Kévin
Salmon

BTS-SIO2

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

6 févr. 2023

Note de synthèse de 2ème année au sein de Martin Brower :

Développement WEB & Salesforce

Lycée Geoffroy Saint-Hilaire
4 Avenue Geoffroy Saint-Hilaire, 91150 Etampes
Année Scolaire 2022-2023

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 était nécessaire.

Sommaire :

I/ Introduction :

Dans le cadre de ma formation en BTS SIO (Option SLAM) Services Informatique aux Organisations, option “Solutions logicielles et applications métier”. J’ai effectué un stage du mardi 3 janvier 2023 jusqu’au vendredi 10 février 2023, au sein de Martin Brower à Lisses.

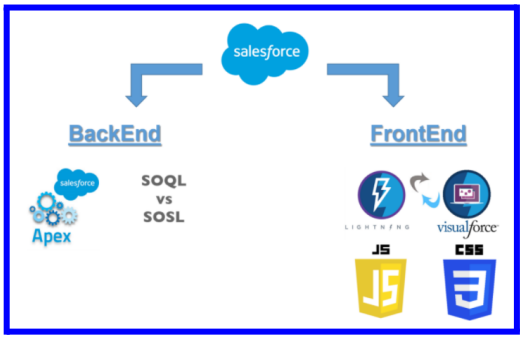
a) Cadre du Stage :

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.



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 agroalimentaire. 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.

Martin Brower est 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 salué par le prix de « Top employé » 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. Il 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 réaliser ces objectifs, il a fallu me former à la plateforme Salesforce que je n'avais jusqu'alors jamais utilisée.

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 que je sache exactement quoi faire et à quel moment. L'entreprise Martin Brower utilise un SmartSheet général pour la gestion de projets et la planification de tâches. Ce SmartSheet est divisé en plusieurs sections, chacune correspondant à une application spécifique. Sur ce SmartSheet, il y a les personnes assignées aux tâches, avec des codes couleurs pour définir si une tâche est en cours ou terminée, par exemple. Cette méthode de gestion de projet permet à Martin Brower de suivre efficacement l'avancement des tâches et de s'assurer que tout est fait dans les délais impartis.

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 termes d'outil CRM et est utilisée 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 qu'« 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 qu'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 : la partie configuration et la partie développement.

La partie configuration n'est pas pour autant simple à apprendre. 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 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 a 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 notre configuration. On peut donc créer des copies de plusieurs environnements, et donc avoir pour conséquence plusieurs sandboxes associé à des tâches respectives.

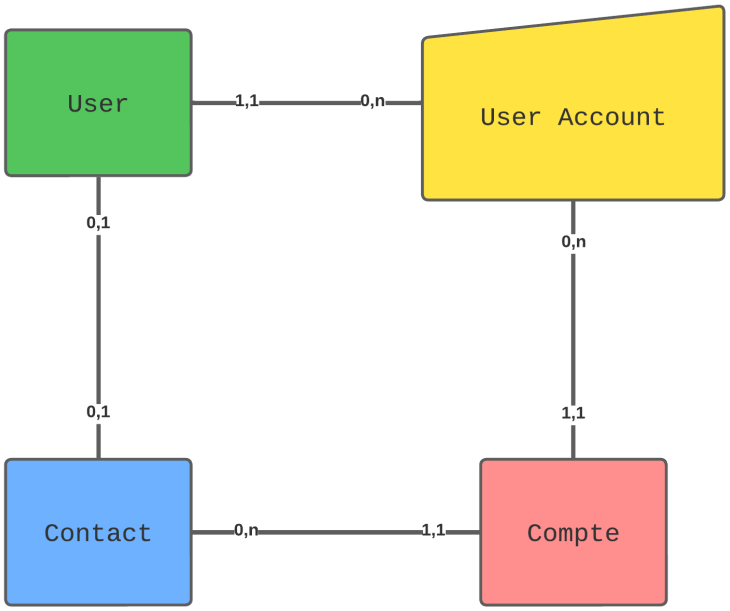
Au sein de cette sandbox, il m'est demandé de vérifier si en cas de changement de nom d'une configuration ou de sa licence, l'utilisateur associé à cette configuration dispose lui aussi de ces changements. Cette première mission représente majoritairement de la configuration de Salesforce et non de la programmation. C'est l'une des principales raisons pour laquelle j'ai commencé ce projet afin de me familiariser au mieux avec l'outil avant de commencer à développer. 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. User Configs

Tout d'abord, au sein de Salesforce il y a des "users configs", ce sont des configurations créées, pour donner par exemple certains droits à des utilisateurs associés à cette configuration.

Prenons l'exemple de l'utilisateur "Martin", Martin sera associé à FR_Évry par exemple.
-Un utilisateur est associé à un contact ou non.
-Un contact est associé à un compte ou non, elle représente la personne physique.
-Un Compte représente le restaurant associé à un contact.

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



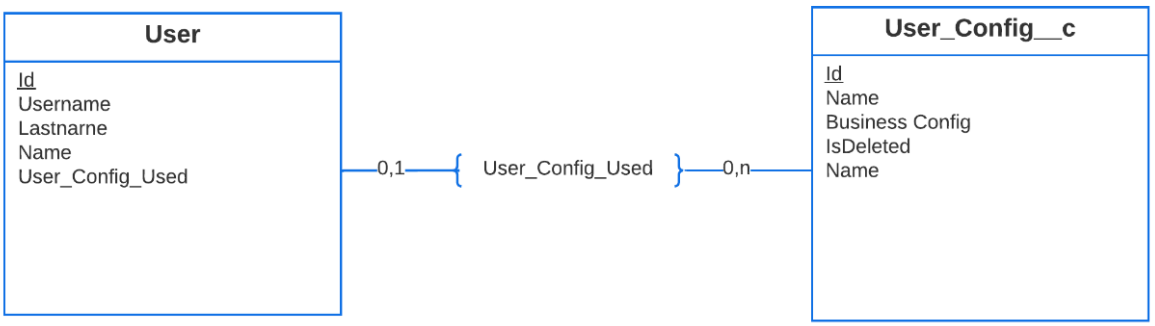
Légende :

0,1 : minimum 0 et maximum 1

1,1 : minimum 1 et maximum 1

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

Dans notre cas nous sommes intéressées plus particulièrement à cette relation :



C'est l'objet *User_Config_Used* qui permet de faire la relation entre ces deux objets. Un objet sous Salesforce est une représentation d'une entité spécifique utilisée pour stocker des données par exemple dans le système Salesforce. En résumé, c'est comme une table dans une base de données MariaDB, contenant différents attributs. Dans ce cas-ci, l'objet User peut stocker toutes les informations sur les utilisateurs de l'entreprise. Et l'objet *User_Config__c* peut stocker toutes les informations sur les différentes configurations.

b.2. Présentation de mes solutions :

Pour ce faire, j'ai envisagé plusieurs solutions, que j'ai répertoriées par la suite dans un document de conception technique (Un TDD en anglais). Le TDD permet d'expliquer un projet dans son ensemble et d'identifier les besoins, les contraintes et les parties prenantes qui y sont liés. Toute cette réflexion permet de clarifier le chemin à parcourir, sans rien oublier. Il s'agit donc d'une version théorique du projet.
Ma première solution était d'utiliser un trigger, pour déclencher un morceau de code en cas d'événement. Ce trigger sera décomposé en 3 grandes parties :

- Nous sélectionnons d'abord la liste des utilisateurs qui sont dans *l'user config* concernée.
- Nous sélectionnons la liste des utilisateurs pour récupérer tous les User Config Uses présents dans la base de données pour cette configuration.
- Enfin faire une comparaison entre l'utilisateur et *l'user config* pour voir s'il y a des changements entre le nom et le Business unit des 3 objets.

Pour plus de détails, voici le lien sur la définition d'un trigger :
Annexe 1 : Déclencheurs (Trigger)

Ma deuxième solution était d'utiliser des contrôleurs personnalisés pour vérifier l'élément problématique dans la liste des *Users Configs*. Cela permettrait de manipuler les données et de pouvoir les vérifier.
Pour plus de détails, voici le lien sur la définition d'un contrôleur personnalisé :
Annexe 2 : Contrôleur personnalisé

Ma troisième solution était d'utiliser un flow qui regarde les créations ou la mise à jour *d'user config* et qui quand ce dernier s'active, il mettra à jour le business unit config et le nom de la config de l'utilisateur.
Pour plus de détails, voici le lien sur la définition d'un flow :
Annexe : Le Flow

Revenons donc à mon document de conception technique. 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 exposées que je pourrais mettre en place pour répondre à cette problématique. Une seule des trois solutions fut retenue : la solution utilisant le Flow, car c'était le plus simple à mettre en place.
Pour plus de détails, voici le lien de mon document de conception technique :
Document de conception technique - User Config

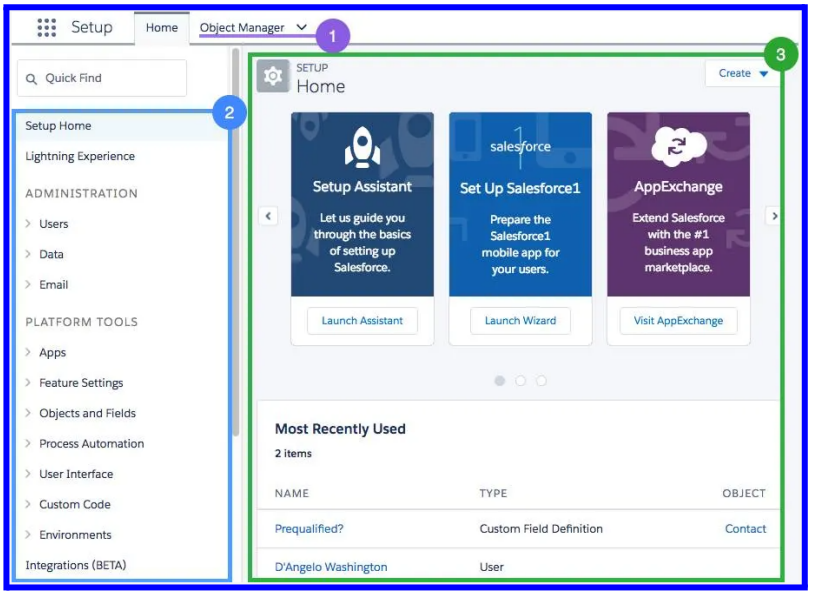
b.3. Créations du flow associé :

Comme expliqué dans l'annexe, le flow est la méthode la plus simple pour répondre à ce problème, malgré la non-présence de code. Le flow demande toutefois de la logique de développeur pour le concevoir. C'est justement la non-présence du code, qui facilite le déploiement de cette dernière, car cela évite les nombreuses adaptations à faire, et les problèmes que cela peut engendrer. C'est pour cela que le choix s'est porté sur le flow, car c'était la solution la plus viable.

Tout d'abord, pour créer un flow sous Salesforce, il faut d'abord identifier l'objet qu'il faut utiliser. D'après le schéma qui représente les relations entre User et *User_Config__c*, nous sommes plus intéressés par l'objet *User_Config__c,* en raison du fait que c'est dans cet objet que l'on peut modifier les champs d'une configuration.

Ces flows peuvent être définis 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.

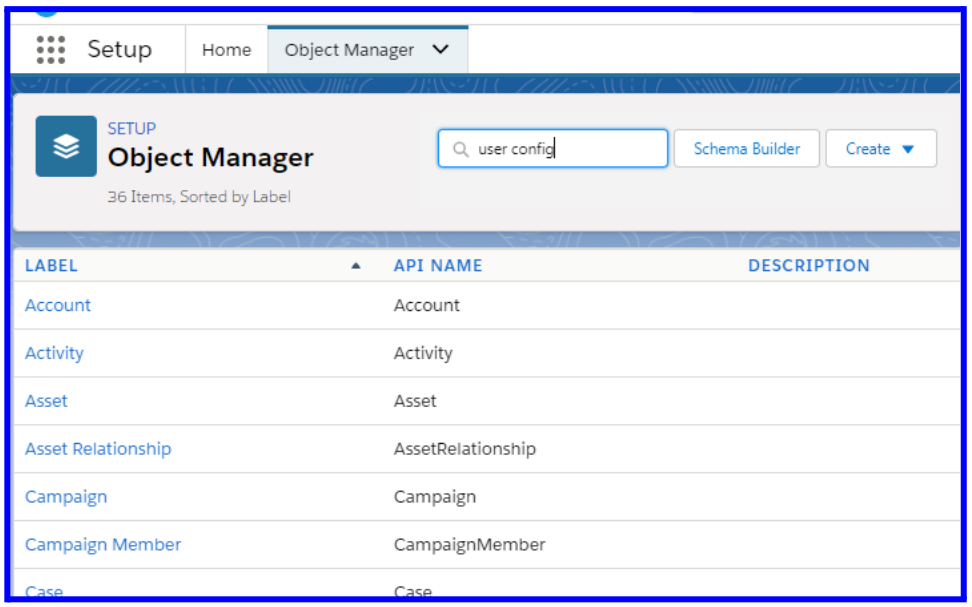


1. Gestionnaire d'objet : le gestionnaire d'objet permet de visualiser et de personnaliser les objets standards et personnalisés dans votre organisation. C'est d'ailleurs dans cet onglet qu'il faut se rendre pour les règles de validations.

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.

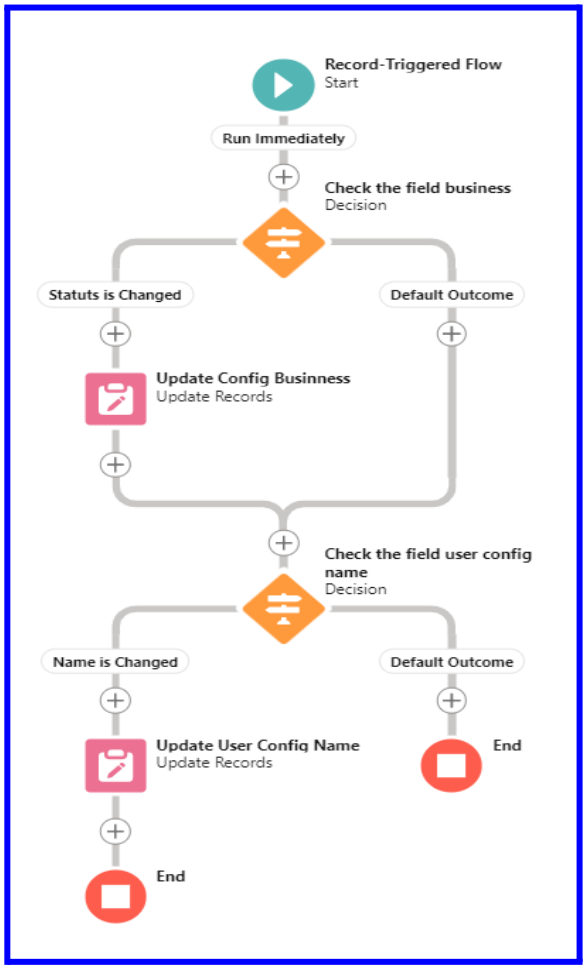
Une fois sur le gestionnaire d'objet, il faut rechercher l'élément sur lequel on veut instaurer des règles. Dans notre cas, on recherche donc *l'User Config* :



L'onglet "Object Manager" dans Salesforce est un outil qui permet aux utilisateurs de gérer les objets de données de leur organisation. Les objets de données sont des types de données tels que les comptes, les contacts, les opportunités, etc. qui sont utilisés pour stocker des informations dans Salesforce. L'onglet "Object Manager" vous permet de créer, personnaliser et gérer ces objets de données, ainsi que de définir les règles de validation et les relations entre eux. Il permet également de personnaliser les pages de détail, de liste et de formulaire pour chaque objet. En utilisant l'onglet "Object Manager", on peut également définir les autorisations d'accès et les rôles pour chaque objet. Une fois l'objet sélectionné, on arrive sur cette page où on peut voir en bas à droite les "Flows Triggers". Il suffit de sélectionner l'onglet, puis d'en créer un nouveau en lui donnant un nom et une description au préalable.

Il est important aussi de spécifier le “configure trigger”, pour définir des actions automatisées qui sont déclenchées lorsqu'un événement spécifié se produit, tel qu'une mise à jour d'un enregistrement ou la création d'un nouveau. Dans notre cas, nous sélectionnons la deuxième option, car on attend une update.

Au sein d’un flow, il y a plusieurs éléments disponibles comme par exemple : les décisions, les boucles,… Cela ressemble beaucoup à la programmation dans la logique.
Voici à quoi ressemble le flow :



Il y a tout d’abord, deux décisions (*if, else*) pour gérer cette vérification.

La première décision permet de vérifier si le champ *Business Unit Config* a été correctement changé en comparant les informations de l’objet *User_Config* et de l’objet *User_Config Used (l’Utilisateur)* . Si la mise à jour est considérée comme valide, le flow continue son exécution. Sinon, il ne se passe rien.
La deuxième décision permet de vérifier si la liste des utilisateurs possédant une configuration est bien mise à jour en cas de changement de nom de configuration. Sinon, il ne se passe rien.

En résumé, les deux décisions permettent de vérifier la bonne mise à jour des données et la complétude des utilisateurs configurés, et de signaler tout problème ou erreur détectés.
Pour plus de détails : ***Explication étape par étape :***

En plus de cette mission, j'ai également été amené à effectuer des modifications sur un flow existant pour répondre à un besoin spécifique d'une autre personne sur un projet déjà entamé. Cela m'a permis de découvrir comment les flows peuvent être utilisés de manière flexible pour répondre à des besoins variés et de développer mes compétences en matière de modification de flows existants. Cela a également été l'occasion de travailler en collaboration avec d'autres membres de l'équipe, ce qui a été une expérience enrichissante pour moi.

Pour plus de détails : Mission en parallèle de ma mission 1 :

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 du flow, car ce n'est pas un modèle habituellement étudié. Il m'a donc fallu chercher des solutions, notamment pour gérer le contenu d'un flow et en comprendre son fonctionnement et sa logique. 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. Mais le flow permet également d'étudier la logique de développeur. Ce projet m'a donc aidé à apprendre le fonctionnement en partie de la plateforme. Pour conclure cette mission, je peux dire que cela a représenté un bon retour parmi l'entreprise.

2. Gestion des limites Salesforce :

a.1. Contexte

Pour ma deuxième mission, mon maître de stage avait besoin de rendre dynamique la classe apex : *Salesforce_limits_Batch.apx*. Ma mission consistait donc à modifier cette classe Apex sous Salesforce. Cette classe était initialement utilisée pour gérer la limite quotidienne de Salesforce.

Mon objectif était de l'adapter pour permettre l'ajout automatique de limites supplémentaires dans le programme, car l'équipe aimerait améliorer la sélection d'un nom de limite Salesforce pour créer de nouvelles limites "personnalisées". C'est donc une demande d'optimisation de leur outil Salesforce.

Cette deuxième mission représente majoritairement des solutions dites fonctionnelles, par la suite après une mise au point avec l'équipe 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. Ensuite de les concrétiser pour que mes solutions soient donc maintenant techniques et non plus fonctionnelles.
Afin de réaliser cette tâche, j'ai dû comprendre le fonctionnement de la classe existante et identifier les points nécessaires à la modification. J'ai ensuite développé et testé les modifications nécessaires pour atteindre mon objectif.

b.1. Présentation de mes solutions :

J'ai donc présenté les différentes solutions à mon maître de stage et à l'équipe Salesforce.
Ma première solution était d'utiliser des langages de programmation, proposés par Salesforce. 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.

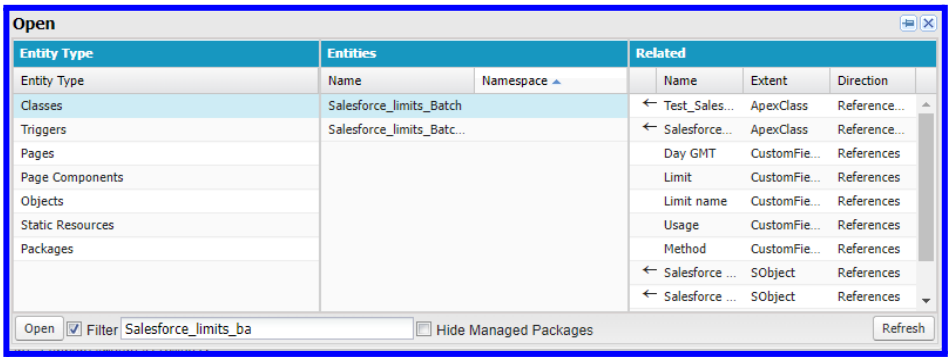
Ma deuxième solution était d'utiliser les outils proposés par Salesforce. Je voulais également utiliser les flows comme pour la mission 1. Mais ce ne fut pas très concluant, mais je leur ai quand même suggéré d'améliorer la problématique en y ajoutant des règles de validation. Pour notamment avoir une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

En conclusion, j'ai présenté mes solutions à mon maître de stage pour la modification de la classe Apex *Salesforce_limits_Batch.apx*. Ma première solution consistait à utiliser des langages de programmation tels que SOQL et Apex pour exécuter des requêtes précises depuis les bases de données et sélectionner les éléments nécessaires. Ma deuxième solution consistait à utiliser les outils proposés par Salesforce, tels que les flows, mais cela n'a pas été très concluant. J'ai néanmoins suggéré d'améliorer cette approche en y ajoutant des contrôleurs personnalisés pour une récupération plus précise. Mon maître de stage a préféré que j'utilise le langage Apex pour directement modifier la classe concernée.

b.2.Utilisation d'Apex et SOQL :

Nous avons donc décidé d'utiliser le langage Apex, qui est un langage par défaut sous Salesforce.
Pour plus de détails : Apex :
Tout d'abord, il faut identifier les ressources nécessaires pour commencer le projet. Pour ce faire, il suffit de se rendre dans la "developer console", qui est un environnement de développement intégré avec une collection d'outils pour créer, déboguer et tester des applications dans Salesforce.

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 étaient stockés notamment, les Accounts, Les User, ...
Je recherche donc la classe apex qui m'intéresse, dans ce cas-ci, je recherche la classe Salesforce_limits_Batch.apx



Il y a donc ce programme qui est présent : Annexe : Code
Pour résumer, ce programme est un batch Apex qui permet de mettre à jour la limite d'utilisation de l'envoi d'e-mails dans Salesforce en utilisant la classe OrgLimits. Il commence par récupérer la date actuelle et la convertit en format GMT. Il vérifie ensuite s'il existe déjà une limite pour cette date dans *Salesforce_limits__c*, et s'il n'y en a pas, il en crée une nouvelle.

Ensuite, il utilise la méthode getAll() de la classe OrgLimits pour récupérer toutes les limites de l'organisation, et parcourt cette liste pour trouver la limite "SingleEmail" et récupérer sa limite et son utilisation actuelle. Il met à jour ensuite les informations de cette limite dans *Salesforce_limits__c*. Enfin, il appelle une méthode *sendEmailIfError* pour envoyer un e-mail si une erreur se produit. Ce programme s'exécute chaque jour à 0 h 00, pour réactualiser tout le système.
Remarque : SingleEmail est une limite Salesforce déjà existante sous Salesforce et permet d'envoyer un e-mail d'avertissement suite à une valeur atteinte par exemple.

J'ai donc commencé par faire les requêtes SOQL, pour mieux récupérer les informations souhaitées.
Comme dit précédemment, on m'a demandé de rendre dynamique cette classe Apex, pour ce faire, on m'a demandé de tester une nouvelle limite qui n'était pas présente sous Salesforce, pour voir si cette limite serait bien prise en compte par le nouveau programme. Nous souhaitons donc ajouter une limite, qui permettra de récupérer le nombre de connexions maximum pour cette communauté : *Customer Community Plus Login*, sachant qu'il existe déjà un élément qui permet de le faire, mais nous voulons justement ajouter cet élément dans le programme de gestion des limites.
Le but étant de limiter le nombre de connexion par utilisateur, dans ce cas-ci : 9 234 par mois, pour leur envoyer une notification par e-mail au bout de 80 % de connexion atteint.
Avec le SOQL, je pourrais donc savoir quel champ me serait intéressant d'utiliser en requête, pour obtenir le résultat souhaité. Avant de modifier quoi que ce soit sur le programme, il est important d'identifier quel Objet permet de récupérer le nombre de connexion et le nombre limite par communauté. Au départ, je souhaitais récupérer les limites dans l'objet Salesforce_limits pour comprendre comment gérer les limites dans Salesforce. Cependant, en examinant l'objet, j'ai remarqué qu'il était vide par défaut et ne gérait que les enregistrements par jour en les réinitialisant à minuit.
J'ai ensuite voulu créer une limite par défaut similaire à SingleEmail, qui est déjà présent dans Salesforce et dans le programme précédent, mais je me suis rendu compte que ce n'était pas envisageable. J'ai également utilisé les objets AuthConfig et AuthSession, pour récupérer les informations de connexion des utilisateurs, mais ils n'ont pas fourni de champs intéressants pour mes besoins.
J'ai donc décidé de créer un programme Apex pour récupérer le loginurl de l'objet LoginHistory. Ce programme a fonctionné, mais le résultat obtenu n'était pas exactement ce que je cherchais :

```
Map<String, Integer> loginCounts = new Map<String, Integer>();

for (AggregateResult ar : [SELECT LoginUrl, COUNT(Id) FROM LoginHistory where LoginUrl
LIKE '%Login%' GROUP BY LoginUrl  ]) {
    loginCounts.put((String) ar.get('LoginUrl'), (Integer) ar.get('expr0'));
}

System.debug('Nb de connection : '+loginCounts+' || ');
```

En effet, ce programme m'affiche le nombre de connexions par utilisateur, mais en fonction de l'environnement de l'utilisateur et non en fonction de sa communauté.
En explorant les différentes options de configuration de Salesforce, j'ai découvert dans l'onglet Setup de Salesforce classique, la section "Company Information", contenant les champs Allowance et AmountUsed qui contiennent respectivement le nombre limites de connexions et le nombre de connexions par utilisateur dans une communauté.
J'ai alors essayé d'accéder à ces champs via SOQL pour les enregistrer dans un tableau Apex ou autre, mais je n'ai pas réussi à trouver la façon de les récupérer. J'ai signalé ce problème à mon maître de stage, mais il n'a pas trouvé de solution non plus. Nous avons donc créé un ticket pour Salesforce pour obtenir de l'aide.
La personne de Salesforce nous a finalement donné une requête SOQL de l'objet *TenantUsageEntitlement* qui sélectionne certains champs :

```
SELECT Id, CurrentAmountAllowed, AmountUsed, MasterLabel FROM
TenantUsageEntitlement
```

En examinant les résultats, j'ai découvert les champs *Allowance__c* et *AmountUsed*, qui étaient exactement ceux que je cherchais. Cette requête me renvoie donc le nombre de connexions limité et le nombre de connexions par utilisateur, par communauté :
Une fois, toutes ces recherches établis, j'ai commencé à modifier le programme *Salesforce_limits_Batch.apx* en y ajoutant des commentaires de ce que j'allais faire :

```
global Database.Querylocator start(Database.BatchableContext context){
    // old : Integer listLimits = [SELECT Count() FROM Salesforce_limits__c WHERE Day__c=:todayDateGMT];
    //TODO  List<String> listLimitname = new List<String>{'SingleEmail','TOBEDEFINE'};
    //TODO : List<Salesforcel_limits__c> listLimits = [SELECT Count() FROM Salesforce_limits__c WHERE Day__c=:todayDateGMT];
    //
    //TODO for(Salesforcel_limits__c lim : listLimits) --> créer map<String, Salesfroce_limits__c> (Name de la limite, record salesforce limits)
    //
    //List<Salesforce_limits__c> listoreturn = new list<Salesforce_limits__c>();
    //for each limit name
    //    if existe un salesforce limits dans la map
    //         ajouter le dans la lisete to return
    //   sinon
    //         créer un salesforce_limits__c
    //         et jouter dans listetoreturn
    //end for
    //return listetoreturn


    /* old :
     * if(listLimits<1){
        Salesforce_limits__c newLimit = new Salesforce_limits__c();
        newLimit.Limit_name__c = 'SingleEmail';
        newLimit.Day__c = todayDateGMT;
        insert newLimit;
    }
    return Database.getQueryLocator([SELECT Id,Limit_name__c,Limit__c,Usage__c,Day__c FROM Salesforce_limits__c WHERE Day__c=:todayDateGMT Limit 1]);   */
}

global void execute(Database.BatchableContext context, List<Salesforce_limits__c> scope){
    List<Salesforce_limits__c> listToUpsert = new List<Salesforce_limits__c>();
```

Depuis le début de la mise en œuvre du programme, j'ai effectué de nombreuses modifications par rapport aux commentaires du programme présentés ci-dessus. Cela a été nécessaire en raison de différents problèmes rencontrés et d'erreurs de compatibilité qui ont été constatées au fil du stage.

Malgré le fait que ces modifications aient été importantes pour améliorer la qualité et la stabilité du programme, je ne peux pas les présenter en détail ici, car cela prendrait trop de temps pour les expliquer de manière approfondie. Cependant, j'ai créé un nouveau champ dans l'objet *Salesforce_limits__c* qui se nomme *Method__c* pour enregistrer les clés de la map : *mapLimitName*. Voici donc le programme finalisé que j'ai réalisé : Annexe : Code

Le programme final permet donc de vérifier les limites de l'organisation, pour la journée en cours. Pour faire court, j'ai rajouté une map qui contient tous les noms des limites custom, permettant d'ajouter directement dans la map des nouvelles limites salesforce. Dans ce cas, nous ajoutons *Customer Community Plus Logins* avec son id respective pour définir où se situe la limite custom. Nous faisons une vérification que si parmi les noms de limites dans la map, il y a des limites qui ne sont pas présentes par défaut, crée un nouvel enregistrement de *Salesforce_limits__c*. Puis retourner les noms de limites par jour. Ensuite, appelé la méthode execute, qui va créer en résumé deux listes Apex, une liste qui contient les limites par défaut avec *OrgLimits* et une autre liste qui contient les limites personnalisées. Dans ce cas-ci *TenantUsageEntitlement* en référence à l'objet qui permet de récupérer le nombre de connexions… Je crée ensuite une boucle qui permet de vérifier si la limite est une *OrgLimits* ou une *TenantUsageEntitlement*. En fonction du résultat, le programme fera comme avant dans le cas d'une Orgs Limits, sinon il fera une boucle foreach de la liste qui contient les limites personnaliser, pour ajouter dans le champ *limitation* et *usage* de l'objet *Salesforce_limits__c* la valeur du champ *Allowance* et la valeur du champ *CurrentAmountAllowed*.

Remarque : le programme ajoute seulement la liste de limites dans l'objet *Salesforce_limits__c*, seulement s'ils ne sont pas déjà présents.

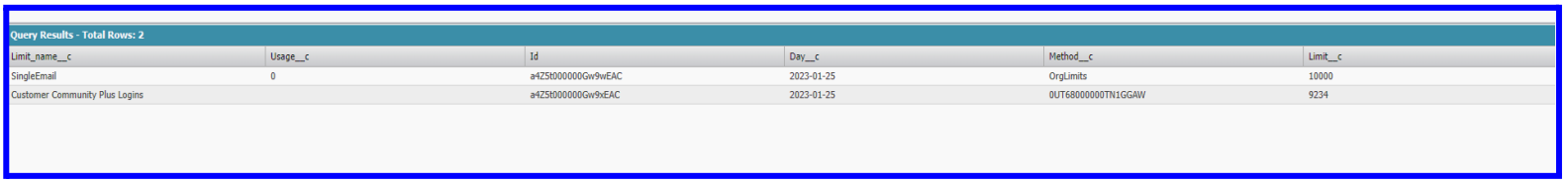
Pour exécuter le programme, il faut exécuter son *Schedule* associé vu que c'est un batch, cela permet entre autres l'exécution automatisée par rapport à une date et une heure précises. Pour ce faire, il faut rentrer dans la classe *Salesforce_limits_batchSchedule.apxc*, permettant de créer une nouvelle limite depuis la classe apex précédente :

```
global class Salesforce_limits_BatchSchedule implements Schedulable{
       global void execute(SchedulableContext SC) {
         Salesforce_limits_Batch batchLimit = new Salesforce_limits_Batch();
              database.executeBatch(batchLimit);
    }
}
```

Pour exécuter ce Schedule, il suffit d'ouvrir une page de débogage anonyme qui permet de tester et d'exécuter du code Apex sans effectuer de modification permanente. J'ai donc écrit ce petit programme, pour tester la classe *Salesforce_limits_batch.apxc* :

```
Salesforce_limits_Batch batchLimit = new Salesforce_limits_Batch();
database.executeBatch(batchLimit);
```

Nous remarquons bien que les limites sont créées :



Query Results - Total Rows: 2

Limit_name__c	Usage__c	Id	Day__c	Method__c	Limit__c
SingleEmail	0	a4Z5t000000Gw9wEAC	2023-01-25	OrgLimits	10000
Customer Community Plus Logins		a4Z5t000000Gw9xEAC	2023-01-25	0UT68000000TN1GGAW	9234

c.1. Difficulté rencontrée

La principale difficulté que j'ai rencontrée durant le déroulement de cette mission est la compréhension des langages Apex et SOQL. 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 de la modification du programme, et cela, afin que tout se passe correctement une fois déployé. Surtout, le moment où il faut chercher les différentes informations pour répondre au besoin, je trouve qu'à mon sens c'est la partie la plus complexe du travail. Le fait aussi, de modifier un programme déjà existant pour le rendre dynamique m'a paru difficile, car il faut se mettre dans la peau du développeur qui a codé ce programme et comprendre sa logique.

d.1. Conclusion du projet :

Le codage d'une classe Apex a été l'une de mes missions 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 sois plus autonome et puisse avoir un projet technique qui est important pour l'entreprise. De plus, ce langage est une bonne manière de se développer sur la plateforme 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 connaissais déjà, grâce à mon premier stage. Mais ça m'a permis d'encore plus m'améliorer en m'attaquant à des tâches que je trouve plus difficile.

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.

3. Gestion des licences

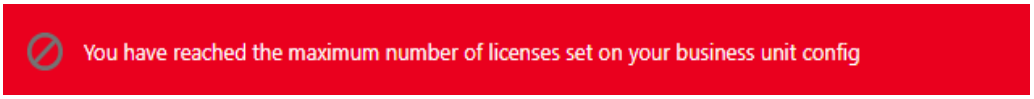
Salesforce :

a.1. Contexte

Cette fois-ci, pour ma troisième mission, cela représente plus une petite tâche à faire sur la partie *User Config Manager*. Mon maître de stage avait besoin de corriger un problème, lors d'un changement quelconque d'un utilisateur associé à une *User Config* n'ayant plus de licence disponible. En effet, un message d'erreur : "You have reached the maximum number of licenses set on your business unit config", s'affichant même en ayant le même type de licence. Ma mission consistait donc à identifier l'élément qui déclenche cette erreur, pour identifier la source du problème. C'est donc une demande d'optimisation de leur outil Salesforce. Pour réaliser cette mission, on me demande d'utiliser cette fois-ci tous les outils Salesforce qui sont mis à disposition.
Afin de réaliser cette tâche, j'ai dû comprendre le fonctionnement et identifier les points nécessaires à la modification. J'ai ensuite développé et testé les modifications nécessaires pour atteindre mon objectif.

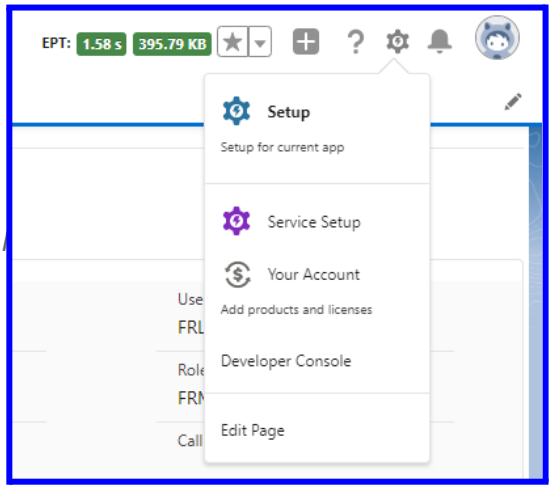
b.1. Utilisation de Visual Studio Code :

Il faut donc maintenant essayer de trouver une solution pour essayer de corriger le problème. J'essaye d'abord d'identifier, à quel moment le message d'erreur apparaît. Le message d'erreur apparaît, seulement quand on veut modifier un utilisateur associé à une *User Config* n'ayant plus de licences disponibles :



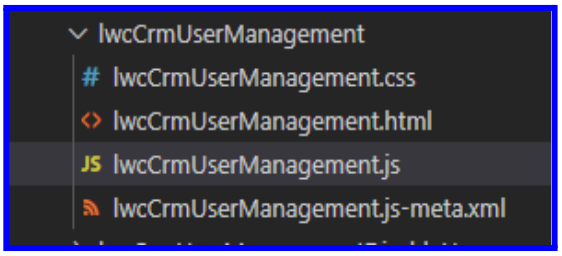
Je vais donc maintenant pouvoir trouver quel programme déclenche ce message d'erreur. Pour ce faire, il faut d'abord trouver la classe Apex qui est attribuée à la page où il y a le message d'erreur.

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 de sélectionner l'élément qui nous intéresse. Salesforce nous indique que le fichier associé à cette page est : *lwcCrmUserManagement*. Les lwc sont des composants Web, qu'on peut modifier seulement

sur Visual Studio Code. C'est pour cela que pour cette troisième mission, je vais utiliser le logiciel Visual Studio Code. **Pour plus de détails :** Visual Studio Code et lwc :



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.

Pour plus de détails : Langages utilisés sous une page Salesforce :

Une fois le fichier trouvé, je commence par regarder dans le fichier *HTML*, et j'ai aussi fait des console.log dans la fonction pour voir si cela affichait quelque chose, afin de voir quel élément contient le message d'erreur, et l'élément correspondant est :

```
<template if:true={blockCreateUser}>
   <div class="slds-box slds-theme_error">
      <div class="slds-inline_icon_text slds-grid slds-inline_icon_text--error">
         <lightning-icon icon-name="utility:error" size="small"></lightning-icon>
         <div class="slds-col slds-align-middle">
           <p>You have reached the maximum number of licenses set on your business unit
config</p>
         </div>
       </div>
     </div>
</template>
```

Nous pouvons voir que c'est bien le message d'erreur avec notamment la balise *<p>*. La partie du code "template if:true={blockCreateUser}" est une condition particulière qui vérifie la valeur de la variable "blockCreateUser". Si la valeur est vraie, le contenu du modèle sera affiché, et si la valeur est fausse, le contenu du modèle ne sera pas affiché. Ceci est utile pour contrôler ce qui est affiché à l'utilisateur en fonction de certaines conditions, par exemple si un utilisateur est autorisé à accéder à certaines informations, ou si une erreur s'est produite dans un processus particulier.

Je recherche donc la variable "blockCreateUser", dans le fichier javascript, et je tombe sur cette condition de la fonction *detailUserConfig* :

```
this.itemDetailsUserConfig = this.userConfigOptions.find(elem => {return elem.Id ==
this.userConfigId });


if(Boolean(this.itemDetailsUserConfig.Business_Unit_Config__r) &&
(this.itemDetailsUserConfig.Business_Unit_Config__r.Remaining_Licences__c <= 0))
        {
             this.blockCreateUser = true;
```

```
            this.createUserFlag = false;

        }else{
            this.blockCreateUser = false;
            this.createUserFlag = true;
```

La première ligne permet de récupérer toutes les informations de l'objet *Business_Unit_Config__c*, dans le tableau *userConfigOptions* en utilisant la fonction "find". Elle permet de trouver la configuration de l'utilisateur. J'ai compris que cette condition, afficher le message d'erreur avec le *this.blockCreateUser = true;*, quand la condition était à true. En regardant de plus près la condition, on remarque qu'elle se déclenche quand il n'y a plus de licences disponibles, et c'est donc cela le problème. En effet, elle se base juste sur le nombre de licences et non autre chose, c'est pour cela qu'il y a le message d'erreur qui s'affiche, même si l'on veut seulement mettre à jour des données dans une même licence, car il manque une condition dans le *if*. Avant de modifier la condition, il faut d'abord savoir quand se déclenche la fonction *detailUserConfig.* Pour cela je mets des *console.log*, pour tester quand est exécutée cette fonction :

```
console.log("Ceci est un test");
```

Ainsi voici le message, dans l'onglet *Console* du menu Inspecter du navigateur :

```
Ceci est un test                                    lwcCrmUserManagement.js:1691
```

Maintenant que je sais quand la fonction s'exécute, je regarde plus en détail cette partie :

```
this.itemDetailsUserConfig = this.userConfigOptions.find(elem => {return elem.Id ==
this.userConfigId });
```

En effet, maintenant il faut essayer d'identifier d'où ces informations sont récupérées. En faisant des recherches, je remarque que la fonction *detailUserConfig()* est appelée dans une autre fonction :

```
    handleRetriveUserConfig(event) {
        retrieveUserConfig({
            entity : this.entitySelected,
            location : this.locationSelected,
            businessUnit : this.bussinessUnitSelected
        }).then((result) => {
            this.wiredResults = result;
            this.isLoading = true;
            if(result && Array.isArray(result)) {
                this.userConfigOptions =  this.userConfigOptionsFiltred = result;
                this.showBlockDetails = true;
                this.isLoading = false;
                if(event == null) {
                    this.detailUserConfig();
                }
            } else if (result.error) {
                this.error = result.error;
                this.isLoading = false;
            }
        });

    }
```

Donc je recherche par la suite cette fonction *retriveUserConfig* et je tombe sur un import :

```
import retrieveUserConfig from '@salesforce/apex/GEN_UserManagement_Ctrl.retrieveUserConfig';
```

Cette importation fait référence à une classe apex : *GEN_UserManagement_Ctrl*. Si on recherche la fonction : *retriveUserConfig*, dans la classe, on tombe sur une variable qui semble être intéressante :

```
string query = 'select Id, Name, User_Accounts_with__c, Business_Unit_Config__c,
Keep_Old_UserAccount__c, Business_Unit_Config__r.Acces_Type__c ,
Business_Unit_Config__r.Remaining_Licences__c,
Business_Unit_Config__r.Hostname_Email_Accepted__c ';
```

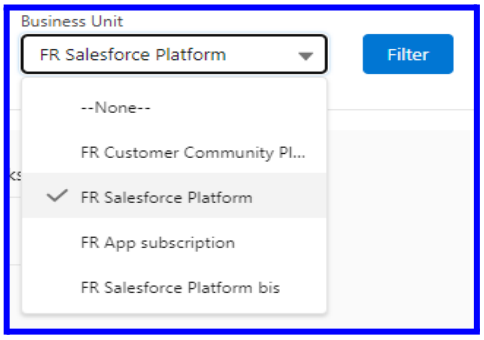
C'est grâce à cette requête SOQL que *this.itemDetailsUserConfig*, récupère toutes les informations de l'objet *Business_Unit_Config__c*. Maintenant que nous savons ceci, on peut commencer le développement :

```
this.old_license_type = this.itemDetailsUserConfig.Business_Unit_Config__r.Id;

if(Boolean(this.itemDetailsUserConfig.Business_Unit_Config__r) &&
(this.itemDetailsUserConfig.Business_Unit_Config__r.Remaining_Licences__c <= 0 &&
this.old_license_type != this.bussinessUnitSelected)){
    this.blockCreateUser = true;
    this.createUserFlag = false;

}else{
    this.blockCreateUser = false;
    this.createUserFlag = true;

}
```

J'ai donc fait cette condition premièrement en ajoutant un *"&& this.old_license_type != this.bussinessUnitSelected "*, sachant que pour le *this.old_license_type*, j'ai rajouté dans la variable *this.itemDetailsUserConfig*, le champ *Business_Unit_Config__r.Id*, qui n'était pas présent de base. J'ai donc dû l'ajouter dans la requête SOQL de la classe Apex :

```
string query = 'select Id, Name, User_Accounts_with__c, Business_Unit_Config__c,
Keep_Old_UserAccount__c, Business_Unit_Config__r.Acces_Type__c ,
Business_Unit_Config__r.Remaining_Licences__c,
Business_Unit_Config__r.Hostname_Email_Accepted__c, Business_Unit_Config__r.Id ';
```

J'ai choisi le champ *this.businessUnitSelected*. En effet, *this.businessUnitSelected* récupère l'élément de la liste des *Business Unit* de manière efficace. Cette partie : *this.old_license_type != this.bussinessUnitSelected*, vérifie si la license type précédente est différente de la nouvelle license sélectionnée pour le business unit.



Sachant que la liste précédente est représentée par cette liste déroulante. Les informations de cette liste déroulante sont récupérées par la variable : *this.bussinessUnitSelected*, dans la fonction : *handleChangeBU* qui s'active en cas de changement d'élément dans la liste déroulante. D'où le fait que je le compare avec la nouvelle variable : *this.old_license_type*.

Ma nouvelle condition fonctionne donc correctement, mais j'ai rapidement remarqué qu'elle causait des erreurs dans certaines situations. Cela a nécessité une modification de la condition pour éliminer ces erreurs et garantir un fonctionnement fluide :

```
this.old_license_type = this.itemDetailsUserConfig.Business_Unit_Config__r.Id;
if (!this.onlyChangeBusinessUnit) { //if there is to be no change of bussiness unit in the dropdown list

          if(Boolean(this.itemDetailsUserConfig.Business_Unit_Config__r) &&
(this.itemDetailsUserConfig.Business_Unit_Config__r.Remaining_Licences__c <= 0 && this.old_license_type !=
this.bussinessUnitSelected))
            {
                this.blockCreateUser = true;        //display of the error message
                this.createUserFlag = false;
            } else {
                this.blockCreateUser = false;       //the error message is not displayed
                this.createUserFlag = true;
            }
        }else{
            if(Boolean(this.itemDetailsUserConfig.Business_Unit_Config__r) &&
(this.itemDetailsUserConfig.Business_Unit_Config__r.Remaining_Licences__c <= 0)){
                    this.blockCreateUser = true;       //the error message is not displayed
                    this.createUserFlag = false;
            }else{
                this.blockCreateUser = false;       //the error message is not displayed
                this.createUserFlag = true;
            }
```

Cette nouvelle condition est comme la précédente, sauf que j'ai rajouté un if qui vérifie s'il n'y a pas de changement de la *Business Unit* dans la liste déroulante, avec ceci : *if (!this.onlyChangeBusinessUnit)*. C'est une nouvelle variable que j'ai créée et qui de base est mis sur "false", sauf que je l'a définie sur "true", quand la fonction : *handleChangeBU* est exécutée, car cette fonction comme dit précédemment, s'active qu'en cas de changement d'élément dans la liste déroulante. Ma nouvelle condition est donc fonctionnelle mais non optimisée. En effet, il y a trois conditions, en même temps, c'est donc potentiellement source d'erreur. Pour résoudre cela, j'ai écrit ma nouvelle condition comme ceci :

```
this.old_license_type = this.itemDetailsUserConfig.Business_Unit_Config__r.Id;
        // This code checks how many licenses are still available for the selected license.
const remainingLicenses =
Boolean(this.itemDetailsUserConfig.Business_Unit_Config__r)?this.itemDetailsUserConfig.Business_Unit_Config_
_r.Remaining_Licences__c : 0;
```

```
        // If there are no more licenses available and the previous license type is not the same as the one
selected or the user only changes business units
        if (remainingLicenses <= 0 && (this.old_license_type !== this.bussinessUnitSelected ||
this.onlyChangeBusinessUnit)) {
            // then we block the user creation
            this.blockCreateUser = true;
            this.createUserFlag = false;
        } else {
            // otherwise, we don't block the user creation
            this.blockCreateUser = false;
            this.createUserFlag = true;
        }
```

J'ai remplacé la condition en deux parties pour vérifier si *Business_Unit_Config__r* existe et pour obtenir *Remaining_Licences__c* en une seule ligne avec l'opérateur conditionnel *ternaire* pour une meilleure lisibilité. J'ai également utilisé l'opérateur de comparaison strict *!==* pour vérifier si *old_license_type* est différent de *bussinessUnitSelected* pour éviter les erreurs potentielles.
Le changement a bien été appliqué, et il n'y a plus de message d'erreur quand on veut simplement mettre à jour un utilisateur. Par contre, bien évidemment quand on veut créer un nouvel utilisateur sur une licence qui n'a plus de licence disponible, cela affiche le message d'erreur comme avant.

c.1. Difficulté rencontrée

La principale difficulté que j'ai rencontrée durant le déroulement de cette mission est la compréhension du Framework utilisant le JavaScript. En effet, c'est un framework propre à Salesforce, malgré le fait qu'il a des bases de JavaScript, c'est compliqué à comprendre au début. Cette tâche touchait un autre domaine de Salesforce au niveau développement, et m'a permis d'acquérir un peu plus de polyvalence au sein du stage. En effet, le lwc regroupe tous les langages étudiés jusqu'à présent, mais en ajoutant un aspect web, avec notamment JavaScript. Cette tâche était donc un moyen, d'utiliser toutes mes connaissances précédentes pour résoudre un problème précis. Cependant, cela m'a permis d'acquérir des compétences.

d.1. Conclusion de la tâche

En conclusion, pour ma troisième mission, j'ai travaillé sur la partie User Config Manager de Salesforce pour corriger un problème lié à l'affichage d'une erreur lors d'un changement d'utilisateur, avec une User Config sans licence disponible. J'ai utilisé différents outils Salesforce pour comprendre le fonctionnement et faire les modifications nécessaires pour optimiser leur outil. J'ai notamment utilisé la partie *lwc* de Salesforce. Grâce à mon travail, le problème a été résolu avec succès.

4. Gestion des licences Salesforce associées à un Profile :

a.1. Contexte

Tout d'abord, pour ma quatrième mission, cela représente également une petite tâche à faire sur la partie *User Config Manager*. Mon maître de stage aimerait résoudre ce problème : lorsqu'une licence type est associée à un profil, il est important de vérifier que les informations sont correctes par rapport à la licence type de son *Business Unit Config*, pour garantir un fonctionnement optimal.
Enfin, il est important de se baser sur le profil filtre pour effectuer cette vérification. Le profil filtre est un outil qui permet de définir les critères d'inclusion et d'exclusion pour les données associées au profil. Il permet de filtrer les données en fonction de critères spécifiques pour garantir que les informations associées à la licence type et au profil sont précises et à jour. En utilisant le profil filtre, les erreurs peuvent être détectées et corrigées rapidement pour garantir un fonctionnement optimal.

b.1. Présentation de mes solutions :

Pour ce faire, j'ai envisagé plusieurs solutions, que j'ai exposé par la suite aux membres de l'équipe.

- Ma première solution était d'utiliser une validation rules, qui permettrait d'envoyer un message d'erreur, si jamais le profil était inexistant, ou si le profil n'avait pas le même type de licence que l'*User Config*.
  Pour plus de détails, voici le lien sur la définition d'une validation rules :
  Validation Rules :

- Ma deuxième solution était d'utiliser un flow qui récupère d'abord toutes les données associées au profil, puis ensuite mettre en place une décision pour alerter l'utilisateur.
  Pour plus de détails, voici le lien sur la définition d'un flow :
  Annexe : Le Flow

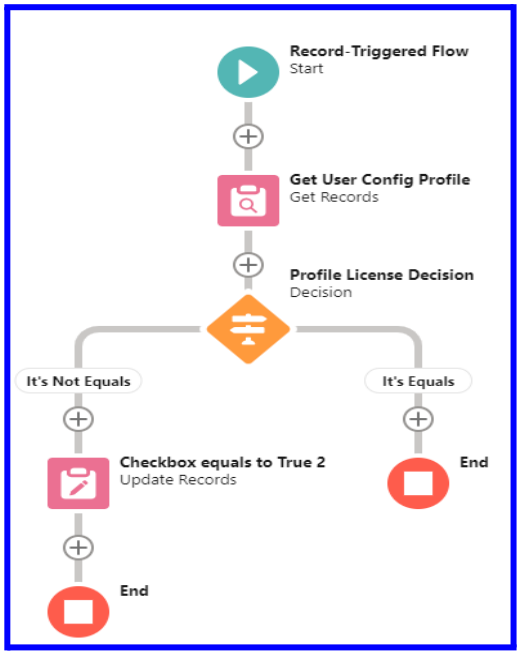
J'ai donc proposé les deux solutions, et les deux ont été acceptées. Cependant, je me suis rendu compte, suite à une erreur de compatibilité Salesforce, que la validation rules n'avait pas accès au même champ de l'objet *Profile* qu'un Flow, ce qui semble totalement bizarre. J'ai donc fait des recherches, et j'ai fini par faire une requête SOQL à moitié imbriquée :

```
select Name, Id, UserLicense.Name from Profile
```

J'ai donc décidé d'utiliser cet exemple de requête pour le flow, puis de le fusionner avec une validation rules, pour utiliser l'affichage d'erreur d'une validation rules et un flow pour l'aspect technique.

b.1. Utilisation d'un flow lié à une validation rules :

Par rapport au besoin exposé, il serait plus judicieux d'utiliser un flow, pour les mêmes raisons exposées précédemment : Mission 1. En effet, pour une petite tâche comme celle-ci, un flow est beaucoup plus simple à mettre en place.
Comme pour la mission 1, je commence donc par créer un flow dans l'objet *User_Config*. Cette fois-ci, le flow va se charger de mettre à jour des données, pendant une mise à niveau ou une création d'un *User Config*. Comme pour la première mission, au sein d'un flow, il y a plusieurs éléments disponibles comme par exemple : les décisions, les boucles, … Cela ressemble beaucoup à la programmation dans la logique. Voici à quoi ressemble mon flow :



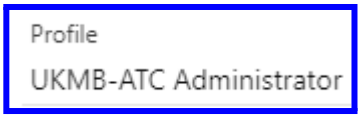
Tout d'abord, le flow récupère toutes les données intéressantes de l'objet *Profile*, et c'est justement cet élément-là qui fait qu'on peut récupérer des données spécifiques contrairement à la validation rules qui est restreint au champ de l'objet. Ensuite il parcourt une décision qui regarde si parmi les éléments du *Get Records*, le champ *UserLicense.Name* est égale à la licence actuelle. Si oui, on ne fait rien. Sinon on définit une "checkbox" sur true que j'ai créé au préalable. Cette checkbox, va permettre justement de faire le lien avec la validation rules.
Pour plus de détails : Captures d'écrans du flow :

J'ai donc créé cette validation rules, dans le même objet que le flow. J'ai simplement ajouter une condition :

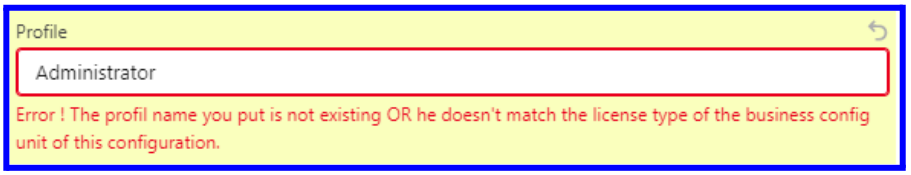
```
User_License_Checkbox__c = true
```

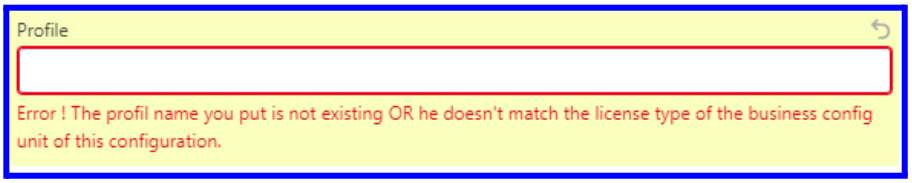
Sachant que le message d'erreur s'affiche si une variable est égale à true. J'ai donc défini ce message d'erreur : " Error ! The profil name you put is not existing OR he doesn't match the license type of the business config unit of this configuration ", que j'affiche en dessous du champ *Profile*.
Pour plus de détails : Captures d'écrans d'une validation rules :

De base, le profil est :



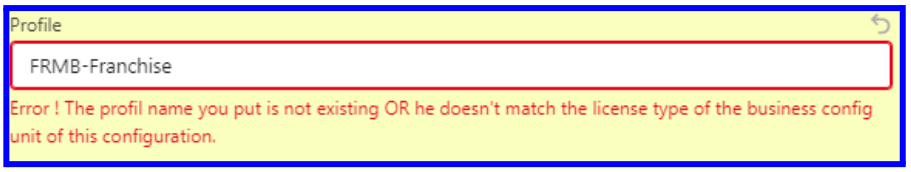
Si on souhaite modifier le profil, mais que le nom du profil n'existe pas alors :





Si on souhaite modifier le profil, mais que le nom du profil ne correspond pas à la licence type de l'*User Config* (Force.com - App Subscription) :
Nous sélectionnons le profil *FRMB-Franchise* qui a une licence *Customer Community Plus* :

Cela affiche également ce message d'erreur :



Par contre, même si on met un profil différent, mais qu'il a le même type de licence que l'*User Config*, il n'y a pas de message d'erreur.

c.1. Difficulté rencontrée

Durant cette troisième mission, j'ai été confronté à certains problèmes. Cependant, j'ai rapidement découvert que les champs de la validation rules et ceux d'un Flow présentaient des erreurs de compatibilité. J'ai donc dû faire face à des problèmes pour trouver un moyen de synchroniser les deux. Malgré les problèmes, j'ai persévéré et j'ai finalement trouvé une solution. Cependant, la mise en œuvre de la solution a été compliquée car c'était la première fois que je liais un Flow et une

validation rules à l'aide d'une “checkbox”. Cette expérience m'a permis de me développer plus. J'ai appris à résoudre les problèmes liés à la compatibilité des champs.

d.1. Conclusion de la tâche

Pour ma quatrième mission en stage, j'ai eu l'opportunité de travailler sur la partie User Config Manager une seconde fois. Mon maître de stage souhaitait résoudre un problème lié à la vérification de la correspondance entre une licence type et un profil. Il était important de s'assurer que les informations associées étaient correctes par rapport à la licence type de la Business Unit Config pour garantir un fonctionnement optimal. J'ai utilisé un flow et une validation rules pour effectuer cette vérification. Cette tâche m'a permis de découvrir l'importance de la vérification des informations associées à une licence type et à un profil. Je suis devenu plus conscient de la nécessité de garantir la précision des informations pour un fonctionnement optimal. Je suis fier de ma contribution à la résolution de ce problème et je suis convaincu que cette expérience sera utile pour mon expérience personnelle.

5. Tester plusieurs conditions à l’aide d’un Flow Schedule :

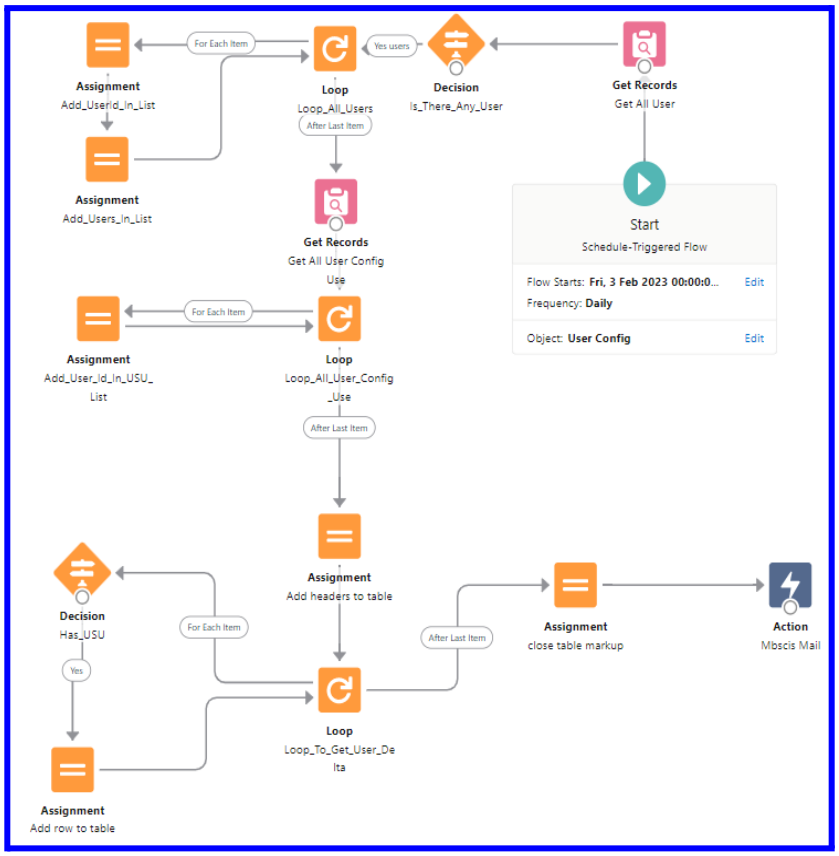
a.1. Contexte

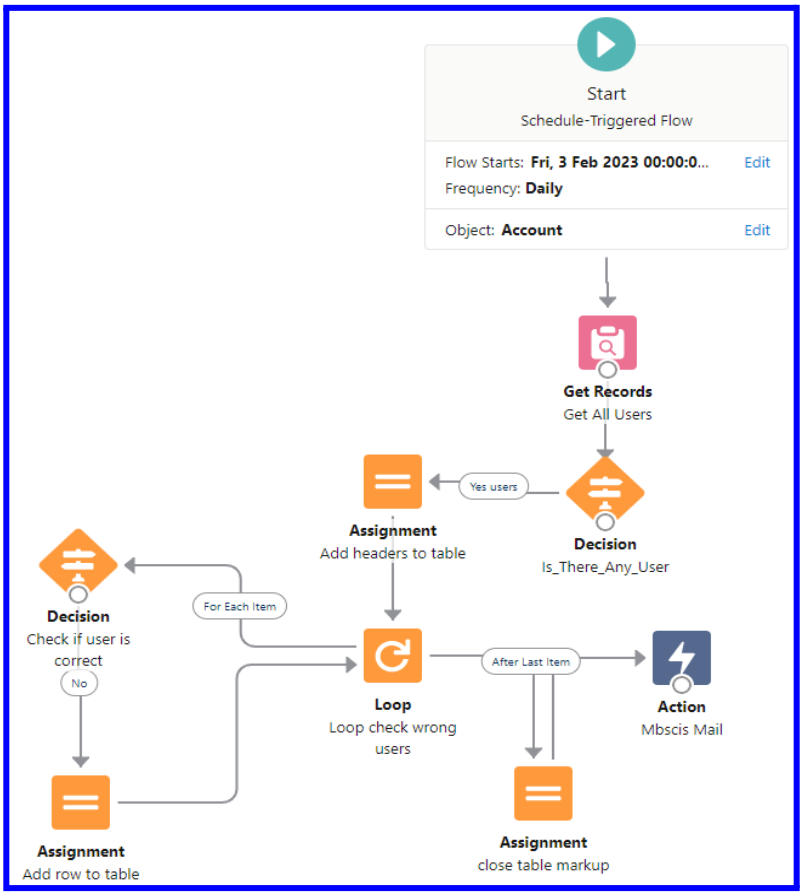
Tout d’abord, pour ma cinquième mission, cela représente aussi une petite tâche à faire sur la partie *Setup* de Salesforce. Cette quatrième mission consistait à créer un Flow schedule pour tester deux conditions. La première consistait à vérifier si l'utilisateur avait une *User_Config* et s'il n'y avait pas de *User_Config_Use* enregistrée dans cette dernière. La deuxième condition impliquait de vérifier si l'utilisateur avait un contact et si le champ Entity__c était égal au champ Company__c du compte associé au contact, lequel était lui-même associé à l'utilisateur. Dans les deux cas, si la condition est respectée, on envoie un mail, qui envoie la liste des utilisateurs associés, ainsi que le message associé. C’est donc une demande permettant de faire le point chaque jour sur les utilisateurs, grâce au mail.

b.1. Utilisation d’un flow Schedule :

Au cours de cette mission on m'a demandé d’utiliser un flow, pour les mêmes raisons exposées précédemment : Mission 1. En effet, nous voulons envoyer un mail chaque jour à minuit si une condition est remplie. Le flow schedule permet d'exécuter ce flow à une heure précise, chaque jour ou chaque semaine, …

Je commence donc par créer un flow dans l'onglet *flows*. Cette fois-ci, le flow va se charger de tester des conditions en récupérant les données, pendant une date et une heure précise. Dans notre cas, nous voulons que le flow s'exécute tous les jours à minuit, pour faire le point de chaque journée. J'ai décidé suite à des changements de séparer mon flow en deux flows distinct correspondant à une condition chacune. Voici à quoi ressemble mes flows :





Au départ ce flow était censé se baser sur l'objet User, mais il y avait un problème. En effet, dès que la condition était remplie le flow envoyait un mail pour chaque utilisateur, il y avait donc des centaines de mail qui était envoyé, ce qui n'est pas préférable. Ce problème était là, car le flow s'exécute un par un et non par paquet. Je me suis donc basé sur leurs *User_Config* associés, vu qu'il y en as beaucoup moins. Dans ce flow, il y à trois boucles à parcourir, permettant de diviser le travaille en trois parties. Le tout permettant de répondre à la première condition de la problématique précédente, en envoyant un e-mail.

Pour plus de détails : ***Captures d'écrans du flow :***

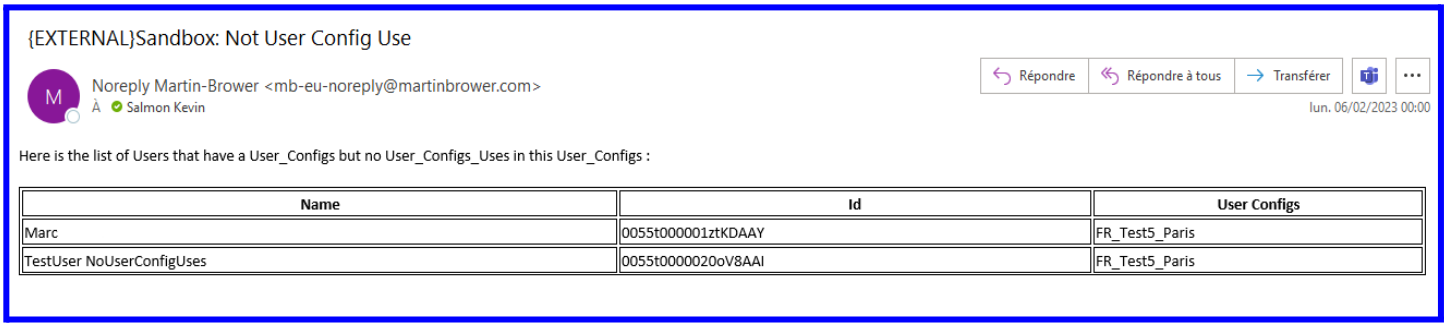
Ensuite, je crée donc mon deuxième flow, pour répondre à la deuxième condition de la problématique précédente.

Tout d'abord, on vérifie s' il y à bien des utilisateurs ou non. Cette fois ci on utilise seulement une loop, pour vérifier la condition. Pareille pour le flow précédent, j'ai dû me baser sur l'objet *account* à la place de *User*, pour filtrer sur leurs comptes associées vu qu'il y en as beaucoup moins, et par conséquent qu'il y est moins de mail. Puis, si la condition est remplie nous envoyons un mail comme pour le flow précédent.

Pour plus de détails : ***Captures d'écrans du flow :***

Voici donc le résultat des deux flows :

1er Flow:



{EXTERNAL}Sandbox: Not User Config Use

Noreply Martin-Brower <mb-eu-noreply@martinbrower.com>
À Salmon Kevin

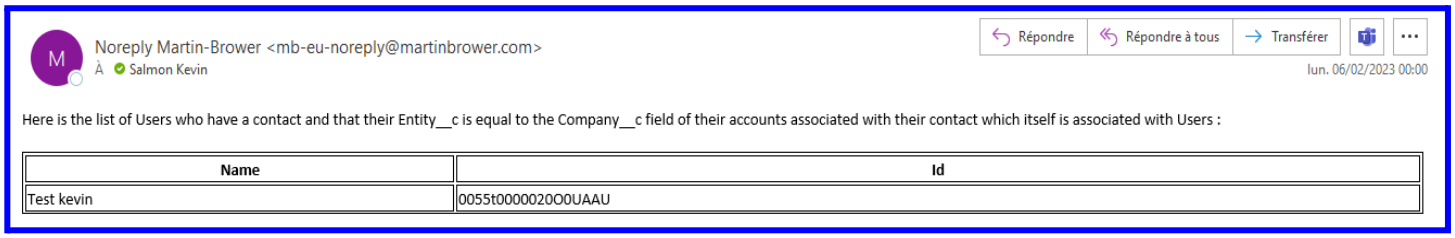
Répondre | Répondre à tous | Transférer

lun. 06/02/2023 00:00

Here is the list of Users that have a User_Configs but no User_Configs_Uses in this User_Configs :

Name	Id	User Configs
Marc	0055t000001ztKDAAY	FR_Test5_Paris
TestUser NoUserConfigUses	0055t0000020oV8AAI	FR_Test5_Paris

2ème Flow :



Noreply Martin-Brower <mb-eu-noreply@martinbrower.com>
À Salmon Kevin

Répondre | Répondre à tous | Transférer

lun. 06/02/2023 00:00

Here is the list of Users who have a contact and that their Entity__c is equal to the Company__c field of their accounts associated with their contact which itself is associated with Users :

Name	Id
Test kevin	0055t0000020O0UAAU

c.1. Difficulté rencontrée

Durant cette quatrième mission, j'ai été confronté à certaines difficultés. En effet, j'ai dû revoir complètement mon flow, pas parce qu'il n'était pas bon, mais car il fallait que je m'adapte à la manière d'où s'exécute un flow. Cependant, j'ai demandé des conseils aux membres de l'équipe et j'ai donc pu m'en sortir. La mise en œuvre de la solution a été compliquée car c'était la première fois que je crée un flow Schedule qui ne s'exécute pas comme un flow classique, et ne disposent pas des mêmes éléments. Cette expérience m'a donc permis de me développer plus. J'ai appris à résoudre des problèmes liés à l'exécution d'un programme.

d.1. Conclusion de la tâche

En conclusion, ma cinquième mission de stage consistait à créer un Flow schedule pour vérifier deux conditions dans Salesforce. La première était de vérifier la présence d'une User_Config et d'une User_Config_Use enregistrée, et la deuxième était de vérifier l'existence d'un contact avec un champ Entity__c correspondant au champ Company__c du compte associé au contact. Si les conditions sont respectées, un mail sera envoyé, affichant la liste des utilisateurs associés dans un tableau, ainsi que le message associé. Malgré certaines difficultés lors de la création de ce Flow schedule, j'ai obtenu de l'aide de l'équipe pour les résoudres et j'ai développé mes compétences en résolvant les problèmes liés à l'exécution d'un programme. Cette expérience m'a également apporté une nouvelle perspective sur la manière dont on peut utiliser les Flows, mais également savoir comment ils sont exécutés.

V. Conclusion :

Ces six semaines en tant que stagiaire dans le secteur du développement informatique Salesforce a été pour moi très enrichissante. Elle m'a permis d'avoir encore plus, un second aperçu des attentes en entreprise et dans le monde professionnel et cela après une première expérience en bts sio. J'ai pu redécouvrir un environnement professionnel riche, grâce à mon stage de l'an dernier. Cela m'a permis de m'épanouir pleinement et de monter en compétence, dans des secteurs jusque-là inconnus pour moi. J'ai pu notamment me former en partie avec plusieurs outils et avec du développement à travers la plateforme qu'est Salesforce, en approfondissant également mes connaissances du stage de l'an dernier. 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 international. J'ai encore 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. 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 six semaines d' apprentissage m'ont permis de me conforter dans mon choix de poursuite d'études. En effet, je désire poursuivre dans le développement tout en gardant aussi un œil sur le côté design d'une page web. L'entreprise Martin Brower a été pour moi une excellente redécouverte 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://www.microsoft.com/fr-fr/microsoft-teams/group-chat-software	Pouvoir communiquer avec les autres membres de l'équipe et faire des réunions

Annexe : Glossaire :

1. Déclencheurs (Trigger) :

Pour rappel, un déclencheur (ou trigger en anglais) est un mécanisme Salesforce qui permet de rendre autonome des actions, lorsqu'un événement spécifique se produit dans le système. Pour définir ces déclencheurs, il faut les définir dans des objets spécifiques, comme un Compte ou un Contact. Elles sont ensuite déclenchées par des actions, comme des mises à jour, insertion ou encore suppression de données.

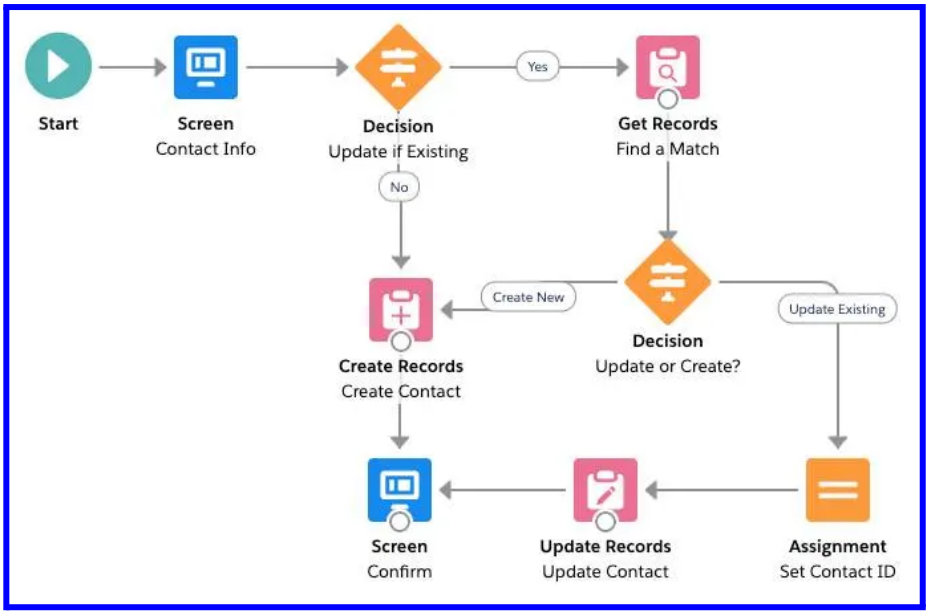
2. Contrôleur personnalisé :

Pour rappel, un contrôleur personnalisé Salesforce est un type de code Apex (Java) qui permet de gérer les actions et les logiques de l'interface utilisateur d'une page Salesforce. Il permet donc de définir des actions qui se produisent lorsque les utilisateurs interagissent par exemple avec des composants de l'interface utilisateur tels que des boutons, des liens et des champs de saisie. Elles peuvent également permettre d'effectuer des opérations de base de données, de validation de données ou de gestion des erreurs. Le but étant d'utiliser du code apex et SOQL pour pouvoir manipuler les données intéressantes.

Le Flow :

Tout d'abord, le flow se présente sous forme de carte mentale, elle comprend plusieurs modules permettant de remplir cette dernière. Les Flows sont utilisés de manière automatique, pour pouvoir automatiser certaines tâches spécifiques, comme la mise à jour des données, la suppression de ces dernières ou la création de nouveaux enregistrements.

Exemple d'un flow :



Dans ce cas-ci, ce flow détermine s'il convient de chercher un enregistrement correspondant, en fonction du choix de l'utilisateur de créer ou non un nouveau contact de façon systématique. Ensuite, il détermine si l'élément "Chercher une correspondance" a trouvé un contact correspondant ou non.

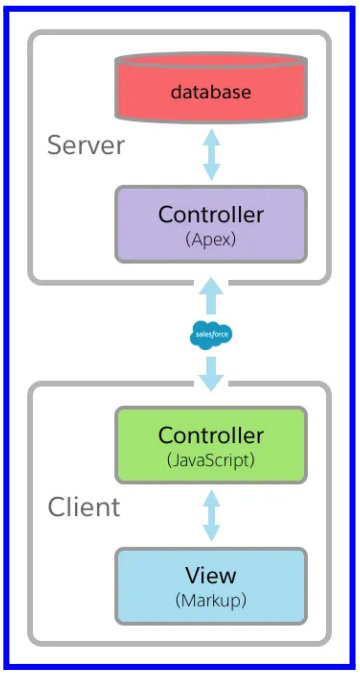
Apex :

Le langage Apex est un langage propre à Salesforce, c'est un langage de programmation orienté objet. Très similaire 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 n'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 a 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 a eu un résultat sur cette dernière. La plupart du temps, il n'y a 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.



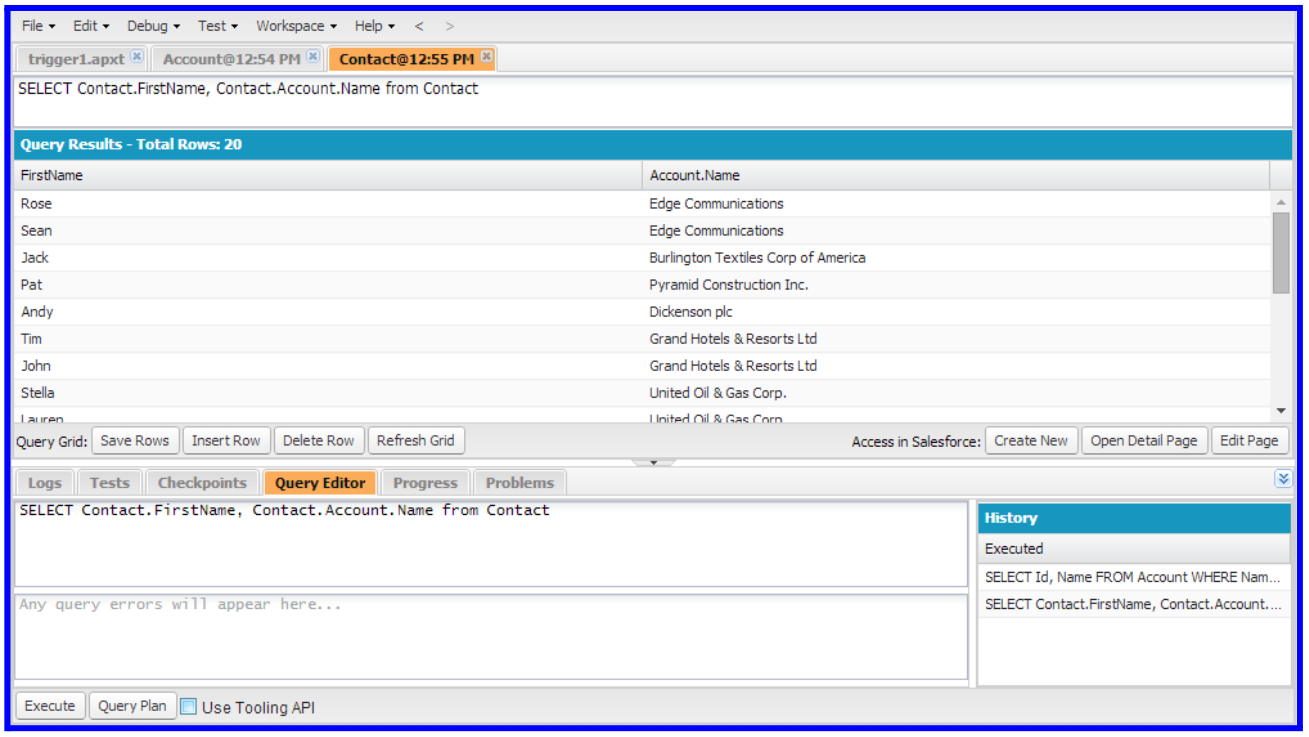
Voici un schéma résumant la place du langage Apex dans Salesforce. Nous pouvons bien voir qu’il est côté serveur. Il est donc du côté de Backend avec SOQL. C'est-à-dire le côté non visible par le client.

Batch :

Un batch Salesforce est un ensemble de tâches ou de données qui sont traitées ensemble en une seule transaction. L’avantage est de traiter un grand nombre d’enregistrements en une seule fois plutôt que de les traiter individuellement. Cela permet notamment d’améliorer les performances et de réduire le nombre d’erreurs, tout en assurant une certaine flexibilité. Dans notre cas, cela est particulièrement utile, car il y a énormément de limites Salesforce.

Le langage SOQL :

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 d’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 “Account,User,...” C’est donc une variation du SQL propre à Salesforce.
Pour utiliser le SOQl, il suffit de se rendre dans la “développée 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 étaient stockés notamment, les Accounts, Les User,...

Visual Studio Code :

Visual Studio Code est un éditeur de code gratuit, multi-plateforme, 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.

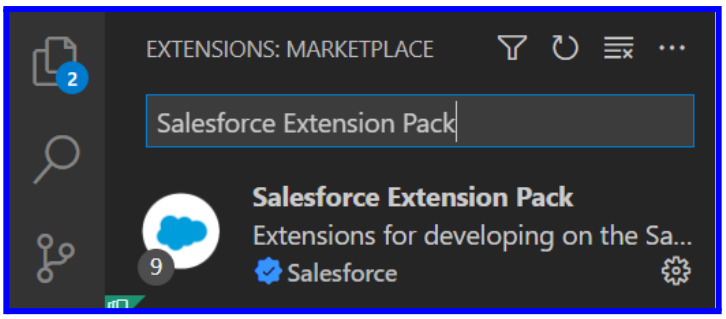
L'éditeur propose cependant plusieurs fonctions avancées qui sont intéressantes, dont :

- 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 claviers.

Lien :

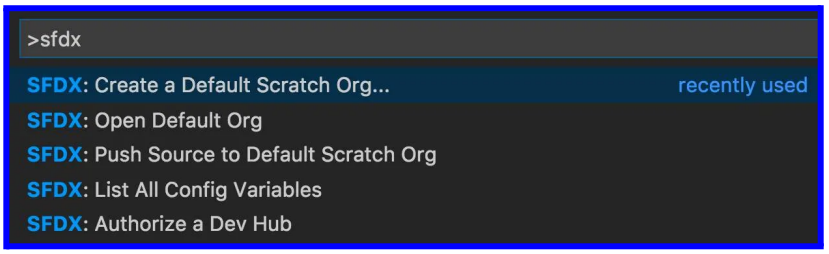
Installation de Visual Studio Code : Visual Studio Code - Code Editing. Redefined

L'avantage de ce logiciel, c'est qu'on peut ajouter une ou plusieurs extensions, parmi la 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 :

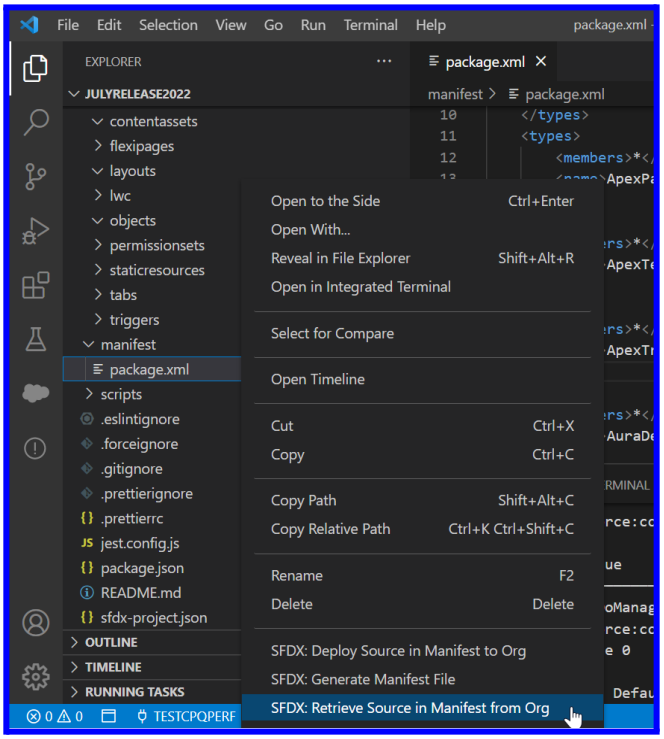
```
sfdx auth:web:login --setalias "nom_de_la_sandbox"--instanceurl
"url_de_la_page_Salesforce"--setdefaultusername
```

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 a maintenant 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.

Langages utilisés sous une page Salesforce :

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 géré la plupart du temps par le html dans ce cas-là, en utilisant notamment des composants déjà existants 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, le JavaScript était seulement un langage de script simple. D'autant plus que pendant longtemps il n'y a eu de mise à jour, ni même d'améliorations quelconques. 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 la version ES6, qui a 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 Martin Brower utilise du Javascript moderne.

Validation Rules :

Ces règles de validation demandent un peu de connaissance en 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.

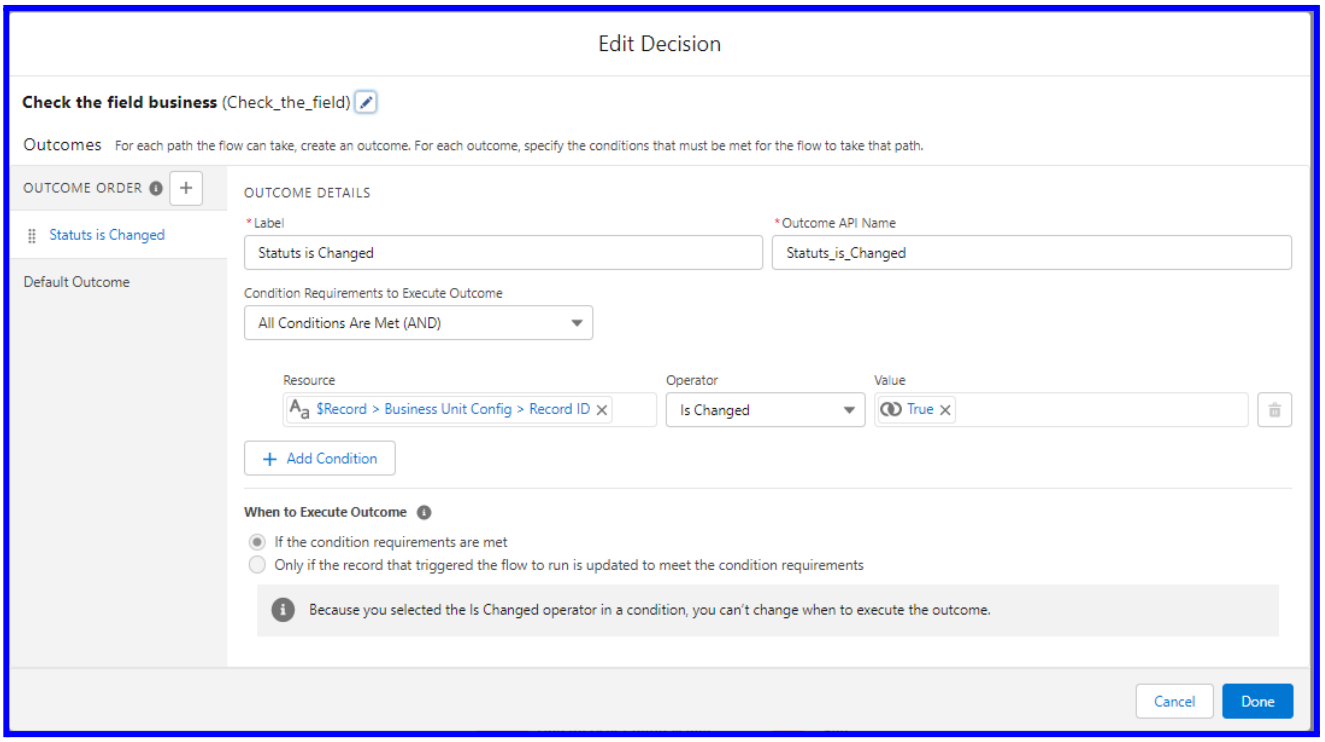
Annexe : Les différentes captures d'écrans utiles :

a) Explication étape par étape :

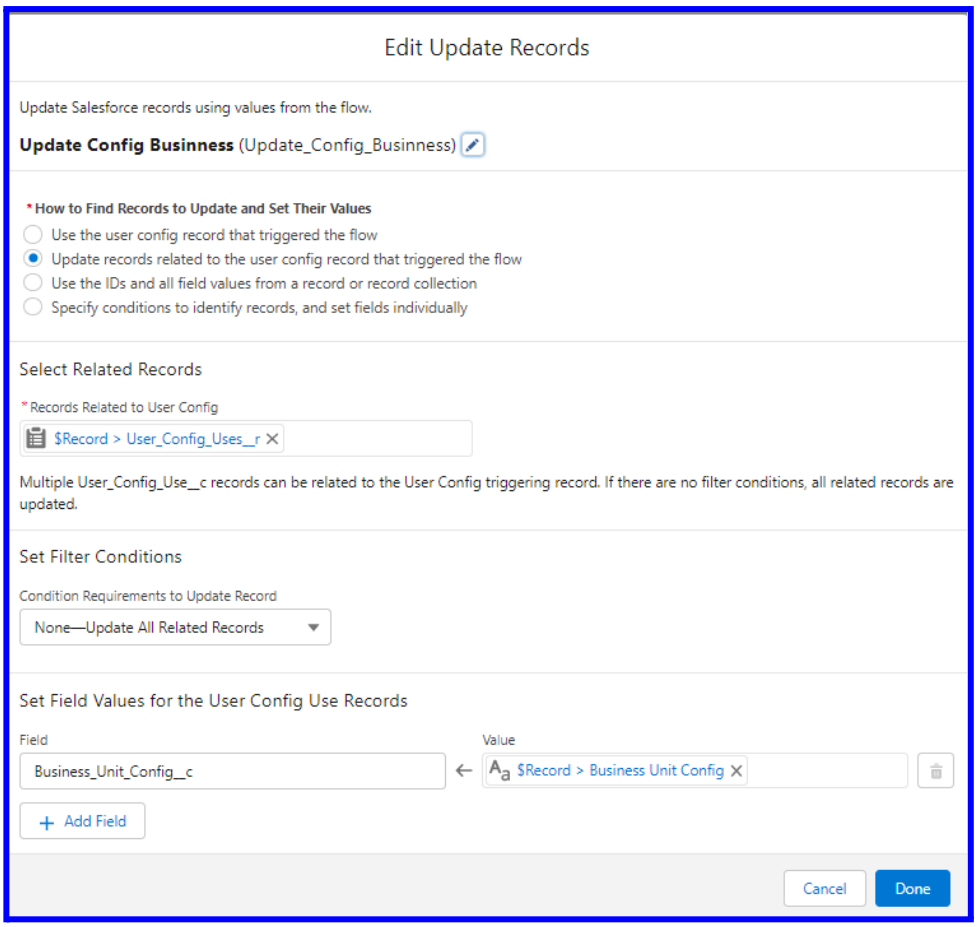
Voici les étapes détaillées pour créer un flow Salesforce pour gérer la vérification de la mise à jour de la base de données des utilisateurs configurés en utilisant deux décisions:

1. Créez un nouveau flow dans Salesforce en utilisant l'outil de création de flux.

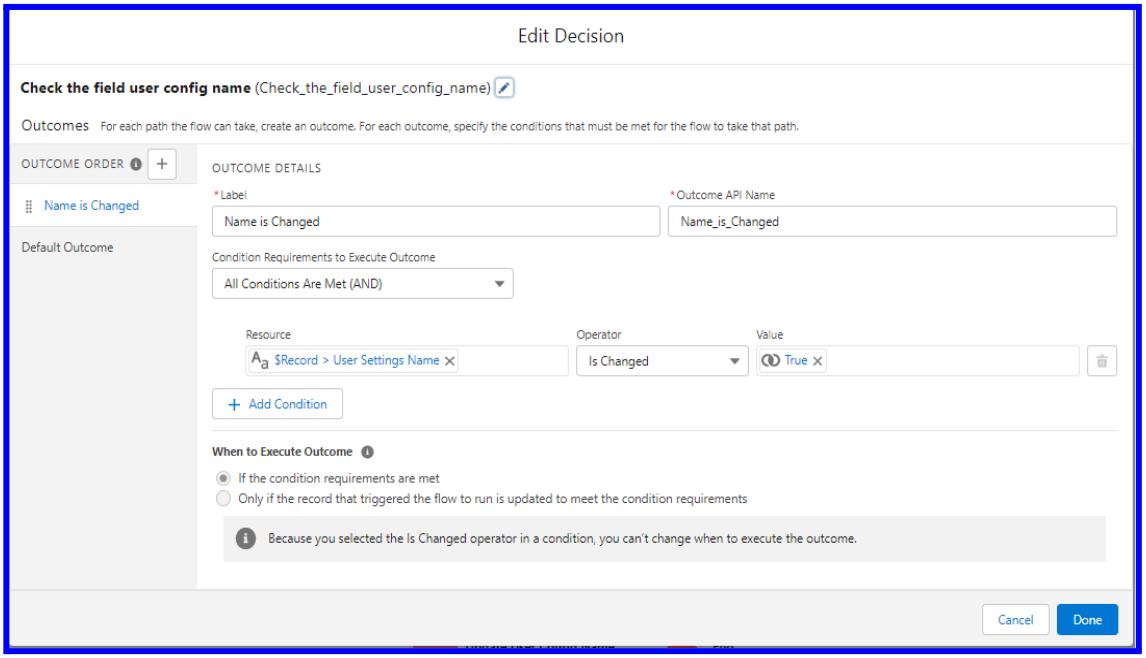
2. Ajoutez une décision "if" pour la première vérification. Cette décision doit avoir pour condition de comparer les informations de l'objet *User_Config* et de l'objet *User_Config Used__r* (l'Utilisateur). On utilise l'opérateur *IsChanged*, pour vérifier si le champ *Business Unit Config* a été modifié.



3. Si la condition est remplie, c'est-à-dire si la mise à jour du champ Business Unit Config a été correctement effectuée, alors le flux continue son exécution jusqu'à l'update records. Sinon, il ne se passe rien. Dans l'update records, on se charge d'update les différents éléments concernés, pour bien les mettre à jour :

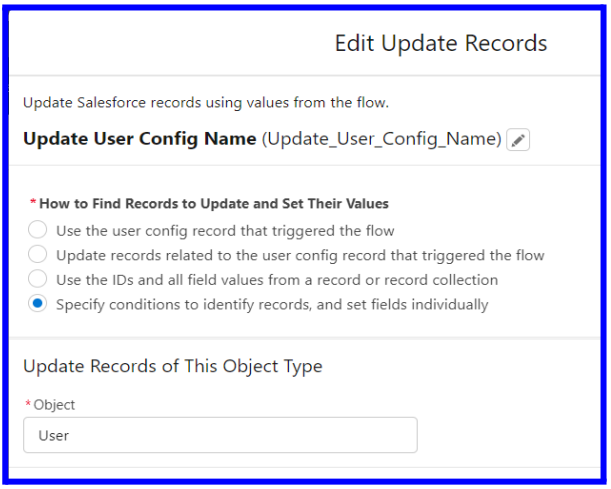


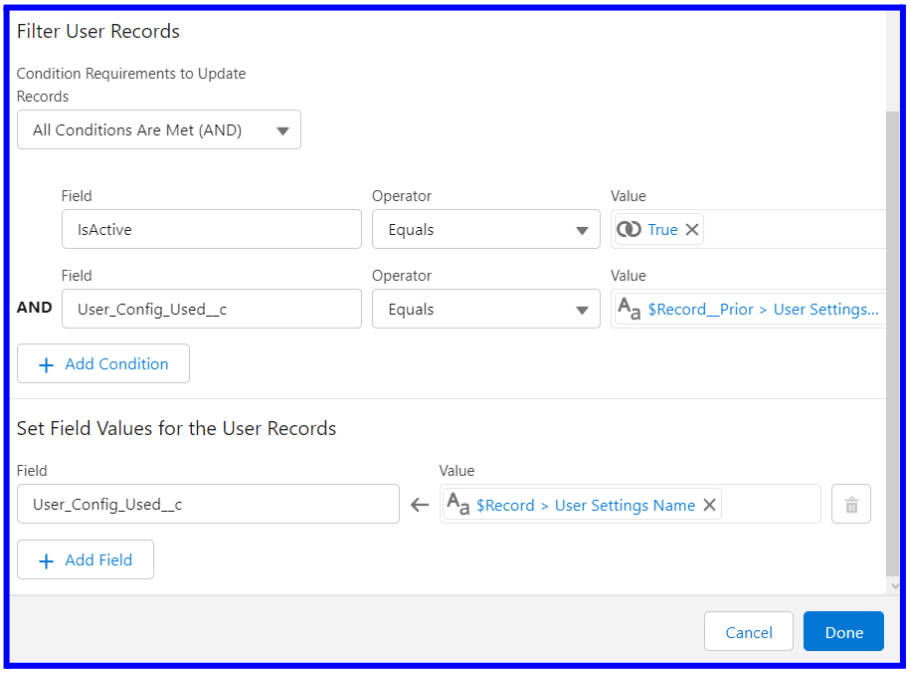
4. Ajoutez une deuxième décision "if" pour la deuxième vérification. Cette décision doit avoir pour condition de comparer, la liste des utilisateurs configurés avec leur nom de configuration.



La condition est égale à True, si le nom de la configuration d'un utilisateur a été changé.

5. Si la condition est remplie, c'est-à-dire si le nom de la configuration d'un utilisateur a été changé alors le flux continue son exécution jusqu'à l'update records. Sinon, il ne se passe rien. Dans l'update records, on se charge d'update les différents éléments concernés, pour bien les mettre à jour :





Dans le cas présent, l'update record se charge de mettre à jour le nom des configurations utilisées par les utilisateurs en utilisant le champ *User_Config_Used__c*. L'update record vérifie également si les utilisateurs sont actifs en utilisant le champ IsActive.

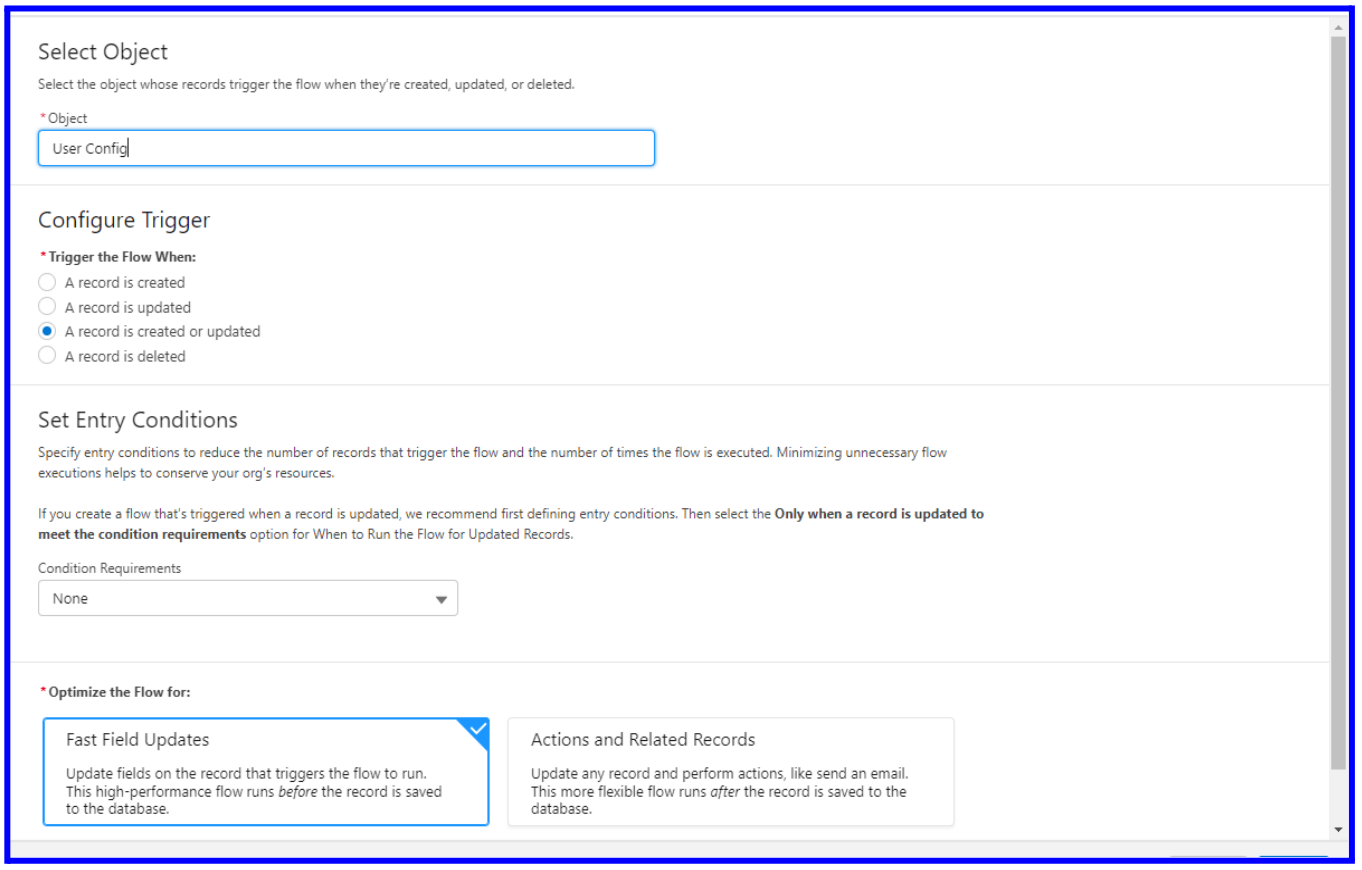
Lorsque le nom d'une configuration est modifié, l'update record parcourt tous les enregistrements d'utilisateurs associés à cette configuration et met à jour le nom de la configuration dans le champ *User_Config_Used__c* pour chacun d'entre eux. En même temps, l'update record vérifie que l'utilisateur est actif en utilisant le champ IsActive, cela permet de s'assurer que seuls les utilisateurs actifs auront leur nom de configuration modifié.

Cette action d'update record dans un flow permet de maintenir à jour les informations des utilisateurs en temps réel.

6. Testez le flow en utilisant des données de test pour vous assurer qu'il fonctionne correctement.

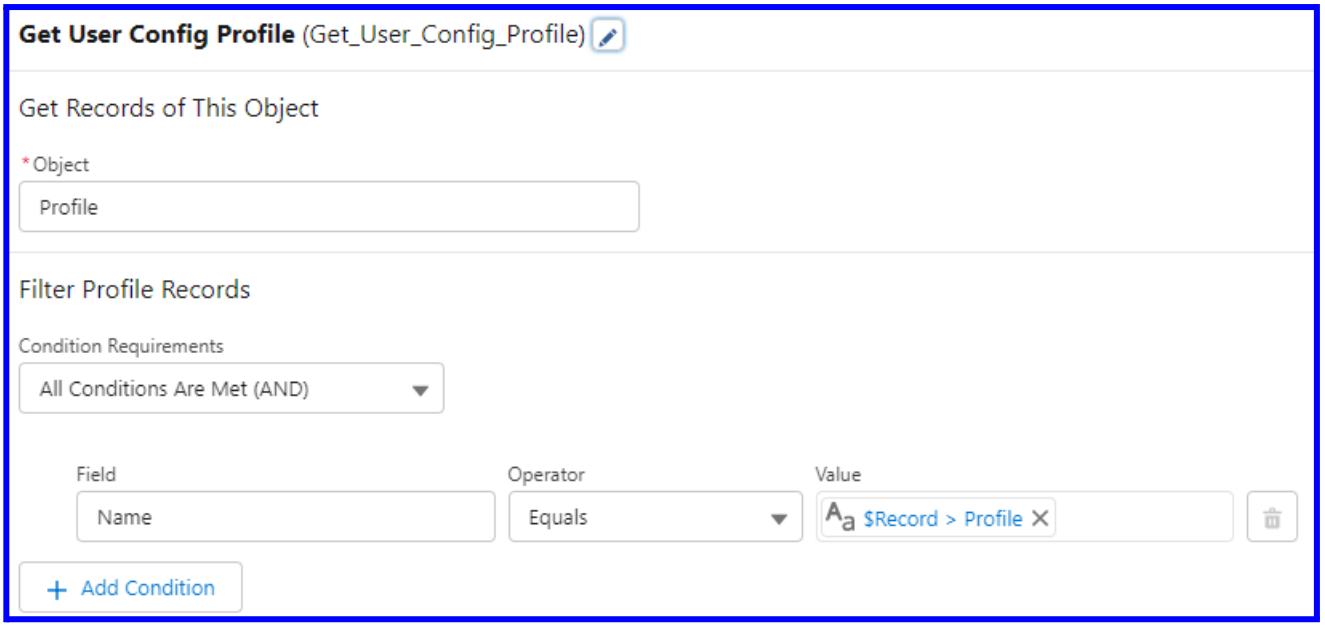
7. Enregistrez et publiez le flow pour qu'il soit utilisable dans Salesforce.

b) Captures d'écrans liée au flow :

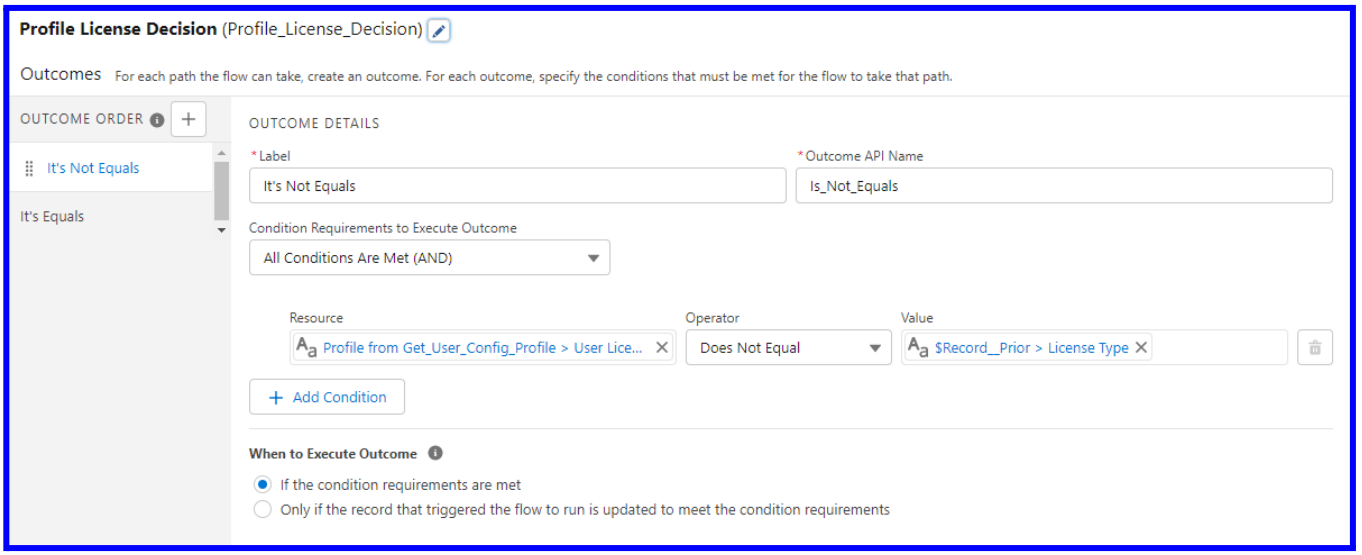


Pour la première étape, on n'oublie pas de définir l'objet *User Config*. On veut que le flow puisse s'activer en cas de création ou de mise à niveau, on sélectionne donc la troisième option.

Point important : Nous sélectionnons l'onglet : *Fast Field Updates*, car sinon la validation rules va s'exécuter avant le flow.



Pour cette partie, on récupère toutes les données de l'objet *Profile*, en fonction du nom du profil saisie dans l'interface utilisateur.

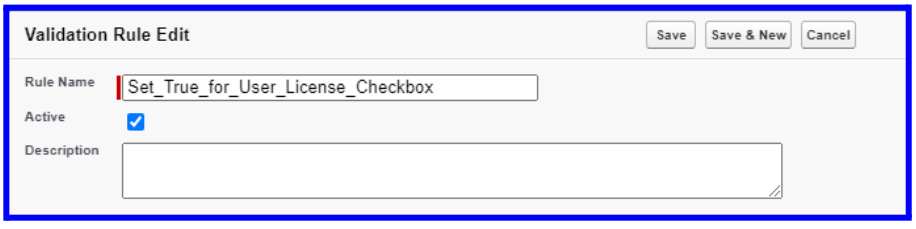


Si cet décision est bonne, nous arrivons sur :

Cela permet de définir la checkbox que j'ai définie au préalable sur true. Cette checkbox est invisible sur l'interface utilisateur, pour éviter qu'on puisse interagir avec. Et cette checkbox, permet de faire justement le lien avec la validation rules.

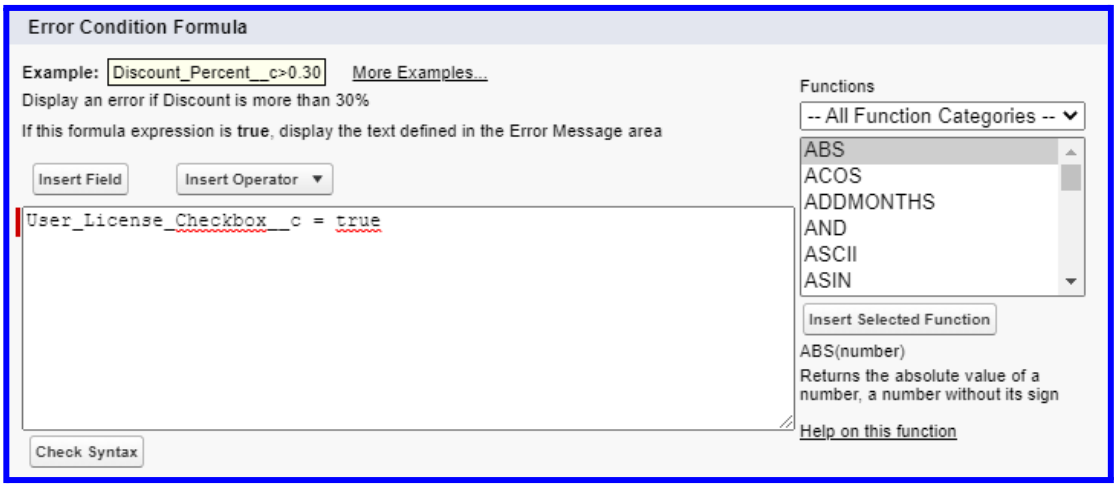
c) Captures d'écrans liés à la validation rules :

Cela se présente donc comme ceci :



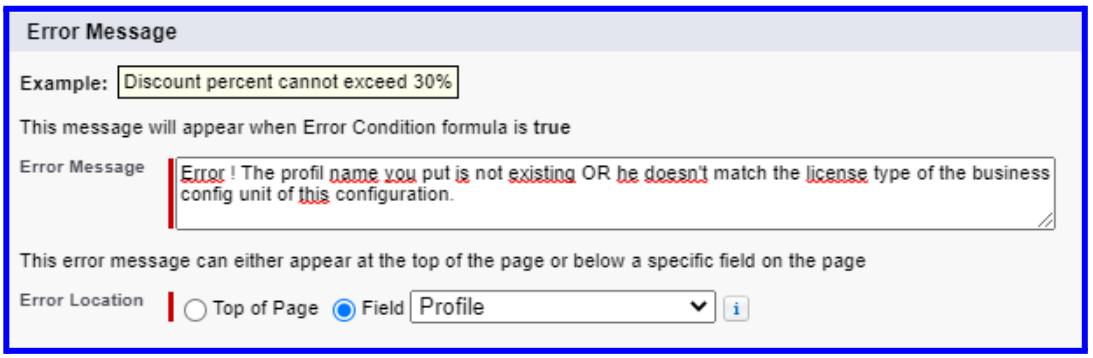
Une première partie où l'on donne le nom de la règle et sa description.

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



Sachant qu'il y a beaucoup de fonctions par défaut et des fonctions propres à une validation rules.

La troisième partie se concentre sur la gestion de la position de l'erreur et le contenue du message :

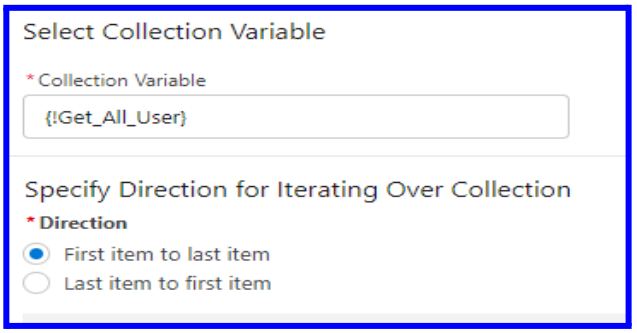


Le message d'erreur s'affiche quand la méthode renvoie True, et je décide de l'afficher sur le champ *Profile*.

d) 1/ Captures d'écrans liés au flow Schedule Partie 1 :

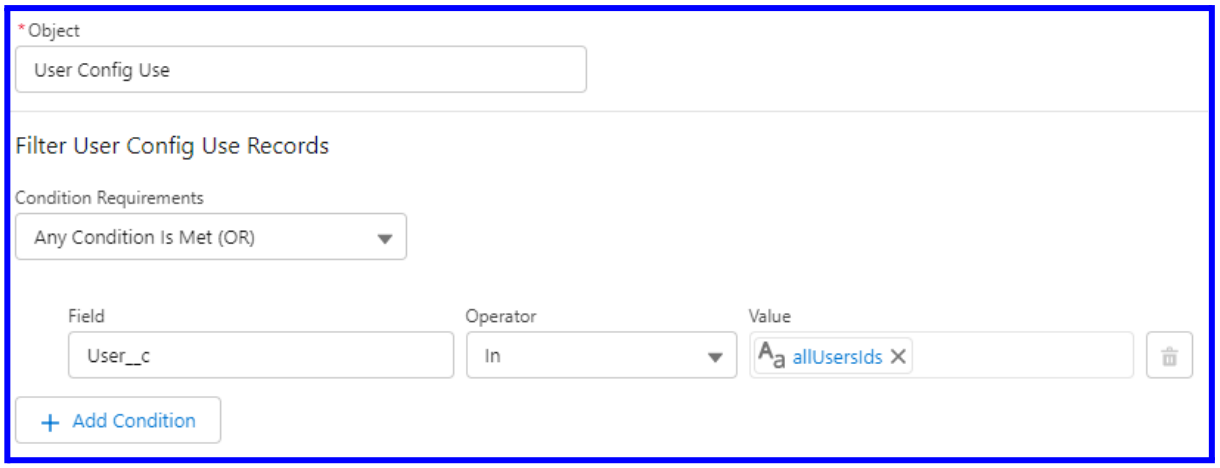
Tout d'abord, nous configurons le flow sur l'objet *User Config*, et définissons l'heure sur minuit. Comme ça le flow pourra s'exécuter tous les jours à minuit. Nous récupérons d'abord la liste de tous les *Users* en apportant des filtres avec un *get Record* :

Nous voulons tous les utilisateurs actifs et les utilisateurs possédant une *User Config*, nous précisons d'ailleurs tous les enregistrements en cochant cette option. Ensuite j'écris une décision pour bien vérifier s' il y à des utilisateurs, si oui nous pouvons continuer le programme. Je met ensuite une boucle qui va parcourir tout les enregistrements du premier *getRecord* :

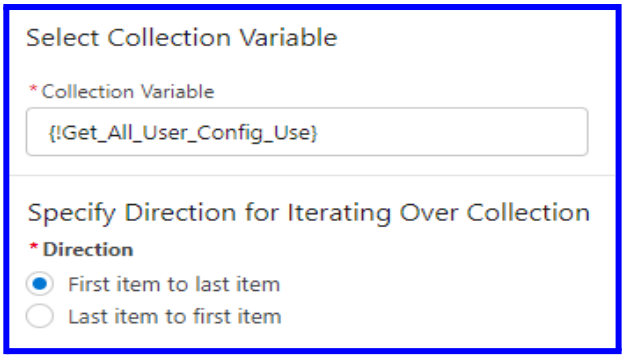


Dans cette boucle, nous récupérons tous les id des utilisateurs, et toutes leurs informations dans deux variables différentes : *allUsersIds, AllUsers*.

Ensuite je remet un *getRecord*, permettant de voir si les id stockées dans la variable *allUsersIds*, apparaissent dans l'objet *User Config Use*, depuis le champ *User__c* :



Je recrée ensuite une deuxième boucle, mais cette fois-ci elle va parcourir le *getRecord* ci-dessus :



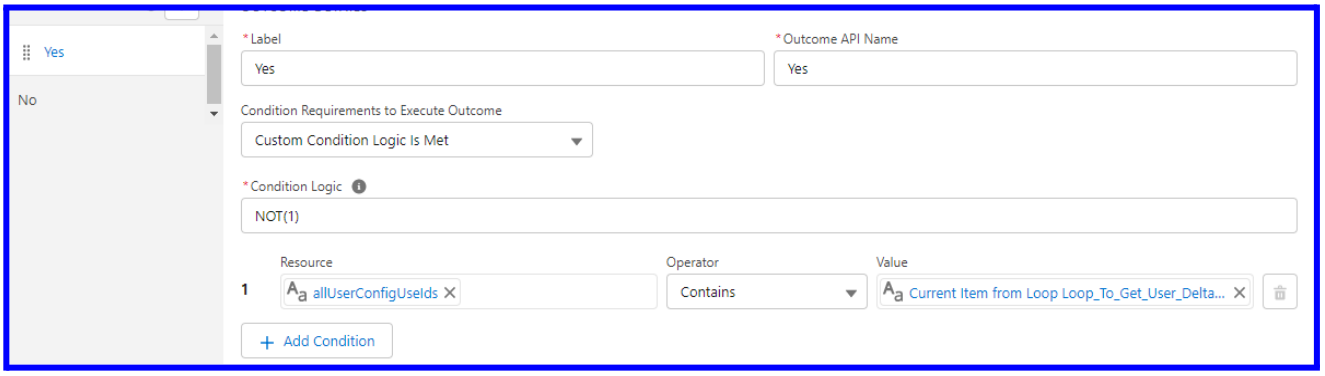
Dans cette boucle je crée une autre variable (*allUserConfigUseIds*), et j'ajoute les id résultant de la boucle.

J'ajoute ensuite le début de mon tableau en créant trois variables : *table*, *tableColumns* et *tableRow*. Dans la variable *table*, on instancie seulement un tableau en html :

Dans la variable *tableColumns*, on ajoute les colonnes dans le tableau :

```
<tr>  <th style="border: 1px solid black;">Name</th>  <th style="border: 1px solid black;">Id</th> <th style="border: 1px solid black;">User Configs</th></tr>
```

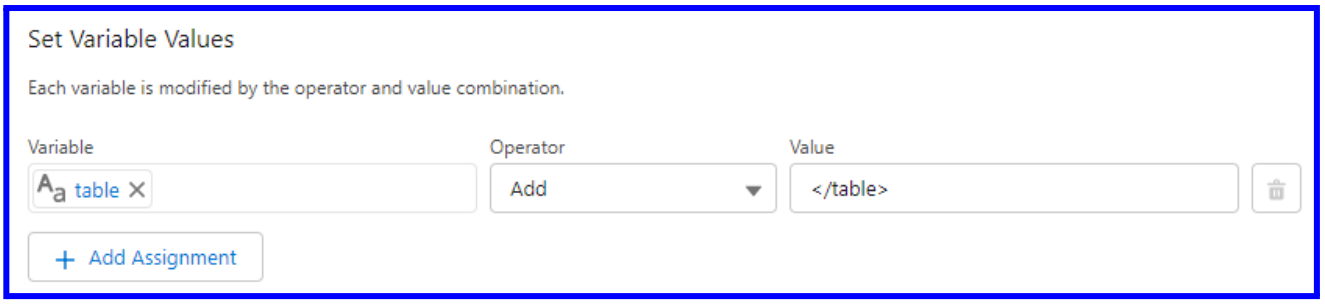
Une fois instancié, je crée une troisième boucle qui parcourt la variable *AllUsers*. Dans cette boucle je crée une décision pour vérifier si *allUserConfigUseIds* ne contient pas les id de la boucle :



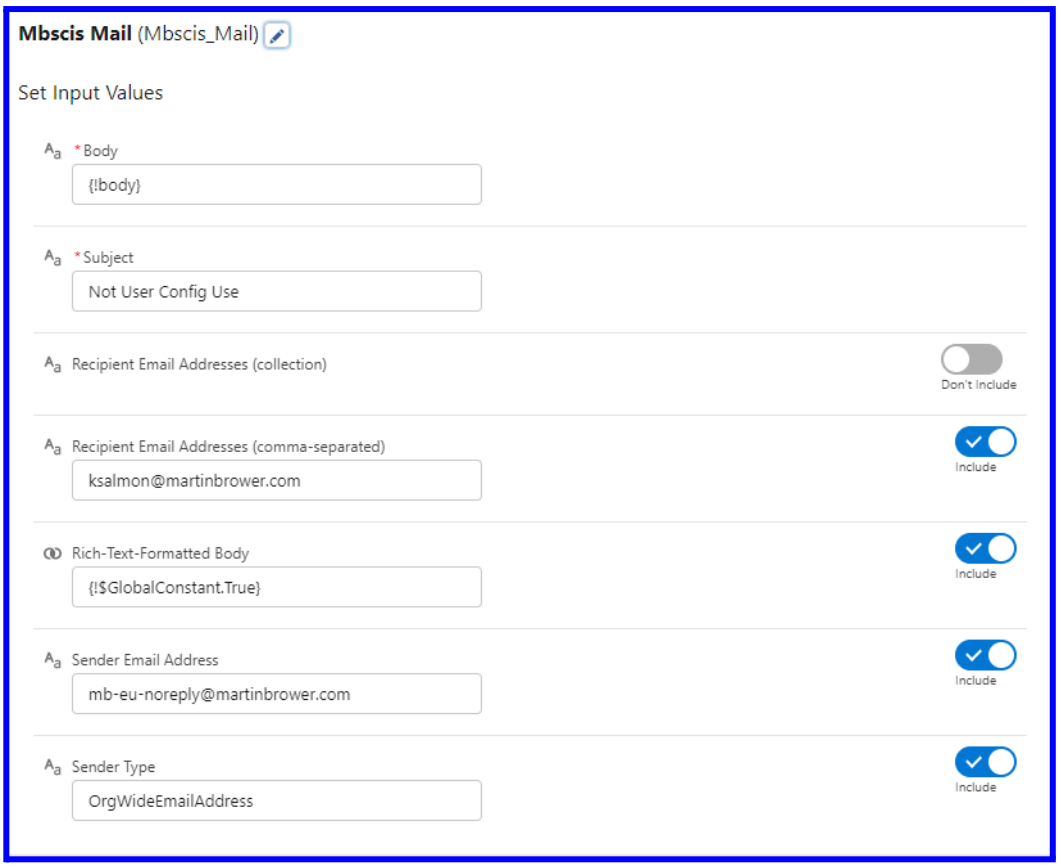
Si la décision est respectée, j'ajoute les valeurs de la boucle au tableau correspondant :
Dans la formule *tableRow*, on ajoute les valeurs dans le tableau :

```
'</th><td style="border: 1px solid black;">' + {!Loop_To_Get_User_Delta.Name}+
'</th><td style="border: 1px solid black;">' + {!Loop_To_Get_User_Delta.Id} +
```

Une fois la boucle terminée, je ferme le tableau avec :



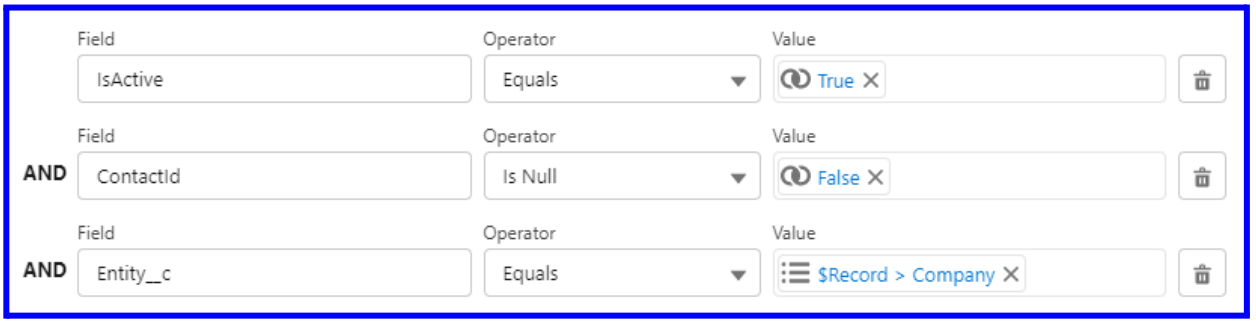
Puis j'envoie un mail avec une "action" dans le flow. Cela se présente comme ceci :



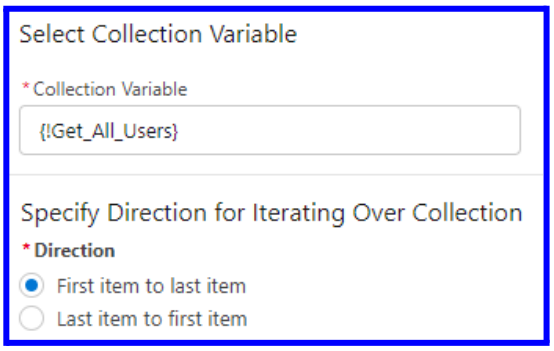
Il y à tout d'abord le corp du message ou l'on définit le contenue du e-mail, le titre de l'e-mail. On définit ensuite le destinataire et l'expéditeur, et le type d'envoi. J'ai mis *OrgWideEmailAddress*, pour laisser par défaut.

d) 2/ Captures d'écrans liés au flow Schedule Partie 2 :

Tout d'abord, nous configurons le flow sur l'objet *Account*, et définissons l'heure sur minuit. Comme le flow précédent, nous récupérons d'abord la liste de tous les *Users* en apportant des filtres avec un *get Record* :



Nous voulons tous les utilisateurs actifs et les utilisateurs possédant un *Contact*, nous précisons d'ailleurs tous les enregistrements en cochant cette option. Ensuite j'écris une décision pour bien vérifier s' il y à des utilisateurs, si oui nous pouvons continuer le programme. Comme pour le flow précédent : nous créons un tableau qui va être envoyé par mail par la suite. Je crée d'abord une boucle sur le *getRecord* d'avant :



Dans cette boucle, je crée une décision permettant de savoir si le champ *Entity__c* de la boucle est bien égale au champ Company__c du compte associé au contact, lequel était lui-même associé à l'utilisateur. Si oui nous continuons le programme en ajoutant les valeurs de la boucle dans le tableau.
Puis nous envoyons le mail comme précédemment.

1. Mission en parallèle de ma mission 1 :

Au cours de ma mission principale, j'ai été chargé de créer des ressources statiques depuis Salesforce en utilisant un diaporama de 29 diapositives. Chacune de ces diapositives a été renommée en utilisant un nom de format "MbSyncTrainingSlide 1 jusqu'à "MbSyncTrainingSlide 29" pour les identifier. Cette tâche a été réalisée pour permettre à l'équipe de disposer d'un support de formation sur les fonctionnalités de la plateforme Salesforce.

J'ai également été amené à effectuer des modifications sur un flow déjà existant pour afficher les 29 diapositives du diaporama. J'ai ajouté 2 décisions sur le flow pour permettre une navigation fluide entre les diapositives et permettre de visualiser les différentes étapes de la formation. La première décision est pour vérifier si l'utilisateur a bien visualisé tous les slides et la seconde décision est pour identifier quel marché il est, cela permet de personnaliser la formation pour chaque utilisateur.

Annexe : Les différentes parties comportant du code :

1. Programme du début :

```
global with sharing class Salesforce_limits_Batch implements Database.Batchable<SObject>, Database.Stateful{

    Datetime myDateTime = datetime.now();
    String todayDateGMTString = myDateTime.formatGMT('dd/MM/yyyy');
    Date todayDateGMT = Date.parse(todayDateGMTString);
    private Integer limitation = null;
    private Integer usage = null;

   global Database.Querylocator start(Database.BatchableContext context){
        Integer listLimits = [SELECT Count() FROM Salesforce_limits__c WHERE Day__c=:todayDateGMT];

         if(listLimits<1){
            Salesforce_limits__c newLimit = new Salesforce_limits__c();
            newLimit.Limit_name__c = 'SingleEmail';
            newLimit.Day__c = todayDateGMT;
            insert newLimit;
        }
        return Database.getQueryLocator([SELECT Id,Limit_name__c,Limit__c,Usage__c,Day__c FROM Salesforce_limits__c WHERE Day__c=:todayDateGMT Limit 1]);
    }
 global void execute(Database.BatchableContext context, List<Salesforce_limits__c> scope){
        List<Salesforce_limits__c> listToUpsert = new List<Salesforce_limits__c>();

        if(scope.size()>0){
            List<System.OrgLimit> limits = OrgLimits.getAll();
            for (System.OrgLimit aLimit: limits) {
                if(aLimit.getName().equals('SingleEmail')){
                      limitation = aLimit.getLimit();
                      usage = aLimit.getValue();
                      break;
                 }
            }

            for(Salesforce_limits__c curentlimit :scope){
                  currentLimit.Usage__c = usage;
                  currentLimit.Limit__c = limitation;
                  if(!listToUpsert.contains(currentLimit)){
                        listToUpsert.add(currentLimit);
                  }
            }


                    update listToUpsert;

        }
    }


    global void finish(Database.BatchableContext context){
        GEN_ScheduledJob_SendEmail.sendEmailIfError(context);
    }
}
```

2. Programme finale :

J'ai séparé le programme en plusieurs parties :

```
global with sharing class Salesforce_limits_Batch implements
Database.Batchable<SObject>, Database.Stateful{

    Datetime myDateTime = datetime.now();
    String todayDateGMTString = myDateTime.formatGMT('dd/MM/yyyy');
    Date todayDateGMT = Date.parse(todayDateGMTString);
    private Integer limitation = null;
    private Integer usage = null;
   // old  List<String> listLimitname = new List<String>{'SingleEmail'};   //Crée une
liste dynamique comportant tous les noms de configuration

    Map<String,String> mapLimitName = new Map<String, String>();
    //Déclaration de la variable mapLimitName qui est un objet de type Map qui
contient des strings en clés et des valeurs qui contiendra les noms des limites
```

La première partie consiste à implémenter les interfaces *Database.Batchable* et *Database.Stateful*, pour permettre à la classe de fonctionner en mode Batch Apex, pour traiter un grand nombre d'enregistrements en même temps, et ceci de manière autonome en arrière-plan. Nous pouvons donc suivre plus facilement les erreurs, et consommer moins de ressources.
La classe stocke d'abord la date en format GMT dans une variable de type *DateTime*, puis la convertit en format de date et la stocke dans une variable de type *Date*. La classe définit également deux variables sur *null*, qui vont contenir la limitation d'une limite et son taux d'utilisation actuel. Enfin, elle va déclarer une *Map* permettant de stocker les noms de limites. Une Map est tout simplement comme un tableau associatif en PHP, elle permet de stocker des valeurs avec des clés qui lui sont associées. La première partie est surtout là pour l'initialisation des variables.

```
global Database.Querylocator start(Database.BatchableContext context){
             // old : Integer listLimits = [SELECT Count() FROM Salesforce_limits__c
WHERE Day__c=:todayDateGMT];


            mapLimitName.put('SingleEmail','OrgLimits');
            mapLimitName.put('Customer Community Plus Logins','0UT68000000TN1GGAW');

            system.debug('1er passage');
            //List<Salesforce_limits__c> listLimits = [SELECT Count() FROM
Salesforce_limits__c WHERE Day__c=:todayDateGMT];

            //récupérer le nommbre d'enregistrementsdans l'objet Salesforcel_limits__c
```

```
pour limiter le nombre de requête sur le jour même
            List<Salesforce_limits__c> listLimits = [SELECT id,name,day__c FROM
Salesforce_limits__c WHERE Day__c=:todayDateGMT];

            //Déclaration d'une map namelimite qui a pour clé le nom de la limite et
pour valeur l'enregistrement de Salesforce_limits__c
            map<String, Salesforce_limits__c> namelimite = new map<String,
Salesforce_limits__c>() ; //(Name de la limite, record salesforce limits)


            for(Salesforce_limits__c lim : listLimits) {
//enregistrer avec des noms plutôts que des id
                    //Ajout de l'enregistrement de Salesforce_limits__c dans la map
namelimite avec comme clé le nom de la limite
                    namelimite.put(lim.Limit_name__c, lim);

             }
            //Crée une liste de retour de Salesforce_limits__c
            List<Salesforce_limits__c> listoreturn = new list<Salesforce_limits__c>();

            //Pour chaque nom de limite dans la map "mapLimitName"
            for (String limitename :mapLimitName.keySet()){
      //for each limitname
                if(!namelimite.containsKey(limitename)){                    //Si le nom
de limite n'est pas présent dans la map "namelimite"
                                    //   sinon
                    Salesforce_limits__c newLimit = new Salesforce_limits__c();
//Crée un nouvel enregistrement de Salesforce_limits__c
                    newLimit.Limit_name__c = limitename;
                    newLimit.Day__c = todayDateGMT;
                    newLimit.Method__c = mapLimitName.get(limitename);//Ajoute la
méthode de limite à l'enregistrement
                    insert newLimit;//Insère l'enregistrement dans Salesforce
                }


            }

        //end for
        /* old :
         * if(listLimits<1){
                    Salesforce_limits__c newLimit = new Salesforce_limits__c();
            newLimit.Limit_name__c = 'SingleEmail';
            newLimit.Day__c = todayDateGMT;
                    insert newLimit;
        }
        return Database.getQueryLocator([SELECT
Id,Limit_name__c,Limit__c,Usage__c,Day__c FROM Salesforce_limits__c WHERE
```

```
Day__c=:todayDateGMT Limit 1]);   */

        //Retourne les enregistrements de Salesforce_limits__c pour la date
d'aujourd'hui
        return Database.getQueryLocator([SELECT
Id,Limit_name__c,Limit__c,Usage__c,Day__c FROM Salesforce_limits__c WHERE
Day__c=:todayDateGMT]);  //retourne les noms de limites par jours
    }
```

Cette seconde partie utilise la map initialisée dans la première partie, en lui ajoutant les noms de limites custom. De base il y avait bien évidemment *SingleEmail*, mais j'ai rajouté la limite : *Customer Community Plus Logins*. Il y à également présence de plusieurs boucles *for*, pour vérifier si des enregistrements existent déjà dans l'objet *Salesforce_limits__c*. Si ces enregistrements n'existent pas, ils en créent de nouveaux en utilisant les informations stockées dans la map précédente. Ainsi, ces nouveaux enregistrements sont créés avec le nom de la limite, leurs date actuelles et la méthode associée. Enfin le programme retourne les enregistrements de *Salesforce_limits__c* pour la date d'aujourd'hui.

```
global void execute(Database.BatchableContext context, List<Salesforce_limits__c>
scope){

             //Création d'une liste pour enregistrer les objets modifiés
            List<Salesforce_limits__c> listToUpsert = new
List<Salesforce_limits__c>();
             //Verification si la liste est vide
            if(scope.size()>0){
                //Récupération de la liste des limites de l'org
                List<System.OrgLimit> limits = OrgLimits.getAll();
                //Récupération de la liste des utilisations des limites de
l'entreprise
                List<TenantUsageEntitlement> listtenantusage = [SELECT Id,
CurrentAmountAllowed, AmountUsed, MasterLabel FROM TenantUsageEntitlement WHERE Id IN
:mapLimitName.values()];
             // Affichage d'un message de débogage pour le résultat de la liste
d'utilisation des limites de l'entreprise

             // Boucle pour chaque objet Salesforce_limits__c dans la liste
             for(Salesforce_limits__c currentLimit : scope){
                if(mapLimitName.get(currentLimit.Limit_name__c) == 'OrgLimits'){
                   // Boucle pour chaque limite de l'org
                   for (System.OrgLimit aLimit: limits) {
                      if(aLimit.getName().equals(currentLimit.Limit_name__c)){
                         // Récupération de la limite
```

```
                        limitation = aLimit.getLimit();

                        // Récupération de l'utilisation
                        usage = aLimit.getValue();
                        break;
                    }
                }
            }else{
                // Vérification si la liste d'utilisation des limites de
l'entreprise n'est pas null
                if (listtenantusage != null){
                    // La liste d'utilisation des limites de l'entreprise n'est pas
null
                    // Boucle pour chaque limite d'utilisation de l'entreprise
                    for(TenantUsageEntitlement currentLimitTenant :
listtenantusage){
                        if(currentLimitTenant.Id ==
mapLimitName.get(currentLimit.Limit_name__c)){
                            // Récupération de la limite
                            limitation =
Integer.valueOf(currentLimitTenant.CurrentAmountAllowed);

                            // Récupération de l'utilisation
                            usage = Integer.valueOf(currentLimitTenant.AmountUsed);
                            break;
                        }
                    }
                }
            }
            // Mise à jour de l'utilisation
            currentLimit.Usage__c = usage;
            // Mise à jour de la limite
            currentLimit.Limit__c = limitation;
             //si la liste ne contient pas l'objet currentlimit
                if(!listToUpsert.contains(currentLimit)){
                    //ajouter le dans la liste
                    listToUpsert.add(currentLimit);
                }
            }
            //mettre à jour tous les objets
            update listToUpsert;
        }
    }
```

Cette troisième partie du programme permet de traiter chaque lot de données renvoyées par la méthode *start*. Le but étant de mettre à jour les données dans l'objet *Salesforce_limits__c* avec les

informations de la deuxième méthode. Il utilise une liste pour stocker les limites de OrgsLimits, et une liste “listtenantusage” pour stocker les informations de la limite custom. Il y a ensuite une boucle *for* qui permet de vérifier si la *map* de la première partie contient une limite située dans OrgsLimits, si c’est le cas le programme fait exactement comme avant, sinon il parcours la liste “listtenantusage” pour enregistrer les informations contenue dans la liste dans les variables limitation et usage. Ensuite, tous les objets sont mis à jour dans la liste : “listToUpserrt”, puis dans la base de données.

```
      //envoyer un e-mail en cas d'erreur
    global void finish(Database.BatchableContext context){
        GEN_ScheduledJob_SendEmail.sendEmailIfError(context);
    }
}
```

Cette quatrième partie sert à envoyer un e-mail en cas d’erreur lors de l'exécution du batch. Elle permet également d’indiquer la fin du batch avec la méthode *finish*.