www.isl.be

INSTITUT SAINT-LAURENT
enseignement de promotion sociale

rue Saint-Laurent 33 - 4000 LIEGE
04 223 11 31

MonLead.

Malmedy, le 17/06/2022

Travail de fin d'études présenté par :

GABRIEL Pierre

En vue de l'obtention du brevet de l'enseignement supérieur de

WebDeveloper

Enseignement supérieur économique de promotion sociale et de type court

Table des matières.

1 INTRODUCTION

1.1 Projet :

Le projet MonLead a été développé dans le cadre de mon stage au sein de l'entreprise IntoTheWeb à Angleur.

Ce projet est un proof of concept destiné à tester le fonctionnement de différentes technologies.

Les objectifs premiers de ce projet étaient de tester un réseau basé sur une blockchain privée grâce à l'outil « HyperLedger Fabric ».

Le service MonLead comprend différents intervenants. L'administrateur (Into the web), des gestionnaires (sociétés proposant des services), des apporteurs d'affaires et des leads (clients).

L'apporteur d'affaires sera rémunéré pour chaque contrat créé avec un lead qu'il aura apporté.

Les contrats seront sauvegardés dans la blockchain et les montants versés aux apporteurs d'affaires seront affichés et calculés en backend et frontend via des « smarts-contrats » du réseau Hyperledger Fabric.

1.2 IntoTheWeb :

IntoTheWeb, c'est bien plus qu'une simple agence de développement. Depuis plus de 10 ans, ils accompagnent de A à Z dans l'élaboration de projets mobile et Web. Situé à Liège, l'équipe travaille pour fournir le meilleur d'elle-même afin de livrer une application à la hauteur des attentes du client.

IntoTheWeb, c'est une famille composée de développeurs iOS et Android, de développeurs back-end et front-end, de développeurs junior et senior ainsi qu'une équipe projet. Tous les membres de l'entreprise ont suivi une formation reconnue et continuent de se former pour acquérir davantage de compétences et d'expériences.

1.3 Contraintes techniques :

Ce travail de fin d'études étant la continuité de mon projet de stage, j'ai été contraint d'apprendre et d'utiliser les technologies utilisées au sein de l'entreprise IntoTheWeb. Les technologies principales sont le langage Ruby et le framework Ruby On Rails.

Le choix d'outils supplémentaires tels que HyperLedger Fabric et les services web AWS d'Amazon m'ont empeché d'utiliser certains outils embarqués dans Ruby On Rails. Beaucoup des ressources concernant ma combinaison de technologies étaient fort dépassées. Cela a donc demandé un gros travail de recherches pour résoudre certains problèmes.

2 DESCRIPTION DU SITE

2.1 Thème et brève description :

Le projet MonLead est avant tout un « proof of concept ». Celui-ci ne sera pas amené à être mis définitivement en ligne.

Le but premier de ce projet est de mettre service un réseau privé basé sur la blockchain tout en testant différents services web notamment, ceux proposés par AWS (Amazon web services).

Un projet permettant la mise en relation de différents intervenants à été imaginé. Les différents intervenants sont :

-L'administrateur (Into the web).
-Des gestionnaires (sociétés proposant des services).
-Des apporteurs d'affaires.
-Et les leads (clients).

Seuls l'administrateur, les gestionnaires et les apporteurs d'affaires auront accès au site.

Un apporteur d'affaires poura se créer un compte sur la plateforme Monlead et ajouter un/des Lead(s) au système MonLead.

Le gestionnaire pourra créer des contrats avec chacun des leads présents sur le système MonLead et donnera les différentes caractérisiques du contrat via un formulaire (type, montant, poucentage, …). Le contrat sera ensuite créé dans la blockhain.

Le contrat sera visible par le gestionnaire et l'administrateur. L'apporteur d'affaires ne pourra voir que les contrats qui lui sont liés et ce, lorsque l'adminitrateur les aura validés.

L'apporteur d'affaires, en consultant ses contrats, aura la possibilité de réclamer le payement de la commission qui lui aura été attribuée pour chaque contrat. L'administrateur se chargera d'effectuer le payement et de renseigner le contrat comme payé une fois celui-ci payé.

Tous les revenus et détails relatifs aux contrats seront disponibles sous le tableau des contrats de l'apporteur d'affaires.

Des menus interactifs permettront d'afficher la géolocalisation liée aux contrats via la position géographique du lead et l'envoi d'email aux différents intervenants.

2.2 Objectifs:

Le projet MonLead a deux objectifs.

1) La mise en service de différentes technologies webs sortant du cadre habituel des outils utilisés par l'entreprise IntoTheWeb.

Voici la liste non exhaustive des serivces non conventionnels à l'environement habituel de l'entreprise IntoTheWeb.

-Services AWS :
-Cognito (login utilisateur)
-DynamodDB (base de données noSQL)
-EC2 (Serveurs API Rest et HyperLedger Fabric)

-HyperLedger fabric

2) Mettre en relations des clients potentiels avec des prestataires de services de type prêts hypothécaires, assurances-vie, rénovation de biens immobiliers… Rémunérer les apporteurs d'affaires qui auront apporté un client potentiel au projet MonLead.

2.3 Public cible :

Des personnes intéressées par des services tels que prêts hypothécaires, assurances-vie, rénovation de biens immobiliers,…

MonLead n'est pas destiné aux utilisateurs non enregistrés.

3 EXIGENCES FONCTIONNELLES

3.1 Fonctionnalités :

MonLead est destiné à l'administrateur (IntoTheWeb), aux gestionnaires et aux apporteurs d'affaires.

Voici la liste des différentes actions possibles par chacun des utilisateurs.

-L'adminitrateur pourra:

- Ajouter un gestionnaire au système monLead
- Afficher ses informations personnelles.
- Mettre à jour ses informations personnelles.
- Effectuer une recherche dans la liste des leads.
- Effectuer une recherche dans la liste des gestionnaires.
- Consulter les contrats
- Consulter/modifier les informations des Leads.
- Consulter/modifier les informations des gestionnaires.
- Valider les contrats .
- Indiquer les contrats comme payés.

-Le gestionnaire pourra :

- S'enregistrer dans le système en tant que nouveau gestionnaire.
- Afficher ses informations personnelles.
- Mettre à jour ses informations personnelles.
- Ajouter un listing de projets (Immobilier, prêt hypothécaire,…) .
- Consulter ses contrats.

-L'apporteur d'affaires pourra:

- S'enregistrer dans le système en tant que nouvel apporteur d'affaires/client.
- Afficher ses informations personnelles.
- Mettre à jour ses informations personnelles.
- Inviter un Lead dans le système MonLead.
- Effectuer une recherche dans la liste des leads qu'il aura apportés.
- Consulter ses contrats.
- Consulter ses revenus.
- Consulter ses Leads.
- Réclamer les commissions relatives à chaque contrat.

L'utilisateur non connecté pourra :

- Se connecter.
- S'enregistrer via une interface de connexion fournie par AWS Cognito et aura la possibilité, si il le souhaite, de s'enregistrer via un réseau social.

3.2 Règles métiers :

Les règles de métiers du projet MonLead sont les suivantes :

- **La création d'un compte de gestionnaire doit au préalable avoir été initié par l'administrateur.**
- **Un mot de passe doit contenir au moins 6 caractères .**
- **Le projet MonLead ne sera pas disponible aux utilisateurs non connectés.**

3.3 Interface :

L'interface doit répondre aux exigences suivantes:

- **Le projet MonLead doit avoir une interface simple et non surchargée.**
- **Le projet MonLead ne sera pas disponible aux utilisateurs non connectés.**
- **Les différents niveaux d'accès auront des interfaces similaires.**

4 EXIGENCES NON FONCTIONNELLES

4.1 Conception :

- **Le maintient d'URL claires et concises.**
- **Un design adatpé à différents médias (PCs, Tablettes, Smartphones)**

4.2 Acteurs :

Apporteur d'affaires:

L'apporteur d'affaires est l'utilisateur qui apporte un Lead (client potentiel) au projet.

Gestionnaire:

Le gestionnaire est l'utilisateur qui représente une société proposant des services tels que prêts hypothécaires, assurances-vie, rénovation de biens immobiliers, ...

Administrateur:

L'administrateur est représenté par l'entreprise IntoTheWeb. Il sera responsable de la gestion des contrats.

Utilisateur non inscrit:

L'utilisateur non inscrit/connecté n'a pas accès aux fonctionnalités du projet MonLead.

5 CAS D'UTILISATION

5.1 Enregistrement :

L'enregistrement comporte deux cas possibles.

1) Un utilisateur anonyme s'enregistre sans avoir été initié par l'administrateur. Son profil sera un profil **apporteur d'affaires**.

   Une fois l'utilisateur enregistré via le pannel d'enregistrement AWS, il sera invité à compléter ses informations personelles.

2) Un **gestionnaire** souhaite s'enregistrer. Dans ce cas, son profil doit avoir été initié au préalable par l'adminitrateur via l'option « Ajouter un gestionnaire » depuis le menu du sidepanel.

   Une fois l'utilisateur enregistré via le pannel d'enregistrement AWS, il sera invité à compléter ses informations personelles.

5.2 Connexion :

L'utilisateur non connecté sera automatiquement redirigé vers la page de d'accueil. Via cette page, il aura la possibilité de se connecter ou de s'enregistrer.

Tous les types d'utilisateurs seront redirigés vers la page affichant leur(s) contrat(s).

5.3 Changement de langue :

Sur toutes les pages, l'utilisateur aura la possibilité de changer la langue d'affichage via le dropdown de choix de langue présent dans le header.

5.4 Ajout de leads :

L'apporteur d'affaires qui souhaite ajouter un Lead au projet pourra accéder au menu de création via l'option « Ajouter un lead » depuis le menu du sidepanel.

Il sera redirigé vers un formulaire où il sera invité à remplir les informations du Lead.

L'utilisateur sera ensuite redirigé vers la liste des leads.

5.5 Ajout de gestionnaire :

Un nouveau gestionnaire devant être soumis à des vérifications par l'administrateur, son profil devra être initié par celui-ci. Un utilisateur ne sera pré-enregistré qu'avec les autorisations nécessaires.

Une fois le compte initié par l'administrateur, le gestionnaire pourra s'enregistrer en suivant la démarche habituelle d'enregistrement. Son compte sera automatiquement défini en tant que gestionnaire. **IMPORTANT** : Le gestionnaire doit impérativement s'enregistrer avec la même adresse email avec laquelle l'administrateur aura initié son compte.

Une fois le gestionnaire ajouté, une newsletter sera envoyée à tous les apporteurs d'affaires pour les en informer.

5.6 Ajout de nouvelles catégories :

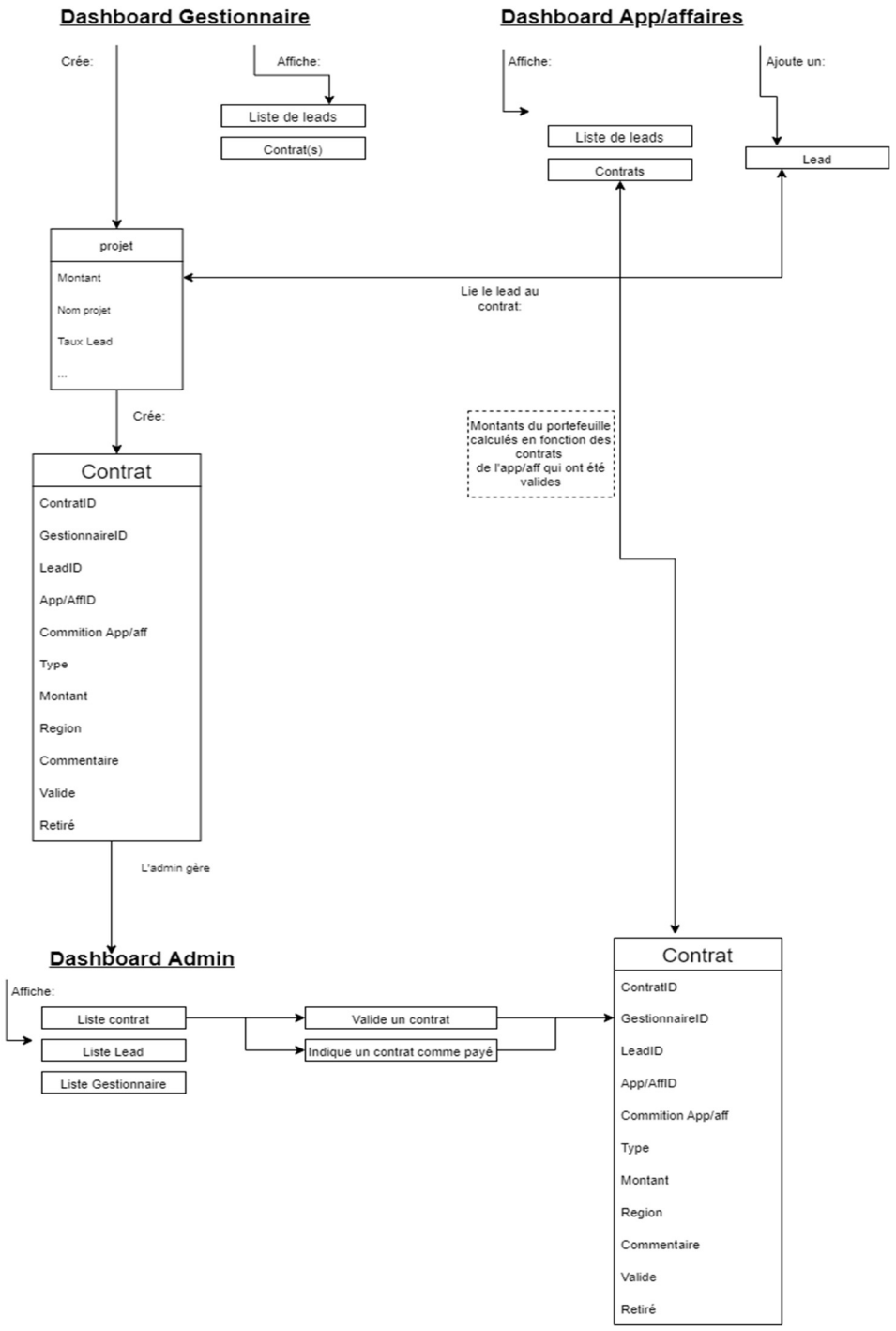
Les catégories seront ajoutées en backend directement par l'administrateur. Les types de contrats seront plus que probablement les mêmes. Il n'y aura donc pas de possibilités d'en ajouter via l'interface MonLead.

5.7 Création d'un contrat :

Un contrat est créé par un gestionnaire. Le gestionnaire sera alors redirigé vers un formulaire qui lui permettra de sélectionner les intervenants et les caractéristiques du contrat.

Une fois le contrat créé, un mail comprenant les informations relatives au contrat sera envoyé aux différents intervenants du contrat.

Flowchart décrivant le cycle de création d'un contrat.



5.8 Consulter la liste des contrats :

L'affichage des contrats est légèrement différent en fonction du type d'utilisateur qui les consulte. Tous les utilisateurs connectés pourront voir leurs contrats.

Une pagination sera disponible sous la liste des contrats. A côté de celle-ci, un indicateur affichera le nombre de résultats retournés par page ainsi que le le total des articles disponibles.

L'administrateur consulte les contrats. Tous les contrats seront retournés. Aucun filtre ne sera appliqué.

Il aura la possibilité:

-D'envoyer un email aux différents intervenants de chaque contrat.
-De valider les contrats non valides via le bouton « Validé ».
-D'indiquer un contrat comme payé via le bouton « Payé ».
-D'afficher la localisation du lead sur une carte.

Les champs du tableau disponibles sont :

-Le nom du gestionnaire lié au contrat.
-Le nom du lead lié au contrat.
-Le nom de l'apporteur lié au contrat.
-Le type de catégorie liée au contat.
-La valeur totale du contrat.
-La commission de l'apporteur d'affaires.
-La ville du lead lié au contrat.
-La validité du contrat.
-L'indication que le contrat a été réclamé.
-L'indication que le contrat a été payé (avec bouton pour indiquer le contrat comme payé).

L'apporteur d'affaires consulte les contrats. Les contrats retournés seront les contrats qui le concernent et qui auront été validés par l'administrateur.

Pour l'apporteur d'affaires uniquement, un tableau sera représenté sous la liste des contrats. Ce tableau contiendra un résumé des revenus perçus, la valeur totale de ses contrats , les revenus en attente de payement ainsi que la valeur totale de toutes les commissions, y compris les commissions non créditées.

Il aura la possibilité :

-D'envoyer un email au gestionnaire du contrat.
-De réclamer le payement d'un contrat non réclamé via le bouton « Réclamé ».
-D'afficher la localisation du lead sur une carte.

Les champs du tableau disponibles sont :

-Le nom du gestionnaire lié au contrat.
-Le nom du lead lié au contrat.
-Le nom de l'apporteur lié au contrat.
-Le type de catégorie liée au contat.
-La valeur totale du contrat.
-La commission de l'apporteur d'affaires.
-La ville du lead lié au contrat.
-L'indication qu'un contrat a été réclamé. (avec bouton pour indiquer le contrat comme réclamé).

-L'indication qu'un contrat a été payé.

Le gestionnaire consulte les contrats. Les contrats retournés seront les contrats qui le concernent, y compris les contrats non validés par l'administrateur.

Il aura la possibilité:

-D'envoyer un email au lead lié au contrat.
-D'envoyer un email à l'apporteur d'affaires lié au contrat.
-D'afficher la localisation du lead sur une carte.
-De créer un nouveau contrat via le bouton « Créer un contrat ».

Les champs du tableau disponibles sont :

-Le nom du gestionnaire lié au contrat.
-Le nom du lead lié au contrat.
-Le nom de l'apporteur lié au contrat.
-Le type de catégorie liée au contat.
-La valeur totale du contrat.
-La commission de l'apporteur d'affaires.
-La ville du lead lié au contrat.
-La validité du contrat.
-L'indication qu'un contrat a été réclamé.
-L'indication qu'un contrat a été payé.

5.9 Consulter la liste des leads :

L'affichage des leads est légèrement différent en fonction du type d'utilisateur qui consulte les leads. Seuls l'administrateur et l'apporteur d'affaires peuvent voir la liste des leads.

Une pagination sera disponible sous la liste des leads. A côté de celle-ci, un indicateur affichera le nombre de résultats retournés par page ainsi que le le total des articles disponibles.

L'administrateur consulte les leads. Tous les leads seront retournés. Aucun filtre ne sera appliqué.

Il aura la possibilité:

-De modifier les informations des leads.
-D'effectuer une recherche parmi les leads via un formulaire de recherche.

L'apporteur d'affaires consulte les leads. Les leads retournés seront uniquement ceux qu'il aura apportés à MonLead.

Il aura la possibilité:

-De modifier les informations des leads qu'il aura apportés.
-D'effectuer une recherche parmi les leads via un formulaire de recherche.
-D'ajouter un lead via le bouton « Ajouter un lead ».

5.10 Affichage profil :

Les trois types d'utilisateurs connectés auront accès à l'affichage du profil via le bouton « voir profil » de la sidebar.

Le profil est affiché en deux champs :

1) La section où l'utilisateur pourra, consulter son avatar si il en a publié un. Dans le cas contraire, un avatar par défaut sera affiché.
   La modification de l'avatar se fera directement via la page d'affichage du profil.

2) La partie « détails » qui listera les différents champs du type d'utilisateur connecté.

5.11. Modification profil :

Les trois types d'utilisateurs connectés auront accès à la modification du profil via le bouton « modifier profil » de la sidebar.

Ils seront redirigés vers une la page d'édition de profil sur laquelle se trouvera un formulaire pré-rempli comprenant les informations existantes du profil utilisateur connecté. Une fois le profil enregistré, l'utilisateur sera redirigé vers l'affichage du profil.

6 DESCRITPTION DE LA BASE DE DONNEES

6.1 Descriptif général du système de base de données :

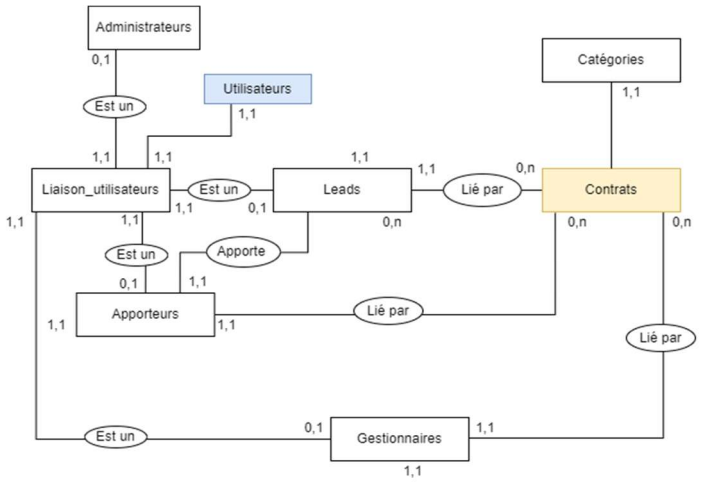
Le système de base de données est scindé en trois parties bien distinctes. L'objectif est de travailler et de tester le fonctionnement d'une blockchain privée faisant office de base de données. Cette base de données devant être la plus simple possible, un minimum d'infomations y seront présentes.

Les données de login des utilisateurs seront présentes via le service Cognito de AWS.

Toutes les autres données seront quant à elles enregistrées sur la base de données noSQL DynamoDB de AWS.

Remarque : Les deux types de bases de données ne communiquent pas ensemble via des relations. Les identifiants des différents intervenants seront donc gérés via le backend en fonction des données de la base de données DynamoDB.

Schéma conceptuel et physique de la base de données.



Utilisateurs	
PK	utilisateur:STRING
	email :STRING
	account_status:STRING
	email_verified:STRING
	created_at: DATETIME
	updated_at: DATETIME

Administrateurs	
PK	ID:STRING
	firstname :STRING
	lastname :STRING
	town : STRING
	street : STRING
	street_nbr : INTEGER
	created_at :DATETIME
	updated_at :DATETIME
FK	liaison_utilisateurID:INTEGER

Gestionnaires	
PK	ID :STRING
	company_name :STRING
	town : STRING
	street_nbr : INTEGER
	street : STRING
	iban :STRING
	bic :STRING
	created_at :DATETIME
	updated_at :DATETIME
FK	liaison_utilisateurID:INTEGER

Categories	
PK	ID:STRING
	name:STRING
	created_at :DATETIME
	updated_at :DATETIME

Contrats	
PK	Key :STRING
	gestionnaireID:STRING
	leadID:STRING
	App_AffID:STRING
	comm_app_aff: STRING
	type: STRING
	montant: STRING
	region: STRING
	commentaire: STRING
	valide: STRING
	retire : STRING
	payer: STRING

Leads	
PK	ID :STRING
	firstname :STRING
	lastname :STRING
	email :STRING
	town : STRING
	street : STRING
	street_nbr : INTEGER
	iban :STRING
	bic :STRING
	created_at :DATETIME
	updated_at :DATETIME

Apporteurs	
PK	ID :STRING
	firstname :STRING
	lastname :STRING
	town : STRING
	street_nbr : INTEGER
	street : STRING
	iban :STRING
	bic :STRING
	created_at :DATETIME
	updated_at :DATETIME
FK	liaison_utilisateurID:INTEGER

Liaison_utilisateurs	
PK	ID :STRING
	email :STRING
	created_at: DATETIME
	updated_at: DATETIME
FK	gestionnaireID: STRING
FK	administrateursID: STRING
FK	apporteurID: STRING

6.2 Détails de table « blockchain » via HyperLedger Fabric :

Comme on peut le constater sur le schéma de table de base de données ci-dessous, le seul type de champ disponible est de type « string ». Par conséquent, toutes les conversions de types devront se faire en back-end ou via les views lors de l'affichage des données du contrat.

Key : ID du contrat. Ce champ doit être unique à tous les contrats.

Remarque : Ce champ n'est pas réellement une clef primaire mais il se comprte comme tel.

GestionnaireID : ID du gestionnaire concerné par un contrat.

LeadID : ID du lead concerné par un contrat.

App_AffID : Apporteur d'affaires concerné par un contrat.

Comm_app_aff : Taux en % de la commission de l'apporteur d'affaires concerné par un contrat.

Type : Type catégorie d'un contrat.

Montant : Montant total d'un contrat.

Region : Ville du lead concerné par un contrat. Ce champ est rempli automatiquement à la création du contrat.

Commentaire : Commentaires facultatifs. (L'affichage des commentaires n'est actif dans aucune des vues)

Valide : Validité ou non du contrat.

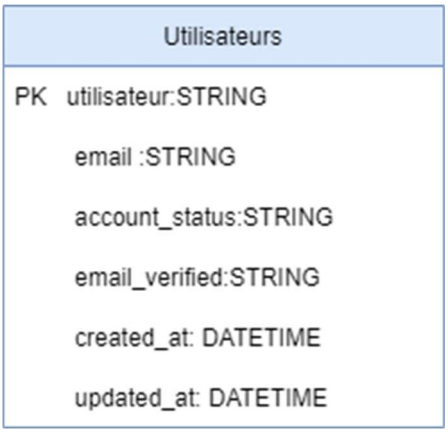
Retire : Commission d'un contrat réclamée ou non.

Payer : Indique si un contrat a été payé ou non par l'administrateur.

6.3 Détails de table utilisateurs via Cognito :

Cette table fait partie du service « Cognito » fourni par les services web d'Amazon (AWS).

C'est ce service qui sera responsable de gérer l'enregistrement et la connexion des utilisateurs.



Utilisateur : ID de l'utilisateur. Ce champ est généré automatiquement par Cognito et doit être unique.

Remarque : Ce champ n'est pas réellement une clef primaire mais il se comporte comme tel.

Email : Email de l'utilisateur.

Account_status : Indique si le compte a été confirmé ou si l'utilisateur a été vérifié depuis un réseau social.

Email_verified : Indique que l'email associé au compte a été vérifié ou non.

Updated_at : Date de la dernière modification apportée à l'utilisateur.

Created_at : Date de création de l'utilisateur.

6.4 Détails de la base de données DynamoDB :

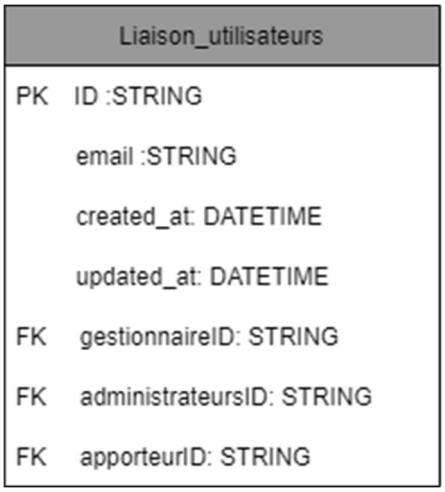
Cette base de données constitue la base de données principale du projet MonLead.

Une base de données dynamoDB étant une base de données noSQL, les tables ne comportent pas nativement de relations. La gestion de relations est gérée par l'ORM « Dynamoid ». Pour dynamoDB sur Ruby On Rails, c'est lui qui sera responsable de la création et de la gestion des clefs étrangères dans les différentes entités.

6.4.1. Liaison_utilisateurs

Cette table a été créée pour faire la liaison entre la base de données « Cognito » et un type d'utilisateur (administrateur, gestionnaire ou apporteur d'affaires).

La table utilisateur de « Cognito » offrant peu d'options par rapport aux champs qui sont créés/utilisés, il était important de faire une table liaison avec la base de données « dynamoDB ».



ID : ID de l'utilisateur. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Email : Email de l'utilisateur.

GestionnaireID : ID du gestionnaire si l'utilisateur correspondant est un gestionnaire.

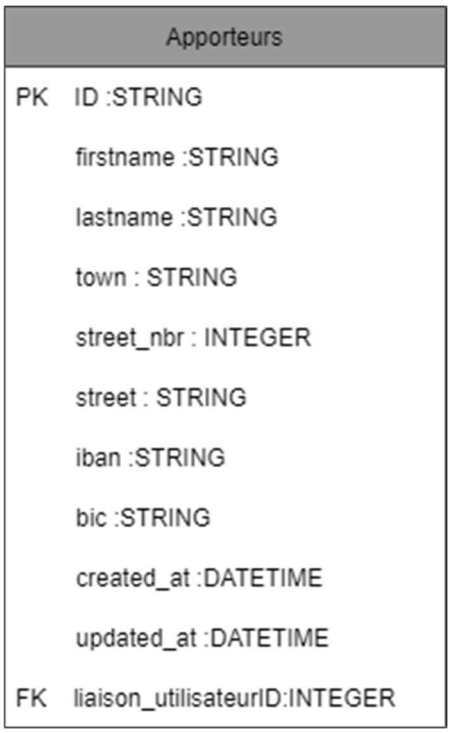
AdministrateurID : ID de l'administrateur si l'utilisateur correspondant est un administrateur.

ApporteurID : ID de l'apporteur d'affaires si l'utilisateur correspondant est un apporteur d'affaires.

Updated_at : Date de la dernière modification apportée à l'utilisateur.

Created_at : Date de création de l'utilisateur.

6.4.2. Apporteurs



ID : ID de l'apporteur d'affaires. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Firstname : Prénom de l'apporteur d'affaires.

Lastname : Nom de l'apporteur d'affaires.

Town: Ville de l'apporteur d'affaires.

Street_nbr : Numéro de rue de l'apporteur d'affaires.

Street : Rue de l'apporteur d'affaires.

Iban : Compte IBAN de l'apporteur d'affaires.

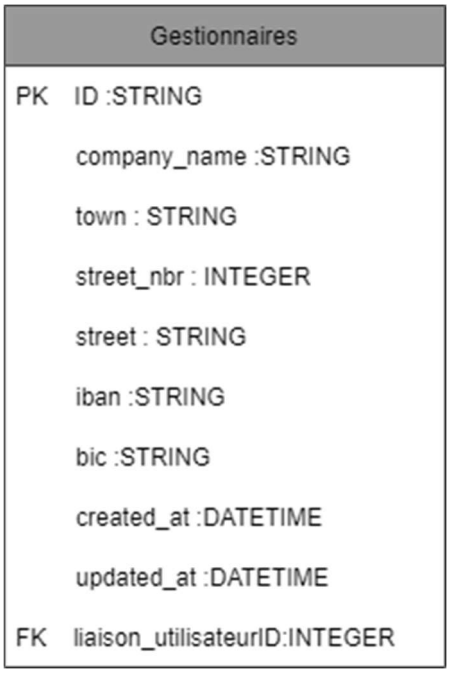
Bic : Code bic de l'apporteur d'affaires.

Updated_at : Date de la dernière modification apportée à l'apporteur d'affaires.

Created_at : Date de création de l'apporteur d'affaires.

UtilisateurID : Clef étrangère liant l'apporteur d'affaires à l'utilisateur via la table liaison_utilisateurs.

6.4.3. Gestionnaires



ID : ID du gestionnaire. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Company_name : Nom de l'entreprise.

Town: Ville du gestionnaire.

Street_nbr : Numéro de rue du gestionnaire.

Street : Rue du gestionnaire.

Iban : Compte IBAN du gestionnaire.

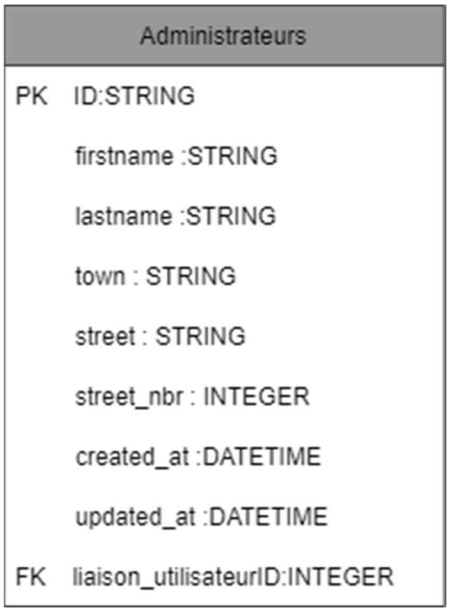
Bic : Code bic du gestionnaire.

Updated_at : Date de la dernière modification apportée au gestionnaire.

Created_at : Date de création du gestionnaire.

UtilisateurID : Clef étrangère liant le gestionnaire à l'utilisateur via la table liaison_utilisateurs.

6.4.4. Administrateurs



ID : ID de l'administrateur. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Firstname : Prénom de l'administrateur.

Lastname : Nom de l'administrateur.

Town: Ville de l'administrateur.

Street_nbr : Numéro de rue de l'administrateur.

Street : Rue de l'administrateur.

Iban : Compte IBAN de l'administrateur.

Bic : Code bic de l'administrateur.

Updated_at : Date de la dernière modification apportée à l'administrateur.

Created_at : Date de création de l'administrateur.

UtilisateurID : Clef étrangère liant l'administrateur à l'utilisateur via la table liaison_utilisateurs.

6.4.5. Leads

Un lead n'ayant pas accès à MonLead, il ne devra pas être lié à un utilisateur. Par conséquent son email ne sera pas enregistré dans la table liaison_utilisateurs mais bien dans sa propre table.

ID : ID du lead. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Firstname : Prénom du lead.

Lastname : Nom du lead.

Email : Adresse email du lead.

Town: Ville du lead.

Street_nbr : Numéro de rue du lead.

Street : Rue du lead.

Iban : Compte IBAN du lead.

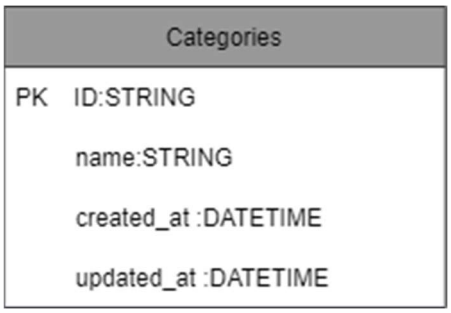
Bic : Code bic du lead.

Updated_at : Indique la date de la dernière modification apportée au lead.

Created_at : Indique la date de création du lead.

UtilisateurID : Clef étrangère liant le lead à l'utilisateur via la table liaison_utilisateurs.

6.4.6. Catégories



ID : ID de la catégorie. Ce champ est un UUID unique qui est généré automatiquement par dynamoDB.

Name : Nom de la catégorie.

Updated_at : Date de la dernière modification apportée au à la catégorie.

Created_at : Date de création de la catégorie.

7 PRESENTATION DES PROBLEMES ET SOLUTIONS ENVISAGEES

7.1 Mise en service de la blockchain privée HyperLedger Fabric.

La première et plus grosse difficulté a été de mettre en service la blockchain privée HyperLedger Fabric. Bien que très fournie, la documentation ne rendait pas la mise en service très simple.

Solution n°1 :

La première piste envisagée a été de faire tourner tout un réseau sur mon PC personnel. La documentation était plutôt orientée vers cette solution. C'est donc pour cette raison que j'ai d'abord choisi cette configuration.

Deux soucis se sont alors présentés. Premièrement, la documentation ne m'éclairait pas vraiment sur la marche à suivre pour communiquer entre le backend et la blockchain privée.

Deuxièmement, le réseau étant hébergé sur ma machine personnelle, il était obligatoire de garder toujours un poste de travail en ligne.

Pour ces raisons, j'ai rapidement abandonné cette solution.

Solution n°2 :

Après discusson avec mon maître de stage, il m'a été conseillé de tester le service web d'AWS « Amazon Managed Blockchain ». Cette solution semblait plutôt intéressante car elle permettait d'automatiser une partie du setup du réseau. De plus, ce service étant un service AWS, il nous avait semblé que l'intégration avec les autres services AWS utilisés auraient été plus simple. Malheureusement, ça n'a pas été le cas.

Le début de la mise en ligne du réseau s'est plutôt bien passée. Je me suis ensuite penché sur le développement des « chaincodes » ou «smart-contrats » qui permettent d'écrire les méthodes qui manipulent et communiquent avec les données de la blockchain. Pour écrire ces smart-contrats, trois langages sont disponibles. Il s'agit de GO, Java et Javascript. Ne connaissant ni le langage GO, ni Java, je me suis tourné vers le Javascript. La documentation utilisait le langage GO mais la procédure de déploiement était relativement similaire.

A la fin de cette documentation, il fallait exécuter un « npm install » pour installer les dépendances nécessaires à l'exécution du smartcontrat. Malheureusement, pour des raisons qui m'échappent, avec la dernière version de hyper ledger fabric il est impossible d'exécuter cette commande sur l'hôte hébergé par le service Amazon Managed Blockchain. Une solution me permettant de contourner le souci a été trouvée mais cela demandait un long travail manuel pour chaque recompilation du package du smart-contract. Je me suis vite rendu compte que cela allait grandement retarder mon développement.

J'ai trouvé la documentation Amazon Managed Blockchain particulièrement trompeuse car elle stipulait bien que les trois langages était utilisables mais le détail concernant l'impossiblité d'exécuter la commande « npm install » n'y était pas présent.

J'ai donc également abandonné cette solution.

Solution n°3 :

La troisième solution à été de suivre une configuration sur quatre hôtes différents communiquant entre eux via le « swarm mode » de docker. Ce guide sur le site medium.com était très clair et détailé. Malheureusement cette configuration n'a pas été concluante. De plus, les quatre hôtes étant des instances EC2 distinctes, cela augmentait grandement les coûts de mise en service de ce réseau.

J'ai également abandonné cette solution.

Solution n°4 :

La quatrième solution a été de suivre une configuration sur un seul hôte pour le réseau HyperLedger Fabric et de communiquer avec lui depuis le backend via un deuxième hôte sur lequel une API REST a été mise service.

Le backend communique donc avec les informations du réseau HyperLedger Fabric via différents appels API

Ce guide était du même auteur que celui de la solution n°3. Et il s'est avéré concluant et fonctionnel.

7.2 Utilisation de dynamoDB avec Ruby On Rails.

L'utilisation de la base de données dynamoDB n'a pas été simple. Il a fallu trouver des solutions à différents problèmes. Ces solutions ont été implémentées pour différentes raisons. Soit pour le confort de l'utilisateur ou simplement pour permettre de résoudre certaines contraintes techniques.

Ruby On Rails embarque l'ORM Active Record. Celui-ci est très bien documenté et contient beaucoup de ressources utilisateurs. Malheureusement, l'utilisation de DynamoDB comme base de données m'a obligé de me tourner vers un ORM conçu pour DynamodDB. Cet ORM est appelé Dynamoid.

Le souci de cet ORM est que la documentation n'était pas toujours très complète et les ressources utilisateurs étaient très souvent dépassées et obsolètes. De plus, cet ORM ne permettait pas d'effectuer toutes les tâches que Active Record proposait pour des bases de données traditionelles. Aux pages suivantes, se trouve une liste des soucis rencontrés lors du développement avec DynamoDB ainsi que les solutions apportées.

Problème n°1

La difficulté d'attacher un fichier à une entité, par exemple un fichier image.

Ruby On Rails comporte un outil nommé Active Storage dépendant de l'ORM Active Record pour attacher différents fichiers à une entité. Ces fichiers seront sauvegardés sous forme de string sur le champ qui y correspond. Pour un utilisateur, il pourrait s'agir, par exemple, d'un champ avatar. Le fichier quant à lui aurait été envoyé sur un cloud de notre choix ou sur l'espace disque de notre propre serveur.

Malheureusement l'ORM dynamoid ne permet pas de faire une telle action. Un bundle non maintenu « dynamoid-paperclip » permet de le faire mais il n'a pas été maintenu. Après de longues recherches et essais infructueux , j'ai été contraint de chercher une autre solution pour attacher des fichiers à mes entités. Dans mon cas, il s'agit de l'ajout d'un avatar pour chaque utilisateur.

Solution :

La solution apportée aura été de gérer l'envoi d'avatar manuellement sur l'espace disque du serveur. Un formulaire de type « file » a été créé. Celui-ci n'accepte que les fichiers .png et .jpg. Le fichier en question est ensuite envoyé et nommé avec le nom de l'utilisateur connecté. Par exemple « administrateur01@admin.test.png ».

Le backend se chargera alors de chercher si un fichier administrateur01@admin.test.png ou administrateur01@admin.test.jpg existe. Si aucun de ces deux fichiers n'existe, un avatar par défaut sera affiché.

Problème n°2

Recherche combinée.

L'ROM dynamoid ne permettant pas d'utiliser plusieurs closes « Where » à la chaîne, il est impossible d'effectuer des recherches complexes. Par exemple, retourner tous les utilisateurs donc le prénom est « Christophe » et qui résident dans la ville de « Ticheville ».

Solution :

Pour effectuer des recherches combinées, j'ai donc filtré tous les utilisateurs dont le prénom est « Christophe » et ensuite via le backend, je n'ai gardé que les utilsateurs se trouvant dans la ville de « Ticheville ».

Cette solution n'est pas vraiment performante pour des applications à grande échelle mais elle me semblait raisonnablement optimale pour mon cas d'utilisation.

Problème n°3

La pagination.

La pagination de l'ORM dynamoid existe mais n'est malheureusment pas très pratique pour l'utilisateur. En effet, elle ne permet pas d'effectuer des recherches par page mais elle permet uniquement changer de pages via le bouton suivant et précédent. Si un utilisateur appuie 10 fois sur le bouton suivant, il lui sera impossible de revenir sur la page n°2 en un seul clic. Il sera contraint de cliquer plusieurs fois sur le bouton précédent jusqu'à ce qu'il arrive à nouveau sur la page qu'il souhaite.

Cette solution est la plus performante d'un point de vue ressources mais n'est clairement pas pratique pour l'utilisateur. J'ai donc préféré ne pas l'utiliser.

Solution :

Pour la pagination, je me suis tourné vers une pagination manuelle.

Le backend se chargera de récupérer tous les éléments de la base de données. Les pages seront envoyées au frontend grâce aux paramètres _GET du numéro de page ainsi que le nombre d'éléments par page définis dans le backend.

Cette solution offre une utilisation beaucoup plus agréable mais a l'inconvénient d'être très gourmande en ressources car tous les éléments de la base de données sont retournés à chaque requête. Je me suis néanmoins tourné vers cette solution car le nombre de données dans chacune des tables n'est pas très important.

Problème n°4

Mise en service du système de login.

Ruby On Rails ayant son propre système de gestion d'utilisateurs nommé devise et devise étant dépendant de Active Record, il m'était impossible d'utiliser cette solution. A nouveau, il existe bien un bundle permettant l'utilisation de devise avec l'ORM dynamoid mais ce bundle était fort dépassé et obsolète.

Solution :

Après plusieurs essais et de longues recherches, je me suis tourné vers la gestion utilisateur via le service AWS Cognito et son interface « hosted UI » qui permet d'être redirigé vers une page hébergée et gerée par AWS. C'est elle qui va être responsable de la gestion de l'enregistrement et de la vérification du login utilisateur. Cette solution a été compliquée à mettre en place car la gestion et la vérification des JSON Web Tokens qui sont renvoyés par AWS Cognito doivent se faire manuellement. Heureusement j'ai été aidé par un repository de l'utilisateur github mheffner. Après avoir effectué plusieurs tests avec son projet j'ai été capable de transposer sa solution d'authentification sur mon propre projet.

7.3 Gestion utilisateurs via Cognito.

Communication en backend avec les utilisateurs cognito.

Comme expliqué dans la solution au problème de la page précédente, je me suis tourné vers une gestion d'utilisateurs externe gérée par le service web AWS d'Amazon Cognito. Celui-ci fait une bonne partie du travail de gestion d'utilisateurs mais n'offre pas une grande flexibilité pour ce qui est des recherches.

Par exemple, il est impossible d'effectuer des recherches sur tous les champs personnalisés, pour un champ « ville » il n'est pas possible de chercher tous les utilisateurs résidents dans la ville de « Ticheville ». Je me suis tout de suite rendu compte que cela allait poser dès problèmes pour la suite du développement du projet.

Solution :

J'ai donc décidé de créer une table de liaison « liaison_utilisateurs ». Cette table contient l'email d'un utilisateur ainsi que la clef étrangère qui correspond au type d'utilisateur (gestionnaire, administrateur ou apporteur d'affaires). Cette table faisant partie de la base de données DynamoDB, il m'était alors possible d'effectuer les recherches nécessaires sur chacun des utilisateurs.

8 CONCLUSION

Beaucoup des technologies étaient totalement nouvelles et inconnues pour moi. Cela a demandé un gros travail de recherches et d'expérimentations.

Voici une liste des technologies nouvelles avec lesquelles j'ai eu l'occasion de travailler :

- Le service AWS Dyanmodb (base de données)
- Le service AWS Cognito (gestion utilisateurs)
- Le service AWS EC2 (hébergent d'HyperLedger Fabric et du serveur api REST)
- Le service Aws Amazon Managed Blockchain (Test de développement d'une blockchain privée gerée par AWS)
- Ruby (Langage de programmation)
- Ruby On Rails(Framework backend)
- Express (API REST)
- Docker (La gestion de l'instance HyperLedger Fabric)

La combinaison des outils et technologies imposée a, dans un premier temps, été une contrainte non négligeable à respecter. Cette combinaison de technologies n'est pas fort documentée par les utilisateurs. Il était donc difficile de trouver des solutions pertinentes et non obsolètes aux problèmes auxquels j'ai été confrontés.

Cependant, une fois les solutions trouvées et le fonctionnement du framework Ruby On Rails maitrisé, le développement de ce projet aura été très agréable et gratifiant. J'ai le sentiment d'avoir énormément progressé durant les quelques mois de développement de ce projet.

9 REMERCIEMENTS

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes auxquelles je voudrais témoigner toute ma reconnaissance.

Je souhaite remercier :

Mes professeurs de l'Ecole de Promotion Sociale Saint-Laurent à Liège qui m'ont fourni les outils nécessaires à l'apprentissage et le création de projets tels que celui-ci. Durant ces 3 années, ils ont été motivés et motivants.

L'entreprise IntoTheWeb qui m'a fourni un projet très intéressant et riche en apprentissages. Il se prêtait particulièrement bien à mon mémoire.

Romain Mathieu qui m'a aidé à solutionner un problème lors du déploiement du projet.

Tous mes camarades de classes avec qui nous avons eu une entente chaleureuse. Les relations ont été riches tant sur le plan humain que sur celui des apprentissages.

Béatrice Cerfontaine qui a relu et la corrigé ce mémoire.

BIBLIOGRAPHIE

hyperledger-fabric. (s. d.). *Using the Fabric test network — hyperledger-fabricdocs main documentation*. Https://Hyperledger-Fabric.Readthedocs.Io/En/Latest/Test_network.Html.

K.C. Tam, K. (2021, 15 décembre). *Rework : An Implementation of API Server for Hyperledger Fabric Network (Fabric v2.2)*. Medium. https://kctheservant.medium.com/rework-an-implementation-of-api-server-for-hyperledger-fabric-network-fabric-v2-2-a747884ce3dc

RubyOnRails. (s. d.). *Getting Started with Rails*. Ruby on Rails Guides. https://guides.rubyonrails.org/getting_started.html

Dynamoid. (s. d.). *GitHub - Dynamoid/dynamoid: Ruby ORM for Amazon's DynamoDB.* GitHub. https://github.com/Dynamoid/dynamoid

Mheffner, M. (s. d.). *GitHub - mheffner/rails-cognito-example: Example Rails application demonstrating server-side Cognito User Pools*. GitHub. https://github.com/mheffner/rails-cognito-example

Ruby on Rails Tutorial => Set locale through requests. (s. d.). Https://Riptutorial.Com/Ruby-on-Rails/Example/9336/Set-Locale-through-Requests. https://riptutorial.com/ruby-on-rails/example/9336/set-locale-through-requests

Amazon. (s. d.). *Announcing Amazon Managed Blockchain*. Amazon Web Services, Inc. https://aws.amazon.com/managed-blockchain/