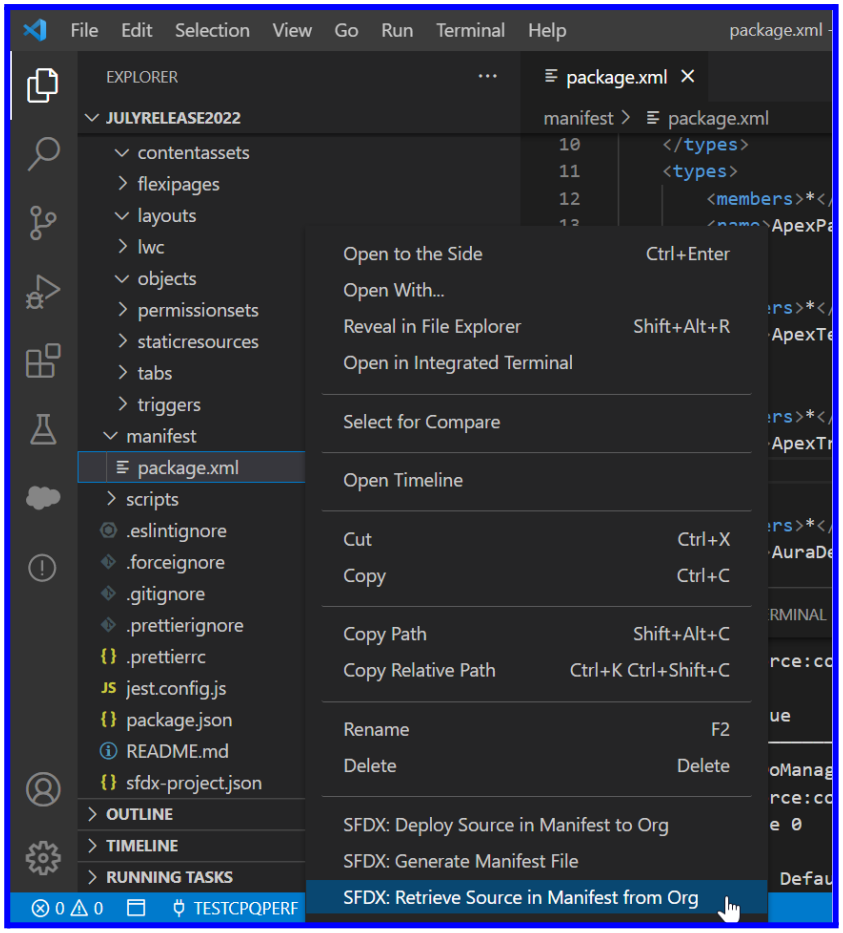
-Appuyez sur Entrée pour accepter l'option URL de connexion par défaut du projet.

-Appuyez sur Entrée pour accepter l'alias par défaut.

-Cette action ouvre la connexion Salesforce dans une autre fenêtre du navigateur. Il faut ensuite se connecter avec vos identifiants.

Il faut donc maintenant importer tous les LWC (Lightning Web Components), les composants Web Lightning sont des éléments HTML personnalisés créés à l'aide de HTML, JavaScript modernes, XML et du CSS. Cela représente entre autres, toutes les pages internet de l'outil Salesforce. On peut donc gérer ces pages et modifier par exemple la logique de ces dernières.

Pour importer tous les LWC, il faut se diriger dans l'onglet fichier et chercher le package : "Package.xml ". Ensuite faire un clic droit et sélectionner l'onglet : "SFDX:Retrieve Source in Manifest from Org".



Il y à mainteant tous les fichiers lwc qui sont directement intégrés dans Visual Studio Code.

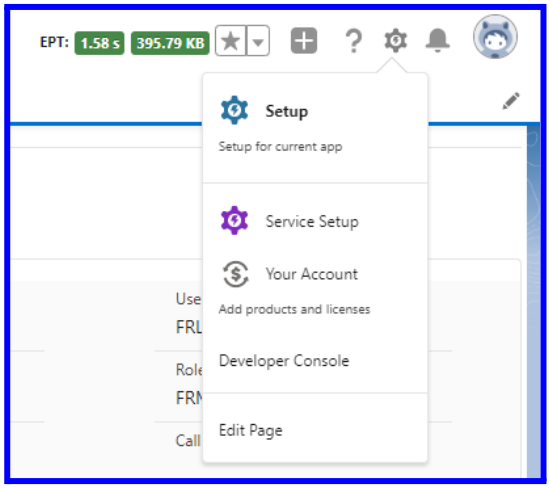
On peut donc maintenant faire le lien entre l'apex et visual studio code.

b.7. Lien entre Apex et Visual Studio Code :

Il faut donc maintenant essayer de trouver une solution pour ajouter toutes les choses établies dans la maquette et pouvoir injecter également le code Apex.

Pour ce faire, il faut d'abord trouver la classe Apex qui est attribuée à la page qu'on veut changer.

Pour faire cela il suffit d'abord de se rendre sur la page concernée puis de se rendre dans cette fenêtre, et ensuite dans l'onglet "Edit Page" :

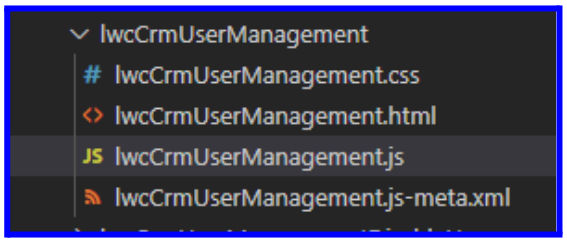


Après il suffit simplement de sélectionner l'élément qui nous intéresse.

Et nous pouvons voir à droite de la page le nom "lwcCrmUserManagement", qui correspond au nom de la classe Apex correspondant à cette dernière.

Il faut donc essayer de trouver dans Visual Studio code, le lwc correspondant à ce nom.

Nous trouvons donc ce dossier comportant, un fichier css, html, JavaScript, XML.



Le XML permet d'identifier, organiser et migrer des composants de métadonnées. Les métadonnées font référence à plusieurs composants de l'organisation.

Le html permet de construire la page web Salesforce.

Le css permet d'apporter du style à la page web, même si cela est gérer la plupart du temps par le html dans ce cas la, en utilisant notamment des composant déjà existantes ou coder par certaines personnes de l'organisation. Ces composants permettent donc de faciliter la tâche du développeur mais aussi d'optimiser le programme.

Le Javascript permet de donner une certaine logique à la page web. Mais aussi de pouvoir faire le lien avec le langage apex. À la base JavaScript était seulement un langage de script simple. Surtout que pendant longtemps il n'y à eu de mise à jour, ni même d'améliorations quelconque. Ce qui cause en quelque sorte un désintéressement de ce langage. Mais depuis l'arrivée de la version ES6, les choses deviennent totalement différentes.Dans ce cas, on utilise donc du Javascript dit moderne. Car elle utilise donc la version ES6, qui à totalement révolutionné ce langage et redonner de l'intérêt pour ce dernier.

On peut prendre par exemple sur les fonctions fléchées, qui ont révolutionné le langage.

```
/*Expression de fonction classique :
let somme = function(a, b) {
   return a + b;
};
*/

//Équivalent en fonction fléchée :
let somme = (a, b) => a + b;
```

C'est donc pour cela que la société utilise du Javascript moderne.

Une fois le fichier trouvé, je remarque qu'il faut importer les méthodes Apex dans le fichier Javascript :

```
import { LightningElement, track, wire, api } from 'lwc';
import { NavigationMixin, CurrentPageReference } from 'lightning/navigation';
import retrieveBUConfig from '@salesforce/apex/GEN_UserManagement_Ctrl.retrieveBUConfig';
import retrieveUserConfig from '@salesforce/apex/GEN_UserManagement_Ctrl.retrieveUserConfig';
import verifyUserData from '@salesforce/apex/GEN_UserManagement_Ctrl.verifyUserData';
import getAccountsDC from '@salesforce/apex/GEN_UserManagement_Ctrl.getAccountsDC';
import createUserFromUserConfig from '@salesforce/apex/GEN_UserManagement_Ctrl.createUserFromUserConfig';
import getInfoByUser from '@salesforce/apex/GEN_UserManagement_Ctrl.getInfoByUser';
```

Il viennent tous du fichier apex : "GEN_UserManagement_Ctrl", les méthodes apex associé à cette page sont donc tous dans ce même fichier.

Le problème c'est qu'il n' y à aucune méthode Apex que j'ai créé dans ce fichier . J'avais fait seulement un programme dans le menu Debug, que j'ai présenté ici : b.5. Lien entre Apex et SOQL :.

Le programme que j'ai fait n'est donc pas encore dans une classe, et c'est tout à fait normal, car je voulais seulement tester le programme.
Il faut donc se rendre dans le fichier apex trouver.

Pour cela il faut aller dans la console de développement, puis transformer mon programme en plusieurs méthodes distinctes.

Mon programme de base est donc comme ça :

```
list<contact> selectedcontact = [select accountid from contact where id='0035t00000m0MgSAAU' ];

ID acctidselectedcontact= selectedcontact[0].accountid;
System.debug('acctidselectedcontact '+acctidselectedcontact);

list<contact> selectedcontacts = [select id,name from contact where accountid=:acctidselectedcontact];


System.debug('selectedcontacts '+selectedcontacts);


list<user> selectedusers =[select id,name from user where contactId in :selectedcontacts];



System.debug('selectedusers '+selectedusers);


list<user_account__c> selecteduseraccounts=[Select account__r.name,account__c,User__r.name,name  from user_account__c where User__c in : selectedusers];

System.debug('selecteduseraccounts '+selecteduseraccounts);
```

Comme nous pouvons le voir j'ai décidé de le décomposer en trois méthodes distinctes.

1. La première méthode permet de gérer la sélection du contact, et affiche toute la liste des contacts du compte associés au contact choisi. J'ai donc choisi comme nom de méthode : "getRefrenceContact". Les commentaires que j'ai écrits en oranges résument la méthode que j'ai faite.

```
    /**
    * @function : getReferenceContacts
    * @param: id saisieid
    * @description : allows you to return the list of contacts of an account, corresponding to a selected contact
    * @return : list of selectedcontacts
    */
    @AuraEnabled(cacheable=true)
    public static list<contact> getReferenceContacts(id saisieid){

        list<contact> selectedcontact = [select accountid from contact where id=: saisieid];
        ID acctidselectedcontact= selectedcontact[0].accountid;

        System.debug('acctidselectedcontact '+acctidselectedcontact);
        list<contact> selectedcontacts = [select id, name from contact where accountid=:acctidselectedcontact];

        System.debug('selectedcontacts '+selectedcontacts);

        return selectedcontacts;

    }
```

J'ai aussi mis en paramètre : "saisieid", cela représente tout simplement l'id du contact qui va être sélectionné. Et le mot-clef "ID" qui ce situe derrière, est un type qui est défini par Salesforce, ce type est utilisé pour spécifier seulement un ID. C'est le même principe que pour un String, spécifiant une chaîne de caractère. Cela permet d'éviter certains problèmes lors de la lecture de l'id du contact.
Pour le reste du programme je l'ai déjà expliqué dans cette partie : b.5. Lien entre Apex et SOQL :.
La seule chose aussi qui peut différer, c'est le return, mais c'est un mot-clefs qui sera présent dans toute mes méthodes, elle permet simplement de renvoyer le résultat de la méthode.

2. Pour la deuxième méthode, elle permet de référencer tous les utilisateurs associés à la liste des contacts d'auparavant. Puis, elle renvoie simplement le résultat. Il y à aussi une présence des commentaires…

```
/**
 * @function : getReferenceUsers
 * @param: List<contact> selectedcontacts
 * @description : allows to return the list of users, among the list of contacts of the method getRefrenceContacts
 * @return : list of selectedusers
 */
@AuraEnabled(cacheable=true)
public static list<user> getReferenceUsers(List<contact> selectedcontacts){

    list<user> selectedusers =[select id, name from user where contactId in :selectedcontacts];

    return selectedusers;
}
```

Je prend en paramètre cette fois ci : "selectedcontacts" , qui serait donc la liste de tous les contacts. Son type est d'ailleurs une liste de contacts. On regarde donc si parmi la liste des contacts, générés dans la méthode 1, il y à des contacts qui ont des utilisateurs.
Pour plus d'informations sur le programme, se référer à ce lien : b.5. Lien entre Apex et SOQL :

3. Pour la troisième méthode, elle permet de référencer tous les user accounts, parmi l'utilisateur de référence sélectionner dans la liste précédente. Puis il y a également des commentaires…

```
/**
* @function : getReferenceUsersAccounts
* @param: List<user> selectedusers
* @description : returns the list of users accounts, among the list of users of the method getReferenceUsers
* @return : list of selecteduseraccounts
*/
 @AuraEnabled(cacheable=true)
public static list<user_account__c> getReferenceUsersAccounts(Id selecteduser){

    list<user_account__c> selecteduseraccounts=[Select account__r.name,account__c,User__r.name,name  from user_account__c where User__c = :selecteduser];

    return  selecteduseraccounts;
}
```

Je prend en paramètre cette fois ci : "selecteduser" , qui serait donc la liste de tous les utilisateurs. Son type est un Id. On regarde donc si parmi la liste des utilisateurs, générés dans la méthode 2, il y à des utilisateurs qui ont des User Accounts.
Pour plus d'informations sur le programme, se référer à ce lien : b.5. Lien entre Apex et SOQL :

J'ai donc transformé mon programme de départ en trois méthodes distinctes, pour assurer une certaine performance mais aussi une meilleure utilisation de ces méthodes pour le futur.

Mais j'ai fait aussi une dernière et quatrième méthode :

```
/**
* @function : getReferenceUsersOfSelectedContact
* @param: id saisieid
* @description : allows the getReferenceContacts and getReferenceUsers method to be pooled together, to avoid repetition
* @return : list of getReferenceUsers(getReferenceContacts(saisieid))
*/
@AuraEnabled(cacheable=true)
public static list<User> getReferenceUsersOfSelectedContact(id saisieid){

    return getReferenceUsers(getReferenceContacts(saisieid));
}
```

Pour faire simple, c'est simplement une méthode qui réunit les deux premières, pour faciliter le programme mais aussi pour éviter de brider ces trois méthodes. En effet, le fait d'écrire cette méthode pourrait permettre dans le future à ce que d'autres personnes puissent réutiliser mes trois méthodes, sans qu'il y est réellement de contrainte avec ces dernières, ni de limitation, …

Remarque : ne pas oublier de mettre la ligne suivante à chaque méthode : @AuraEnabled (cacheable = true). Cela permet de rendre visible une méthode à Visual Studio Code et donc pouvoir mieux l'utiliser.

Une fois toute les méthodes écrite et tester, il faut maintenant les importer dans Visual Studio Code.
Pour importer une méthode, il suffit de se rendre dans le JavaScript de la page concernée, en l'occurrence : "lwcCrmUserManagement".

Il suffit d'écrire ces deux lignes, pour importer mes deux méthodes :

```
/*Méthode Importer*/
import getReferenceUsersOfSelectedContact from '@salesforce/apex/GEN_UserManagement_Ctrl.getReferenceUsersOfSelectedContact';
import getReferenceUsersAccounts from '@salesforce/apex/GEN_UserManagement_Ctrl.getReferenceUsersAccounts';
```

Import nom_de_la_méthode from 'destination'

Une fois importé nous pouvons utiliser nos méthodes Apex dans Visual Studio Code.

Avant de faire quoique ce soit, j'ai d'abord analyser comment fonctionne le Javascript moderne. Dans notre cas, il est utilisé comme langage asynchrone. C'est-à-dire que les fonctions ne vont pas attendre les fonctions précédentes pour s'exécuter. Cela permet notamment d'assurer une meilleure performance dans les échanges clients serveurs. Mais cela nécessite donc plusieurs rappel de fonctions.

Puis dans le html, j'ai essayé de voir à quoi correspondaient les éléments. Par exemple, j'ai rajouté à plusieurs endroits des balises , pour voir où était situé les éléments, et c'est comme ça après que je suis parti ensuite dans le JavaScript, pour voir avec quoi les div html était en adéquation avec JavaScript.

J’ai donc ensuite cherché comment était fait les listes dans le code HTML, car rappelez- vous, je dois faire une liste qui se nomme “Referent User” (voir annexe: b.3. Maquettage :).

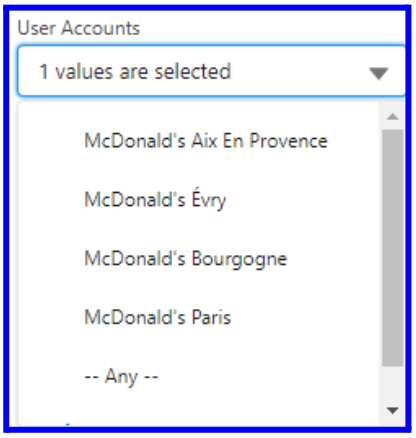
J’ai remarqué que dans ce langage, il y avait plusieurs types de listes, qui était définie avec des composants. Les composants sont des fichiers externes au fichier que j’utilise, composant un code html, Javascript, … Ce code est par exemple, un programme pour une liste, ou pour un design. C’est à peu près le même principe qu’une fonction, tu écris un code quelconque pour coder par exemple une liste dans un fichier appart, et tu réutilise le code à chaque fois qu’on veut une liste.

J’ai donc remarqué qu’il y avait trois liste utiliser donc trois composants :

- Une liste, ou l’on peut sélectionner qu'un seul élément ou non.
- Une autre liste ou l’on peut sélectionner un ou plusieurs éléments.
- Puis une liste, où l'on peut sélectionner aucun élément.

Dans mon cas, la deuxième liste correspond à ce que je dois faire comme tâche.

J’ai donc recherché une liste déjà coder et implémenter sur la page par défaut, pour m'inspirer de cette dernière.
J’en n’ai donc trouver une :



Quand j’inspecte le code source de la page avec la touche “F12” du clavier, je peux voir à quel composant cette liste appartient. J’ai utilisé ce procédé de nombreuses fois, pour faciliter ma recherche sur les éléments de la page internet. Surtout qu’il y à plus de 500 lignes pour le html, et plus de 800 lignes pour le JavaScript. Avant de trouver cet élément j’ai fait plein de tests partout pour savoir vraiment lequel des éléments pourrait être pertinent pour ma solution. Je n’ai pas trouvé cela du premier coup, j’ai dû demander de l’aide, car la manière dont le code est fait est différente avec ce que j’ai pu faire habituellement.

Je constate donc que cette liste appartient à ce code html :

```
<template if:true={showUserAccountList}>
    <template if:false={deleteUserAccountBefore}>
        <template if:true={refreshUserAccount}>

            <c-gen-multi-select-combo-box
                onselecteduseracc={handleSelectedUserAccount}
                values={optionsUserAccount}
                picklistlabel="User Accounts"
                selected-value={listUserAccountSelected}>
            </c-gen-multi-select-combo-box>
```

Quand j'ai vu le code pour la première fois, je me suis dit que cela ne ressemblait pas vraiment à du code HTML classique. Exactement pareille pour le JavaScript, et la la différence est encore plus flagrante. J'ai quand même essayé de faire quelque chose, mais c'était seulement pour essayer de trouver des éléments déjà existants, ou les modifier.
L'équipe m'a donc conseillé avant de continuer ma solution, des formations en lignes gratuites, qui pourrait m'aider à mieux comprendre comment est fait le code.

J'ai donc utilisé la plateforme en ligne de formation Salesforce, qui se nomme Trailhead.
Lien du site : https://trailhead.salesforce.com/fr

Il faut juste créer un compte pour pouvoir accéder au contenu total de la formation.

J'ai donc effectué environ cinq formations, dont trois qui sont primordiales pour comprendre le code.

Lien des trois formations :

1- Lightning Web Components Basics | Salesforce Trailhead

2- JavaScript Skills for Salesforce Developers | Salesforce Trailhead

3- Modern JavaScript Development | Salesforce Trailhead

Une fois toutes les formations effectuées, j'ai fait plusieurs vidéoconférences avec plusieurs membres de l'équipe pour m'aider. On n'a donc décomposé la consigne à l'oral en plusieurs étapes, pour mieux coder par la suite. C'est d'ailleur un conseil, qu'on m'a donné, de d'abord réfléchir à ce qu'on veut faire avant de commencer directement à coder.

Il y à donc toutes ces étapes pour assurer le codage finale :

Etapes

1) Sélectionner un contact
Il doit déja y avoir une fonction qui gère l'événement de sélection/changement de contact
Il faut : que dans cette fonction, faire un appel à la fonction APEX qui permet de récupérer la liste des "user Referent"
userReferents = *****

A) trouver la fonction qui gère le changement/selection de contact
B) console.log ("j'ai trouvé la fonction qui gère le changemenbt de contaact")
C) faire un appel à callGetReferentContact() (déplacer le console.log dans callGetReferentContact())
D) Faire un appel APEX à la fonction getRefere,tContact (quand j'ai un resultat, j'affiche console.log("jai réussi à faire mon appel à getreferentcontact")
E) au lieu du console.log actuel, faire un console.log ("data = ", data)
F) Affecter les valeurs récupéré à la liste userReferent

2) Afficher la liste des Users référents dans le field "User Referent"
Afficher la liste des referents dans mon nouveau champ "user Referent"

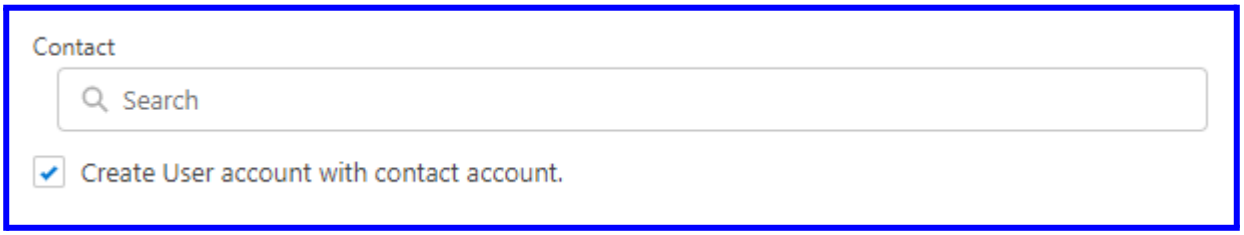
3) Selectionner un User Referent dans liste (de type combox box)
Gérer la selection/chan,gement d'un "User Referent"
--> faire un appel à la fonction qui va récupérer la liste des userAccounts
userAccounts =[****]

4) D'afficher la liste des "useraccount" dans le composant déja existant
Afficher la liste des "userAccount"

1) A-B) Pour la première étape, pour trouver la fonction, j'ai d'abord chercher avec le code source de la page l'élément. Et j'ai aussi fait des console.log dans la fonction pour voir si cela marchait. J'ai donc trouver la fonction, avec le code html :

```
<template if:true={showContactAccount}>
    <div class=" slds-grid slds-gutters_small slds-m-top-xxx_small slds-p-top_small slds-p-bottom_x-small " >
        <div class="slds-p-right_small slds-p-left_small slds-size_8-of-12">
            <label class="slds-form-element__label" >Contact</label>
            <template if:false={editUser}>
              <!--template if:false={createNewContact}-->
                <c-lwc-crm-user-management-lookup iconname='standard:contact'
                        entity={entitySelected}
                        oncountcontact={handleToCreateContact}
                        onselectedrec={handleChangeContact}>      <!--Fonction rechercher-->
                </c-lwc-crm-user-management-lookup>
              <!--/template-->
            </template>
```

Ce qui correspond à cette barre de recherche :



Donc malgré les formations, j'ai eu quand même des difficultés de compréhension. J'ai donc continuer de demander de l'aide de temps en temps à l'équipe. Revenons maintenant au code html. Dans la première ligne nous pouvons voir une balise template, c'est tout simplement une balise qui s'occupe de l'affichage, et du stockage d'un code html, pour que ce code s'affiche sous certaines conditions. Dans ce cas, cette barre de recherche est visible, quand on choisit :

Situé à gauche de l'écran.

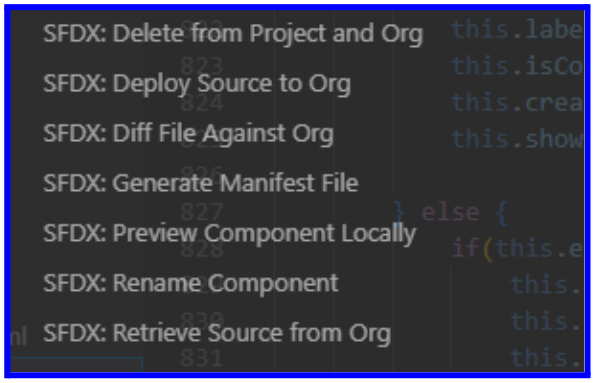
Il y à ensuite deux div avec une classe à l'intérieur, ou l'on définit directement la taille de la barre de recherche en longueur et en largeur. Ce procédé va être utilisé dans tout le code, c'est pour cela que pour toute la page, il y à qu'une seule ligne dans le fichier CSS. Le style de la page est gérer par des composants, ou directement dans des classes html. Dans le template il y à une présence du true et du false. SI l'événement est true cela s'affiche, sinon cela afficher autre chose.

Ensuite nous pouvons voir la présence du composant c-lwc-crm-user-management-lookup, il s'agit donc d'un composant coder par une personne, et qui permet d'avoir la liste de tous les contacts.

Les mots clefs : “entity, oncountcontact, onselectedrec ” correspond au paramètre que va prendre le composant. Le premier correspond à la liste des contacts, le deuxième à la création du contact, et le troisième à l’événement quand on clique sur le composant.
Celui qui m’intéresse et le troisième mot-clefs, car on voudrait qu’il y est une action quand on clique sur le composant.
Nous recherchons donc le troisième mot clef, dans le javascript, et nous tombons directement sur un appel d’une fonction.

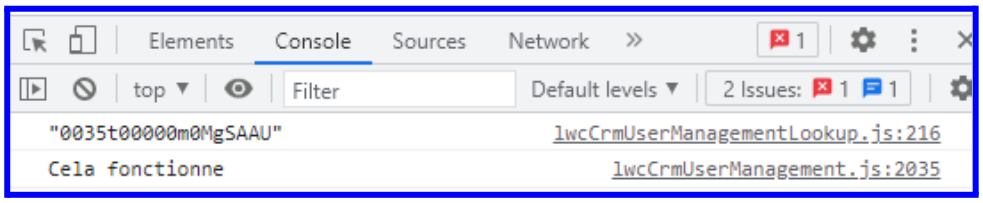
```
/**
 * @function handleChangeContact
 * @param Event event
 * @description Assign info contact to the form user
 */
handleChangeContact(event) {



    if(Boolean(event.detail.recordId)) {
        if(this.editUser == false) {

            console.log("Cela fonctionne");
            this.contactAccount = event.detail.recordId;
            this.firstNameUser = event.detail.contactObject.FirstName;
            this.lastNameUser =  event.detail.contactObject.LastName;
```

Avant de tester, il faut bien vérifier de sauvegarder, mais aussi de déployer le code à chaque changement :

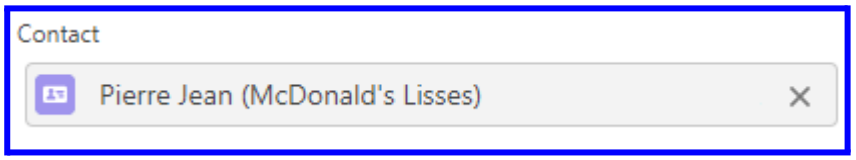


Sur le deuxième élément.

Nous pouvons bien voir que cela fonctionne dans la console, de la page web quand on sélectionne un contact :



Nous avons donc, trouver l'élément dont on a besoin.



Dans ce cas on n'a sélectionné le contact Jean Pierre, c'est pour cela que dans la console, il y à l'id de Jean Pierre qui apparaît.

1) c-d) Il faut maintenant faire appel à la fonction callGetReferenceContact(), dans la fonction ou on l'a mis le console.log.

La fonction callGetReferenceContact() est une fonction d'appel qui n'existe pas encore, c'est donc moi qui vais la créer. Elle permet entre autres d'appeler mes méthodes Apex qui sont importées., au départ la fonction est vide.

Pour l'instant la fonction que j'ai fait ressemble à ça :

```
callGetReferenceContacts(){

    //getReferenceContacts({saisieid : ""}).then((data) => {


        getReferenceContacts({saisieid: '0035t00000m0MgSAAU'})
        .then(result => {
            console.log("Cela fonctionne")
        })

        .catch(error => {
            console.log("Cela ne fonctionne pas")
        });
// });
 }
```

On met donc en paramètre la saisieid, comme pour la méthode getReferenceContact. On fait donc appel à cette fonction Apex depuis Visual Studio Code.

Ensuite le .then, permet d'enregistrer le résultat obtenu dans la variable : “result”. Si cela marche cela affiche “Cela fonctionne”, sinon cela affiche “Cela ne fonctionne pas”.

Dans ce cas, la méthode callGetReferenceContact(), permet de faire un appel des méthodes qui suivent. Elle permet donc d’appeler une fonction, suivant l'événement qui la déclenche. Comme nous pouvons le voir, la saisieid, est au début remplie manuellement, mais cela va changer dans le futur.

J’ai rajouter ensuite une petite spécificité dans le message d’erreur :

```
.catch(error => {
    console.log("cela ne fonctionne pas");
    console.log('error :', JSON.stringify(error));
});
```

Cela permet en résumé, d’afficher un message d'erreur plus complet, en convertissant notamment un objet en JSON. C’est à peu près le même procédé qu’un toString() en Java.

Et on fait donc appel à la fonction au même endroit que le console.log :

```
handleChangeContact(event) {


    if(Boolean(event.detail.recordId)) {
        if(this.editUser == false) {

            console.log("Cela fonctionne");
            this.contactAccount = event.detail.recordId;
            this.firstNameUser = event.detail.contactObject.FirstName;
            this.lastNameUser =  event.detail.contactObject.LastName;
            this.emailUser = event.detail.contactObject.Email;
            this.userNameUser = event.detail.contactObject.Email;
            this.selectedAccountIdFromContact = {label:event.detail.contactObject.Account.Name, value:event.detail.contactObject.Account.Id};
            this.listUserAccountSelected = [this.selectedAccountIdFromContact];
            this.callGetReferenceContact();
        }
```

Le but étant qu' à chaque fois, qu'on cliqueras sur la barre de recherche, ma méthode va être exécutée ainsi que l'affichage, …

1) e-f) Pour afficher la liste de tous les utilisateurs, il suffit simplement de faire un console.log de result, ou console.table pour que le résultat apparaissent sous forme de tableau :

```
callGetReferenceUsersOfSelectedContact(){

    this.listReferenceUser=[];

        getReferenceUsersOfSelectedContact({saisieid: this.contactAccount})
        .then(result => {
            console.log("Cela fonctionne ---");
            console.log ("result = ", JSON.stringify(result));

            console.table(result);


        })

        .catch(error => {
            console.log("cela ne fonctionne pas");
            console.log('error :', JSON.stringify(error));
        });
}
```

Le this.contactAccount qui est entré en paramètre, correspond à une variable qui est déjà existante dans le code. Cela correspond simplement à la liste des ID de tous les contacts.

Ce qui donne en résultat :

Cela fonctionne --- lwcCrmUserManagement.js:1366

result = lwcCrmUserManagement.js:1367

[{"Id":"0055t000001D4bKAAS","Name":"Olivier Martin"},
{"Id":"0055t000001D4iIAAS","Name":"Jean Pierre"},
{"Id":"0055t000001D4hKAAS","Name":"Marie Laure"},
{"Id":"0055t000001D4gqAAC","Name":"Pierre Smith"},
{"Id":"0055t000001D4glAAC","Name":"Marie-Fred Think"},
{"Id":"0055t000001D4ypAAC","Name":"Test Condition"}]

lwcCrmUserManagement.js:1368

(index)	Id	Name
0	'0055t000001D4bKAAS'	'Olivier Martin'
1	'0055t000001D4iIAAS'	'Jean Pierre'
2	'0055t000001D4hKAAS'	'Marie Laure'
3	'0055t000001D4gqAAC'	'Pierre Smith'
4	'0055t000001D4glAAC'	'Marie-Fred Think'
5	'0055t000001D4ypAAC'	'Test Condition'

▶ Array(6)

Dans le cas ou j'ai sélectionner Jean Pierre comme contact :
Nous pouvons bien voir, que cela affiche tous les utilisateurs associé au comptes de Jean Pierre.

Dans le cas ou j'ai sélectionner Gabriel Martinez comme contact :

Cela fonctionne --- lwcCrmUserManagement.js:1366

result = lwcCrmUserManagement.js:1367

[{"Id":"0055t000001DZtcAAG","Name":"Gabriel Martinez"}]

lwcCrmUserManagement.js:1368

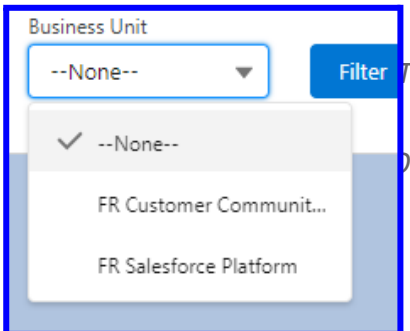
(index)	Id	Name
0	'0055t000001DZtcAAG'	'Gabriel Martinez'

▶ Array(1)

Il y à qu'un seul utilisateur qui lui est associé.

Bon maintenant qu'on sait que le résultat renvoie bien la bonne valeur, il faudrait renvoyer ce résultat dans une liste qui se nomme Referent User.

Il faut d'abord aller dans le code HTML, pour pouvoir voir s' il y à déjà un composant qui exploite une liste déroulant.



J’ai donc pris exemple sur cette liste, pour savoir quel composant elle utilisait, car avec cette liste nous pouvons sélectionner plusieurs éléments.

Voici donc ci-dessous, le code html de la liste par défaut :

```
<div class="slds-col slds-size_3-of-12 slds-medium-size_3-of-12 slds-large-size_3-of-12 slds-text-align_left  slds-align-bottom slds-form-element_stacked ">
   <lightning-combobox placeholder={labels.UCM_Choose_Business_Unit}
      name="Business Unit"
      label={labels.UCM_Business_Unit}

      value={bussinessUnitSelected}
      options={bussinessUnitOptions}
      onchange={handleChangeBU}
      >
   </lightning-combobox>
</div>
```

Puis voici ma liste réadapter :

```
<template if:true={contactAccount}>
   <template if:false={deleteUserAccountBefore}>
      <div class=" slds-col slds-size_4-of-12 slds-medium-size_4-of-12 slds-large-size_4-of-12 " >
         <lightning-combobox placeholder={labels.UCM_Referent_User}
            name="Referent User"
            label={labels.UCM_Referent_User}

            value={referentUserSelected}
            options={listReferenceUser}
            onchange={handleChangeReferentUser}
            >
         </lightning-combobox>
      </div>
   </template>
</template>
```

Cette fois-ci, il y à encore plus de mot-clef que précédemment :

Name : fait référence au nom de la liste.

Label : Cela correspond à la traduction du titre de la liste.

Value : La valeur qui à été sélectionnée dans la liste.

Option : la valeur de la liste.

Onchange : l'événement en cas de sélection dans la liste.

Donc voila il y à maintenant, le code pour la liste Referent User. Mais évidemment la liste est au départ vide. Du coup, on va revenir sur ma fonction JavaScript créer, et modifier certaines choses.

Il faut notamment modifier l'affichage, car une liste sous Salesforce, ne comprend pas l'affichage sous forme de tableau.
Il faut l'afficher de cette manière sous JavaScript : { label: 'None': value: '' }.

Pour faire simple, le label est utilisé comme une étiquette masquée visuellement si aucun `labels.label` n'est fourni. Alors que le value, correspond en quelque sorte à la valeur qui lui est associée, souvent c'est l'id.

Mon programme ressemble donc maintenant à ça :

```
listReferenceUser = [];



callGetReferenceUsersOfSelectedContact(){

    this.listReferenceUser=[];

        getReferenceUsersOfSelectedContact({saisieid: this.contactAccount})
        .then(result => {
            console.log("Cela fonctionne ---");
            console.log ("result = ", JSON.stringify(result));
            console.table(result);

            this.listReferentUsersId=result;

            this.listReferenceUser = result.map((element) => {

              return  {label:element.Name, value:element.Id};

            });

            /*  Deuxième possibilité */
           /* result.forEach((element, index) => {

                this.listReferenceUser.push({label:element.Name, value:element.Id});

            })
            //this.referentUsers = this.listReferenceUser;
            this.listReferenceUser = [...this.listReferenceUser];
            console.table(this.listReferenceUser);*/

        })

        .catch(error => {
            console.log("cela ne fonctionne pas");
            console.log('error :', JSON.stringify(error));
        });
}
```

Dans ce cas j'ai créé un nouvel attribut : "listReferenceUser[]" qui contient un tableau vide. En JavaScript pour faire appel à une variable dans une méthode, il suffit d'insérer le mot-clefs suivant : "this" avant le nom de l'attribut. Cela permet de faire référence à un objet. C'est exactement ce que j'ai fait avec : this.listReferenceUser=[] pour réinitialiser le tableau à chaque exécution de la méthode. Comme dit précédemment, le callGetReferenceUsersOfSelectedContact(), permet de faire appel aux méthodes qui suivent. D'ailleurs, vous avez peut-être remarqué que la méthode d'appel et la méthode qui suit n'ont pas du tout le même nom que les captures d'écrans d'avant. En effet, j'ai maintenant utilisé la quatrième méthode que j'ai expliquée dans cette partie : b.7. Lien entre Apex et Visual Studio Code :. Je l'utilise car cette méthode regroupe directement les deux premières en une. Ce n'est en aucun cas un choix qui est obligatoire, mais cela permet notamment d'assurer une certaine performance mais aussi, pouvoir réutiliser mes méthodes sans forcément

avoir la même finalité. Sinon, le principe est le même dans le code, nous prenons en paramètre la saisie id avec le this.contactAccount. On enregistre également tout le résultat dans “result”.

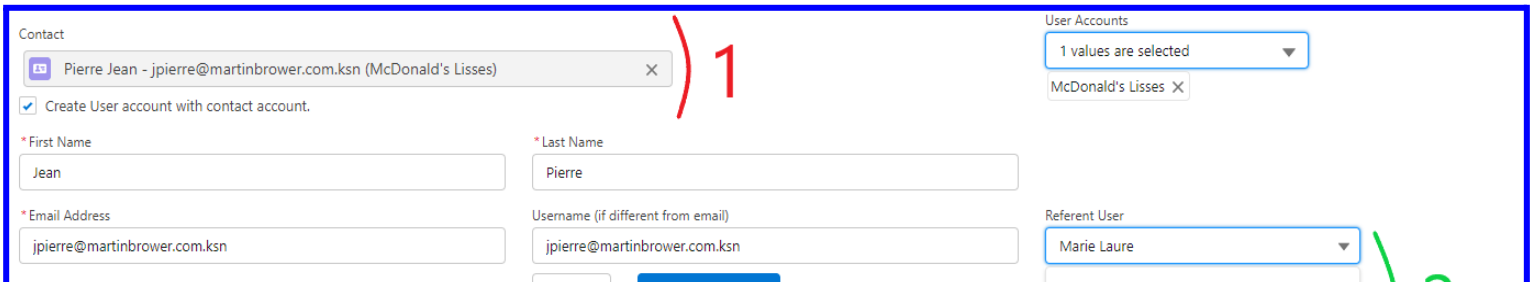
Et c’est là que les choses nouvelles apparaissent, comme je l'avais dit plus tôt, pour afficher des éléments dans une liste Salesforce, il faut utiliser des labels avec une valeur qui lui est associée.

Et c’est là que j’ai pensé d’utiliser la méthode forEach, qui permet d'exécuter du code sur chaque élément d'un tableau. Je parcours donc mon tableau result, en précisant les éléments et l’index qui lui est associé. Puis j'appelle mon autre tableau vide dans le forEach, en lui ajoutant les éléments de result sous forme de label : this.listReferenceUser.push({label:element.Name, value:element.Id});. J’utilise pour cela la méthode push, qui ajoute des éléments à la fin du tableau. Dans ce cas je veux les noms des éléments et son id. Le nouveau tableau contient maintenant les résultats sous forme de label.
Il y à aussi la présence de cette ligne : this.listReferenceUser = [...this.listReferenceUser];, cela permet tout simplement d’actualiser le tableau dans un nouveau tableau qui lui est identique. Car par défaut, HTML n’est pas suffisamment performant pour actualiser autant de données dans un tableau, c’est pour cela que je le force à le faire. Le “…”, est une syntaxe de décomposition, il permet notamment d' étendre le tableau dans des endroits où zéro ou plusieurs éléments sont attendus. Tous ces éléments c’est pour la méthode foreach.

Mais on m'a appris une nouvelle méthode qui est plus simple et plus courte : c’est la méthode map qui permet de créer un nouveau tableau, avec les résultats d’un tableau déjà existant, tout en modifiant ou non les valeurs du tableau déjà existant. Dans ce cas ci je reprends mon tableau vide : ”listReferenceUser”, et je lui dis comme quoi il est égale au tableau de résultat, mais avec un affichage sous forme de label. Toutes ces choses en seulement trois lignes, contrairement à ma méthode forEach :

```
this.listReferenceUser = result.map((element) => {

  return  {label:element.Name, value:element.Id};

});
```

Les deux méthodes sont totalement fonctionnelles, mais j’ai gardé la méthode qui utilisait la méthode map, car elle était beaucoup plus simple et plus courte.
Vérifions maintenant le résultat de tout ça :



1) On sélectionne pour la première étape le contact Jean Pierre, parmi la liste de tous les contacts.

2) Une fois que Jean Pierre à été sélectionné dans l'étape 1, cela à pour effet d'afficher toute la liste des utilisateurs associés au comptes de Jean Pierre, dans la liste Referent User. Dans ce cas j'ai sélectionné Marie Laure, en théorie à la fin cela devrait afficher les user account de Marie Laure, dans le champs User Account. Pour l'instant ce n'est pas le cas.

Maintenant, ce qui serait intéressant c'est que lorsqu'on clique sur un utilisateur de la liste Referent User. Cela met à jour automatiquement la liste des User Account, en affichant les User Account de l'utilisateur sélectionner.

Pour ce faire, il faut d'abord analyser quelle méthode dans le code permet d'exécuter une action quand on sélectionne un utilisateur.

On va donc revenir au code html, de la liste Referent User :

```
<!-- formulaire kevin -->
<template if:true={contactAccount}>
    <template if:false={deleteUserAccountBefore}>
        <div class=" slds-col slds-size_4-of-12 slds-medium-size_4-of-12 slds-large-size_4-of-12 " >
            <lightning-combobox placeholder={labels.UCM_Referent_User}
                name="Referent User"
                label={labels.UCM_Referent_User}

                value={referentUserSelected}
                options={listReferenceUser}
                onchange={handleChangeReferentUser}
                >
            </lightning-combobox>
        </div>
    </template>
</template>
```

Comme je l'avais expliqué, le onchange est une propriété qui s'active quand il y à une action de la part de l'utilisateur. Je vais donc utiliser cette propriété pour poursuivre le développement.

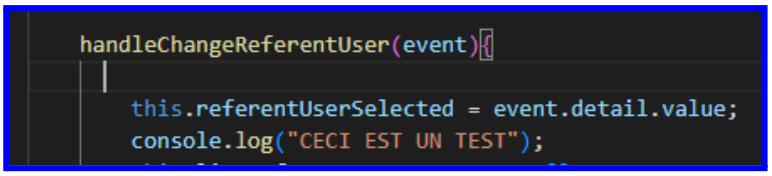
Pour vérifier si c'est cette méthode qui est la bonne, je vais mettre un console.log dans la fonction :

```
handleChangeReferentUser(){

    console.log("CECI EST UN TEST");

        getReferenceUsersAccounts({selecteduser: this.referentUserSelected})
        .then(data => {

            console.log("Cela fonctionne ---");
            console.log(data);
            console.log ("data = ", JSON.stringify(data));


        })

        .catch(error => {
            console.log("cela ne fonctionne pas");
            console.log('error :', JSON.stringify(error));
        });

}
```

Nous pouvons bien voir que cela fonctionne :

Maintenant que je sais que c'est cette fonction qui permet d'exercer une action lorsqu'on choisit un utilisateur de référence, je peux commencer à pouvoir coder. Le but étant de récupérer tous les User Account associés à l'utilisateur sélectionné.

Pour cela il faudrait déjà récupérer l'id de la personne sélectionnée. Pour ce faire, rien de plus simple, je me suis inspirée de plusieurs autres fonctions qui utilisent le onchange, et j'ai remarqué qu'il récupérait à chaque fois quelque chose qui s'appelait "event". On peut l'appeler comme on veut, mais on l'appel en général comme ça pour la clarté du code.
Mon code ressemble donc à ça :



J'ai juste mis en paramètre de la fonction, le mot-clé "event", puis j'ai mis le résultat de event dans la variable referentUserSelected que j'ai définit vide juste avant. Cette variable, contient donc l'id de la personne sélectionnée.

Donc voila mon code entier :

```
handleChangeReferentUser(event){
    this.refreshUserAccount = false;
    this.referentUserSelected = event.detail.value;
    console.log("CECI EST UN TEST");
    this.listReferenceUserAccount = [];

        getReferenceUsersAccounts({selecteduser: this.referentUserSelected})
        .then(data => {

            this.listUserAccountSelected = [];

            console.log("Cela fonctionne ---");
            console.log(data);
            console.log ("data = ", JSON.stringify(data));


            this.listReferenceUserAccount = data.map(elements => {

               return {label: elements.Account__r.Name, value : elements.Account__r.Id   };
              });

            console.table(this.listReferenceUserAccount);


            this.optionsUserAccount = [...this.listReferenceUserAccount];

            this.listUserAccountSelected = this.optionsUserAccount;

            console.table(this.listUserAccountSelected);

            this.refreshUserAccount =true;


        })

        .catch(error => {
            console.log("cela ne fonctionne pas");
            console.log('error :', JSON.stringify(error));
        });

}
```

Je définis en plus un tableau vide : “listReferenceUserAccount”, je le récupère ensuite avec le this. Je le met avant le getReferenceUsersAccounts, pour réinitialiser le tableau à chaque nouvel utilisateur sélectionné. Je fais donc appel à ma fonction “getReferenceUsersAccounts”, pour sélectionner tous les user account de l’utilisateur qui est sélectionné. J’enregistre le résultat dans data. Le this.listUserAccountSelected, correspond tout simplement au valeur sélectionnée dans la liste de User Accounts, au départ elle vaut rien. Puis je reprends mon tableau défini avant, et j’utilise le même principe qu’avant en utilisant la méthode map. J'affiche donc ce tableau pour vérifier dans la console, le résultat. Puis, je dis comme quoi le this.optionsUserAccount qui correspond à la valeur de la liste User Accounts est égale à mon tableau précédent.
Puis je reprécise, que les valeurs sélectionnées sont bien égales, à toutes les valeurs de la liste. Cela permet de tout sélectionner.
Je n’ai pas parlé jusqu'à présent du this.refreshUserAccount, alors on n’as mis ça en place, car il y à eu quelque petit problème par rapport à la sélection de toute les valeurs de la liste des User Accounts. Les valeurs en soi, était bonne et s'actualisaient correctement à chaque sélection d’un

utilisateur de référence. Mais l'affichage n'était pas actualisé, en raison du manque de performance du code HTML dans l'exécution de trop grande valeur.

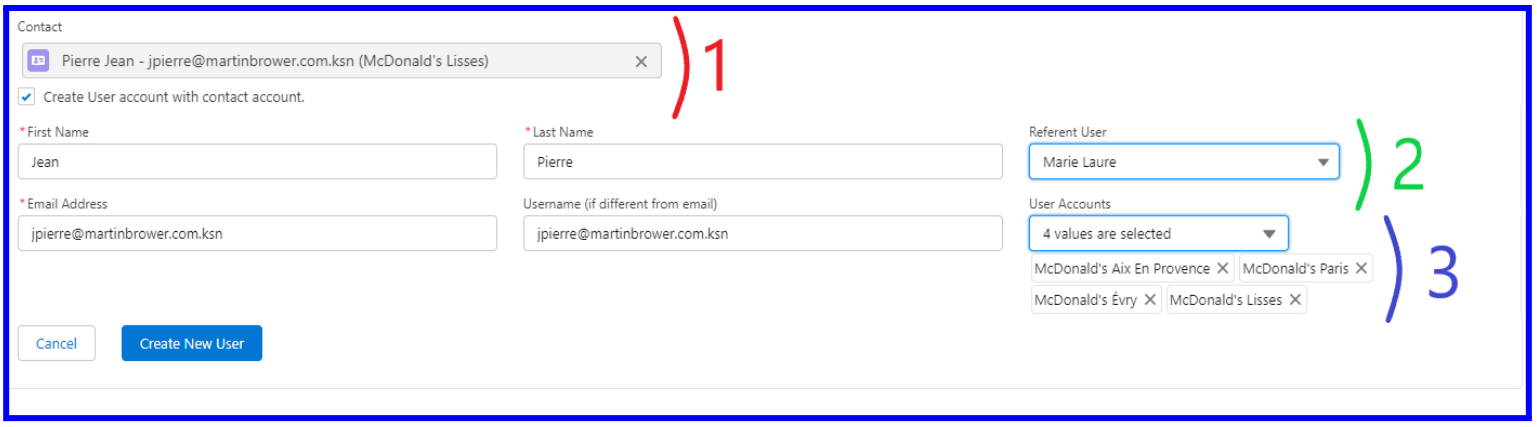
Dans ce cas, deux solutions pouvait être envisager :

- Pour la première solution soit utiliser la méthode reload, qui permet d'actualiser des ressources.
- Ou alors, modifier le code html, pour faire disparaitre et réapparaitre la liste pendant quelques secondes.

Dans les deux cas, les méthodes sont identiques, mais on n'est quand même parti sur la deuxième méthode par simplicité.

```
<div class="slds-col slds-size_6-of-12 slds-medium-size_6-of-12 slds-large-size_6-of-12 ">
    <template if:true={showUserAccountList}>
        <template if:false={deleteUserAccountBefore}>
            <template if:true={refreshUserAccount}>


                <c-gen-multi-select-combo-box
                    onselecteduseracc={handleSelectedUserAccount}
                    values={optionsUserAccount}
                    picklistlabel="User Accounts"
                    selected-value={listUserAccountSelected}>
                </c-gen-multi-select-combo-box>
        </template>
    </template>
</template>
</div>
</div>
```

Nous avons donc rajouté une template dans le code HTML de la liste User Accounts. Cette template, on l'a nommée : "refreshUserAccount", et elle va permettre de gérer ce problème d'actualisation.
C'est pour cela que dans le code JavaScript, on précise avant la méthode comme quoi cela est false, pour le faire disparaître, puis à la fin de la méthode on lui donne la valeur de true, pour qu'il puisse réapparaître. Il y à donc pendant quelques microsecondes, la liste qui disparaît. Mais cela n'est pas très dérangeant pour l'interface.
Vérifions maintenant le résultat de tout ça :



L'affichage à changer mais c'est exactement la même chose. Je l'ai changé car, je trouve que comme cela c'est beaucoup plus compréhensible pour l'utilisateur.

1) On sélectionne pour la première étape le contact Jean Pierre, parmi la liste de tous les contacts.

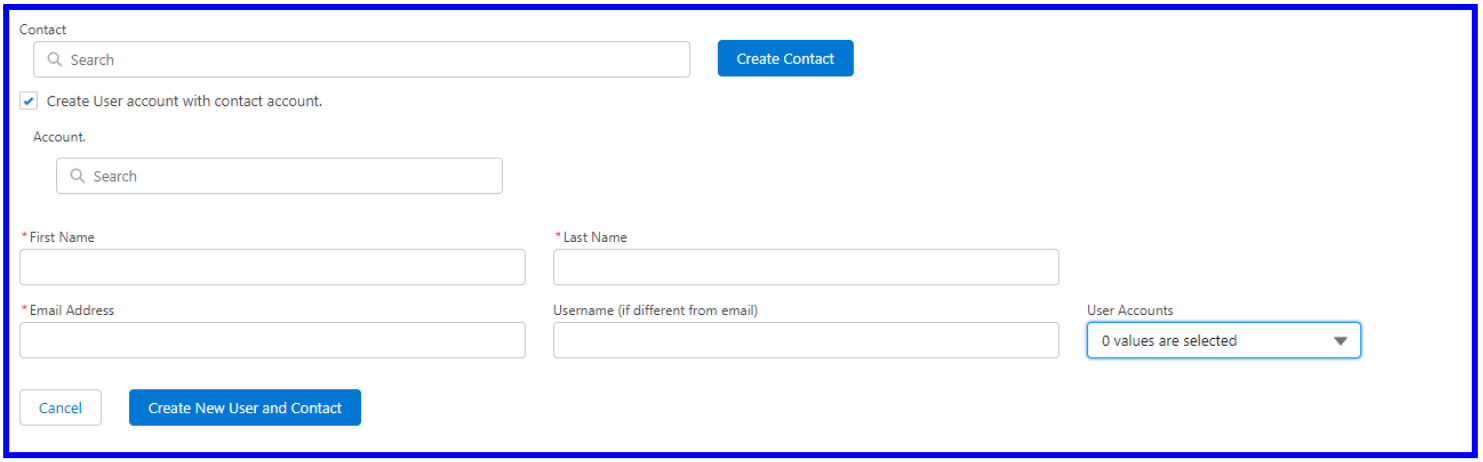
2) Une fois que Jean Pierre à été sélectionné dans l'étape 1, cela à pour effet d'afficher toute la liste des utilisateurs associés au comptes de Jean Pierre, dans la liste Referent User. Dans ce cas j'ai sélectionné Marie Laure.

3) J'ai bien maintenant tous les User Account de Marie Laure dans la liste User Accounts. Je pourrais donc créer un nouvel utilisateur qui à également ces user account. J'aurais très bien pu prendre un autre utilisateur dans la liste, du moment qu'il à des user account.

Je suis donc maintenant arrivé au résultat final demander.

b.8. Quelques petites précision :

Une fois ma tâche principale finie, on m'a donné une autre petite tâche qui était en rapport.



Pour résumer, ma tâche consistait à faire apparaître ma liste Referent User, mais cette fois ci quand on clique directement sur le bouton Create Contact. Cette fois-ci la liste ne récupère pas le contact sélectionné vu qu'il y en n'as pas, mais directement les comptes dans l'onglet Account. Il faut donc que je change ma méthode apex getRefrenceContacts.

Donc ma nouvelle méthode ressemble à ceci :

```
@AuraEnabled(cacheable=true)
public static list<contact> getReferenceContacts(id saisieid){

    list<contact> selectedcontact = [select accountid from contact where id=: saisieid];

    ID acctidselectedcontact;

    if(selectedcontact.size() >0){
        acctidselectedcontact= selectedcontact[0].accountid;
    }else{
        list<account> selectedaccount = [select id from account where id=: saisieid];
        if(selectedaccount.size() > 0){
            acctidselectedcontact = selectedaccount[0].Id;
            System.debug(acctidselectedcontact);
        }
    }

    System.debug('acctidselectedcontact '+acctidselectedcontact);
    list<contact> selectedcontacts = [select id, name from contact where accountid=:acctidselectedcontact];

    System.debug('selectedcontacts '+selectedcontacts);

    return selectedcontacts;

   // }



}
```

Je n'ai pas changé ce que la méthode prenait en paramètre, mais j'ai rajouté plusieurs conditions, pour vérifier si le résultat est nul, mais aussi pour prendre en compte que ce soit un contact en paramètre ou un compte. C'est pour cela que j'ai rajouté une requête SOQL. Ma méthode peut donc maintenant récupérer en paramètre un contact sélectionné ou un Account.

En ce qui concerne Visual Studio Code, j'ai fait exactement la même chose que pour le premier appel, j'ai fait ce code :

```
callgetReferenceContactWithAccount(){
    this.listReferenceUser=[];

        getReferenceUsersOfSelectedContact({saisieid: this.assignContactToAccount})
        .then(datas => {

            console.log("Cela fonctionne ---");
            console.log ("data = ", JSON.stringify(datas));
            console.log(datas);

            this.listReferenceUser = datas.map((element) => {

                return  {label:element.Name, value:element.Id};

            });

        })
        .catch(error => {
            console.log("cela ne fonctionne pas");
            console.log('error :', JSON.stringify(error));
            console.log(datas);
        });

}
```

Comme vous pouvez le voir, c'est la même chose car cela se base sur le même principe. La seule différence c'est que cette fois ci on prend en paramètre : saisieid: this.assignContactToAccount. Le this.assignContactToAccount correspond à l'id de l'account sélectionner dans la liste Account.

Mais j'ai donc maintenant un autre problème, car je récupère bien les résultats mais la liste ne s'affiche pas, et c'est tout à fait normal, car la liste Refrent User s'affiche seulement quand on sélectionne un contact, hors la on en sélectionne aucun. Il faut donc que je modifie le code HTML de la liste, j'avais pensé au début de mettre un ou dans le html. Mais on peut seulement mettre des outils de comparaison en JavaScript.

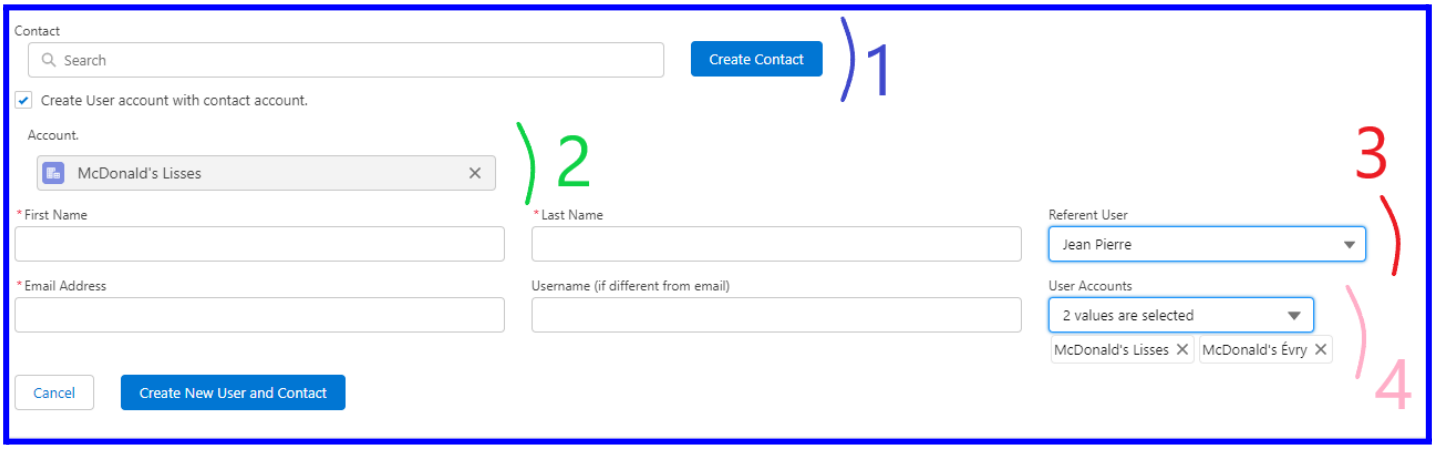
```
<!-- formulaire kevin -->
<template if:true={contactorsaccountselected}>
    <template if:false={deleteUserAccountBefore}>
        <div class=" slds-col slds-size_4-of-12 slds-medium-size_4-of-12 slds-large-size_4-of-12 " >
            <lightning-combobox placeholder={labels.UCM_Referent_User}
                name="Referent User"
                label={labels.UCM_Referent_User}

                value={referentUserSelected}
                options={listReferenceUser}
                onchange={handleChangeReferentUser}
                >
            </lightning-combobox>
        </div>
    </template>
</template>
```

J'ai donc créé une nouvelle variable qui se nomme : "contactorsaccountselected", cette variable prend en compte les deux événement qui permet d'afficher cette liste. J'ai donc appeler cette variable dans le fichier JavaScript et j'ai coder cela :

```
contactorsaccountselected(){

    return Boolean(this.contactAccount || this.assignContactToAccount);
}
```

C'est tout simplement un booléen entre les deux événement déclencheurs, soit c'est l'un ou soit l'autre. Cela dépend surtout du choix de l'utilisateur.

Cela donne maintenant cela en résultat :



1) On appuie sur le bouton "Create Contact"

2) On sélectionne un account parmi la liste. Dans ce cas j'ai sélectionné McDonald's Lisses.

3) C'est exactement le même principe que précédemment, cela affiche tous les utilisateurs de McDonald's Lisses.

4) C'est exactement pareille aussi. Cela affiche les user account de l'utilisateur référence qui est sélectionné.

On m’a aussi demandé de faire pareille, mais lorsque l’utilisateur appuie aussi sur le bouton Create Contact, mais avec un contact sélectionner en plus. En résumé, au niveau du code c’est presque pareil, et le résultat est le même. Le but étant de coder tous les choix que peut faire l'utilisateur.

J’ai dû aussi corriger un problème qu’il y avait avec leur interface. Le problème était que quand on voulait créer un utilisateur depuis un contact n’en n’ayant pas et bien cela créait bien l’utilisateur avec son contact associé, mais cela gardait aussi l’ancien contact. Du coup on se retrouvait avec deux contacts en double. Je suis donc aller dans la classe Apex : “UserManagerCreationUserQueueable” pour corriger le problème de codage. Le problème était juste le fait que la fonction pour créer le nouveau contact avec son utilisateur qui lui est associé était mal indentée. Cela veut dire que dans n'importe quel cas il crée un contact, donc si c'était true cela créait deux fois le contact.

J’ai donc maintenant fini mon projet, je ne l’ai pas forcément précisé, mais j’ai dû mettre évidement des commentaires pour chaque modification que j’ai pu apportée dans le code mais aussi enlever une bonne partie des console.log pour les tests, pour ainsi alléger le programme. Pour valider mon projet, j’ai fait une réunion avec toute l’équipe Salesforce, pour leur expliquer mon projet et le but qui était derrière. Puis j’ai dû faire une liste de toutes les choses que j’ai pu modifier ou créer, pour savoir sur quoi impactait mon programme et quelles sont les modifications que j’ai pu apporter. Mon projet à donc été validé et sera mis en place sur la plateforme publique de l’entreprise.

c.1. Difficulté rencontrée

La principale difficulté que j’ai rencontrée durant le déroulement de cette mission est la prise en main de LWC et Apex. En effet, ce sont tous deux de nouveaux langages qui diffèrent selon moi de beaucoup d'autres langages. J’ai d’ailleurs été surpris que JavaScript moderne soit totalement différent par rapport à JavaScript classique.
Malgré la plateforme de formation de Salesforce et l’aide de mon maître d’apprentissage ainsi que l’équipe Salesforce, il m’a tout de même fallu être extrêmement minutieux lors du codage dans LWC et Apex, ou encore dans le rendu final de l'interface et cela, afin que tout se passe correctement une fois déployé. Il fallait aussi essayer de réfléchir à la logique du code mais aussi de l'utilisateur pour que cela soit le plus clair possible. Il m’a donc fallu chercher des solutions et écrire avant sur feuille les étapes que je devais faire pour coder. Notamment pour gérer le contenu

d’une liste déroulante qui demandait une fonction précise, et gérer les différentes actions de cette liste. Mais aussi pour éviter au maximum des régressions de code, même si j’en n'ai quand même un peu eu. Cependant, à travers cette difficulté, j’ai pu être responsabilisé davantage par mon maître d’apprentissage afin de gagner en assurance dans la réalisation du projet. J’ai dû aussi par exemple exposer des problèmes à d’autres personnes qui n'étaient pas dans l’entreprise. J’ai pu prendre notamment contact avec celui qui à coder le lwc de cette entreprise, pour m’expliquer certaines choses et pour m’aider.

d.1. Conclusion du projet :

Le codage d’une page lwc a été ma dernière mission à laquelle j’ai pris part durant ce stage d’apprentissage. C’était une très grande mission comportant plusieurs petites missions. Le choix de cette mission ne s’est pas fait de manière anodine. En effet, mon tuteur souhaitait que j’accompagne un projet qui touche à beaucoup de langages de programmation, pour que je soit plus autonome et puisse avoir un projet technique qui est important pour l'entreprise.
De plus, travailler sur le LWC est une excellente manière de se former sur la plateforme Salesforce, pour pouvoir commencer le développement sous Salesforce. Cette mission concerne en grande majorité le développement sous Salesforce, ce qui pousse à étudier le fonctionnement et la logique de Salesforce et une partie de ses particularités. Ce projet m’a donc aidé à apprendre le fonctionnement en partie de la plateforme que je ne connaissais pas avant le début de mon apprentissage, et m’a aussi permis d’apprendre de nouveaux langages de programmation avec notamment une nouvelle vision dans la manière de coder.

Pour conclure cette mission, je peux dire que cela a représenté une bonne initiation aux méthodes de travail de l’entreprise. Elle m’a aussi permis de devenir petit à petit plus autonome, et cela, afin de pouvoir travailler sur d’autres projets personnels tout en gérant moi-même mon temps de travail.

V. Conclusion :

Ces cinq semaines en tant que stagiaire dans le secteur du développement informatique a été pour moi très enrichissante. Elle m'a permis d'avoir un second aperçu des attentes en entreprise et cela après une première année en BTS SIO.
J'ai pu découvrir un environnement professionnel riche me permettant de m'épanouir pleinement et de monter en compétence dans des secteurs jusqu'alors inconnus pour moi. J'ai pu notamment me former en partie à plusieurs outils et à du développement à travers la plateforme qu'est Salesforce.

J'ai également pu ajouter une nouvelle dimension dans mon parcours professionnel. En effet cela a été pour moi la première fois où le travail que j'ai effectué dans le cadre professionnel a eu un impact sur l'entreprise. L'ajout de cette dimension m'a notamment poussé à travailler mes compétences en terme de communication, aussi bien en Français au sein de mon équipe qu'en Anglais de temps en temps pour échanger avec des équipes extérieures.
J'ai été très encadré par mon maître d'apprentissage de manière à m'intégrer le plus naturellement possible à l'équipe aussi bien en termes de cohésion qu'en termes de projet. Mon maître d'apprentissage et l'équipe Salesforce m'a également permis d'apprendre à me mettre à la place du client, de savoir échanger avec lui et de comprendre au mieux son besoin.
Les différentes missions que j'ai effectuées m'ont permis d'acquérir des compétences nouvelles sur une plateforme Web en pleine expansion. Dans le cadre de celles-ci j'ai également pu tester mon autonomie notamment à travers l'apprentissage de nouveaux langages (Apex, SOQL...).

Ces cinq semaines en apprentissage m'a permis de me conforter dans mon choix de poursuite d'études. En effet, je désire poursuivre dans le développement tout en gardant aussi le côté design d'une page web.
L'entreprise Martin Brower a été pour moi une excellente découverte et je remercie encore toute l'équipe de son accueil m'ayant permis d'expérimenter une nouvelle étape dans mon parcours professionnel.

VI. Annexe :

Annexe : Les différentes sources m'ayant aidé durant mes missions :

Source	Les Raisons
https://trailhead.salesforce.com/fr	Se former personnellement sur l'outil Salesforce
https://www.deepl.com/translator	Pouvoir participer à certaines réunions en Anglais, ou pouvoir traduire certains termes liés à Salesforce, car l'outil est en anglais.
https://help.salesforce.com/home	Accéder à la documentation officielle de Salesforce.
L'équipe Salesforce	Me donner des informations et des aides
www.google.fr	Effectuer des recherches
https://www.lucidchart.com	Pouvoir créer facilement des diagrammes en ligne
https://genial.ly/fr/	Pouvoir créer facilement des diaporamas en ligne
Visual Studio Code - Code Editing. Redefined	Pouvoir coder en ayant directement un lien avec Salesforce
https://www.microsoft.com/fr-fr/microsoft-teams/group-chat-software	Pouvoir communiquer avec les autres membres de l'équipe et faire des réunions