

PROJET GENIE LOGICIEL (SINFO015) :

Rapport de modélisation

Auteurs:

Adrien Fievet
Claire D'Haene
Julien Ladeuze
Maxime Dupuis

2022-2023

Titulaire

Tom MENS
Enseignants
Sébastien BONTE
Jeremy DUBRULLE
Pierre HAUWEELE

Equipe 7

Adrien (220625) 4
Claire (220323) 1
Julien (220101) 10
Maxime (212107) 7



Contents

1 Use cases diagrams	6
1.1 Points communs	6
1.2 Client	7
1.3 Fournisseur	8
2 Interaction overview diagrams	9
2.1 Introduction	9
2.2 Points communs	10
2.3 Accès à l'application pour les clients	11
2.4 Accès à l'application pour les fournisseurs	14
3 Entity relationship diagram	17
3.1 Préambule	17
3.2 Utilisateur	17
3.3 Langue	18
3.4 Client	18
3.5 Portefeuille	18
3.6 Contrat du portefeuille	18
3.7 fournisseur	18
3.8 Proposition	19
3.9 Contrat du fournisseur	19
3.10 Contrat	19
3.11 Compteur	20
3.12 Consommation	20
3.13 Notification	20
3.14 Schéma	21
4 Class diagram	22
4.1 Introduction	22
4.2 Le package API	23
4.3 Le package database	24
4.4 Le package dataObject	25
4.5 La classe App	26
5 Sequence diagrams	28
5.1 Introduction	28
5.2 Common	29
5.2.1 Gérer les données	30
5.2.2 Gérer les langues	31
5.2.3 Modifier son mot de passe	32
5.2.4 Répondre aux notifications	33
5.2.5 S'authentifier	34
5.2.6 Visualisation des données de consommations	36
5.2.7 Voir les contrats	37

5.2.8	Voir notifications	38
5.3	Client	39
5.3.1	Contrats	39
5.3.2	Fournisseurs et contrats relatifs	40
5.3.3	Portefeuilles	42
5.4	Provider	44
5.4.1	Gestion des clients	44
5.4.2	Gestion des propositions	46
5.4.3	Gestion de la consommation	48
6	Design REST Api	49
6.1	Introduction	49
6.2	Gestion d'un utilisateur	49
6.3	Ajout de données	49
6.4	Affichage de données	50
6.5	Modification/Suppression de données	50
6.6	Conclusion	50
6.7	Schéma	51
7	Interface	52
7.1	Introduction	52
7.2	Système de logs	53
7.3	Points communs	54
7.4	Interface client	55
7.5	Interface fournisseur	60
8	Introduction-extension	64
9	Extension 1.6.1 : Gestion d'utilisateurs par Claire D'Haene	65
10	Use case diagram	66
10.1	Introduction	66
10.2	Client	66
10.3	Diagramme	67
11	Interaction overview diagram	68
11.1	Introduction	68
11.2	Accès à l'application pour les clients	69
11.3	Diagramme	70
12	Entity relationship diagram	71
12.1	Introduction	71
12.2	Utilisateur	71
12.3	Portefeuille invité	72
12.4	Contrat du portefeuille	72
12.5	Diagramme	73

13 Class diagram	74
13.1 Introduction	74
13.2 Le package api	74
13.3 Le package DataBase	75
13.4 Le package dataObject	75
13.5 Diagramme	76
14 Sequence diagram	77
14.1 Introduction	77
14.2 Client	78
14.2.1 Gestion des portefeuilles invités	78
14.2.2 Gestion des utilisateurs invités	80
15 API REST	82
15.1 Introduction	82
15.2 Les utilisateurs invités	82
15.3 Les portefeuilles invités	82
15.4 Diagramme	83
16 Interface	84
16.1 Introduction	84
16.2 L'interface client	85
16.3 Image de l'interface	88
17 Extension 1.6.4: Analyse statistique de la consommation énergétique par Adrien Fiévet	89
18 Use cases diagrams	90
19 Interaction overview diagram	91
20 Schéma de la base de données	92
21 Class diagram	93
22 Séquence diagram	95
23 Api rest	97
24 Schéma de l'interface	98
25 Extension 1.6.10 : Application android par Julien Ladeuze	100
26 Aspect pratique	100
27 Aspect technologique	100
28 Interface	100

29 Ressources utilisées	101
30 Schéma	101
31 Extension 1.6.7 : Facturation et paiement d'acomptes par Maxime Dupuis	102
32 Use cases diagram	103
32.1 Description	103
32.2 Schéma	104
33 Interaction overview diagram	105
33.1 Description	105
33.2 Schéma	106
34 Class diagram	107
34.1 Description	107
34.2 Les variables	107
34.3 Les méthodes	108
34.4 Schéma	109
35 Entity relationship diagram	110
35.1 Description	110
35.2 Schéma	111
36 Sequence diagram	112
36.1 Description	112
36.2 Schéma	113
37 Design API Rest	114
37.1 Description	114
37.2 Schéma	115
38 Interface	116
38.1 Description	116
38.2 Page d'accueil	116
38.3 Liste des factures	117
38.4 Facture	117
38.5 Paiement	118
38.6 Modification de la proposition	118
38.7 Changement de méthode de paiement	119
38.8 Changement des informations bancaires	119
39 Conclusion	120

1 Use cases diagrams

1.1 Points communs

Etant donné la nature des deux applications, de nombreux points communs sont à souligner dans les use cases tels que les logs, gestion des notifications et gestion des paramètres.

Concernant le système de log, cela peut s'apparenter à une application "classique" contenant des utilisateurs avec des fonctionnalités telles que créer un compte, s'authentifier, se déconnecter et réinitialiser le mot de passe.

Pour la gestion des notifications, l'utilisateur aura la possibilité de voir les détails des notifications qu'il aura reçues, de les marquer comme lues et aura le choix de les accepter ou de les refuser. De plus, l'utilisateur pourra rafraîchir la page pour voir apparaître de nouvelles notifications (de même que mentionné dans l'énoncé).

En ce qui concerne la gestion des paramètres, l'utilisateur aura notamment la possibilité de modifier son mot de passe en ayant reçu un mail de confirmation au préalable. Au sujet de la gestion des langues, ce dernier aura le droit d'en ajouter, d'en choisir une qui sera sa langue favorite et de changer la langue actuelle de l'application.

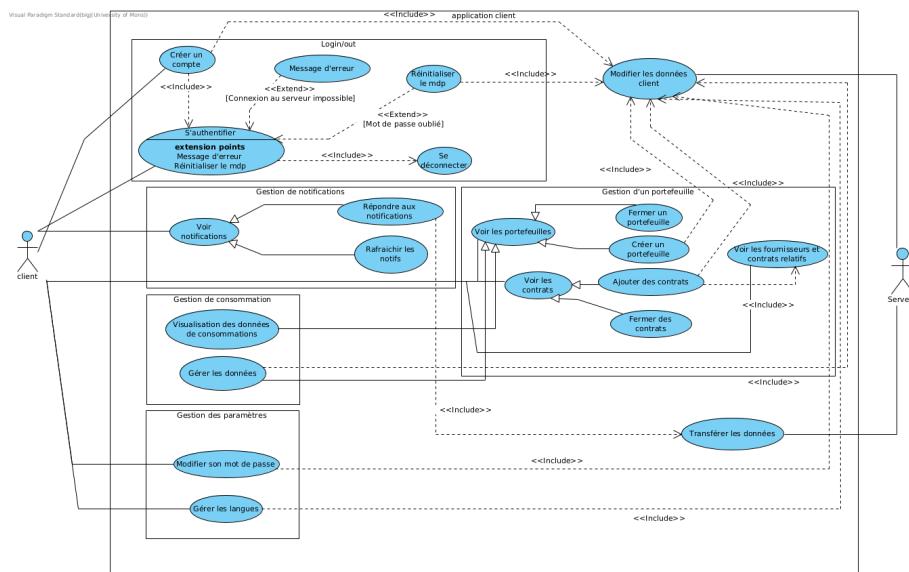
Nous pouvons observer que le serveur est un acteur qui enregistrera toutes les informations utiles et qu'il jouera le rôle de correspondant entre le client et le fournisseur par le biais du use case : "transférer les données". Pour permettre une meilleure lisibilité et compréhension de notre diagramme, nous avons décidé d'impliquer le serveur uniquement dans certaines actions. De ce fait, celui-ci n'est pas relié aux Use cases portant par exemple le nom de « Fermer un portefeuille ».

1.2 Client

Les deux points principaux à noter sont les Use cases « Voir les portefeuilles » et « Voir les contrats ». En effet, c'est à partir de ces derniers que la logique même des fonctionnalités disponibles pour le client repose.

Lorsque le client est sur la section « Voir les portefeuilles », il pourra créer un portefeuille, fermer un portefeuille, voir les données de consommation et gérer ces données. Il convient de signaler que nous avons décidé de créer une sous-catégorie supplémentaire "Gestion de consommation" afin d'apporter une meilleure lisibilité et séparation des sections.

Quant à la section « Voir les contrats », il aura la capacité d'ajouter ou fermer des contrats. En vue d'ajouter des contrats, le client devra forcément voir les fournisseurs et contrats relatifs. Nous avons choisi que cette option devra également être disponible sans se rendre dans la section mentionnée précédemment.

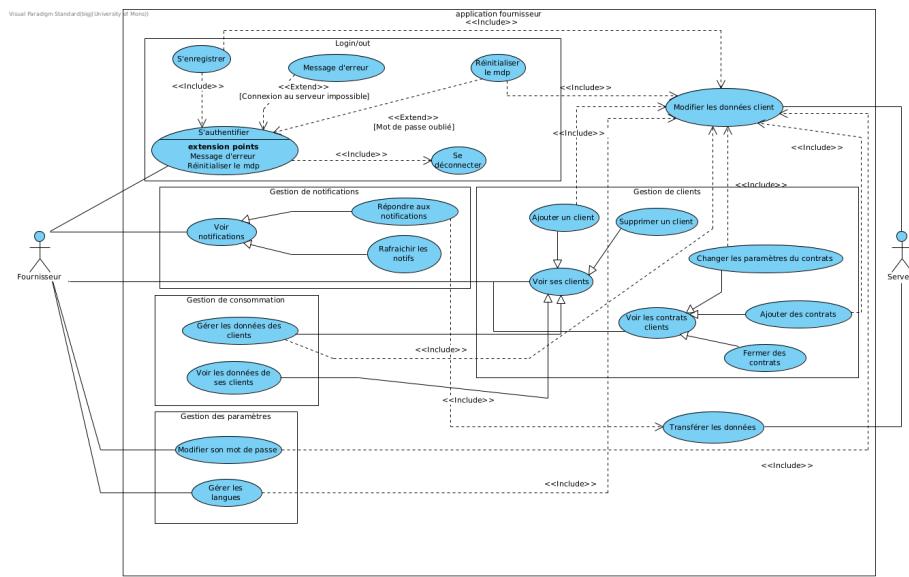


1.3 Fournisseur

Pour le fournisseur, les principales différences résident dans le fait qu'il n'y aura plus les sections « Voir les portefeuilles » et « Voir les contrats » mais « Voir ses clients » et « Voir les contrats clients ».

Dans la même optique que pour la gestion d'un portefeuille chez le client, le fournisseur une fois sur la section « Voir ses clients » sera apte à ajouter ou supprimer un client, voir les données de ses clients ainsi que les gérer.

Ce dernier aura, de plus, la possibilité d'ajouter ou fermer des contrats et changer les paramètres des contrats/propositions sur « Voir les contrats clients ».



2 Interaction overview diagrams

2.1 Introduction

Nous avons réalisé nos interaction overview diagrams sur base de nos use cases diagrams. Ce qui nous a permis de détailler la structure de ces diagrammes intuitivement.

Ces derniers permettront d'obtenir une idée globale du parcours des utilisateurs sur l'application ainsi que les fonctionnalités leur étant disponibles.

Nous décrirons donc deux applications, l'application « client » et l'application « fournisseur » et de nouveau, nous verrons apparaître des points communs entre ces dernières.

2.2 Points communs

1. Système de logs:

Une fois que l'utilisateur arrivera sur l'application, plusieurs choix s'offriront à lui. Soit ce dernier est nouveau et il peut donc s'enregistrer en respectant les conditions qui lui sont imposées telles qu'une adresse mail valide, vérifiée par une demande de confirmation, et inexistante sur l'application, la mention de son rôle sur l'application (client ou fournisseur), un mot de passe ayant un niveau de sécurité convenable et le choix de sa langue, soit, il possède déjà un compte et se connecte.

Cependant, celui-ci pourra réinitialiser son mot de passe à l'aide d'un mail de confirmation ou réessayer de se connecter si lors de la vérification des données, les identifiants sont invalides.

De plus, lorsque l'utilisateur sera connecté sur l'application, il aura la possibilité de se déconnecter s'il le souhaite.

2. Menu représenté par un « grand » fork :

Une fois connecté, l'utilisateur aura accès à un menu reprenant la déconnexion, l'accès aux paramètres, les notifications et les éléments lui étant propres en fonction qu'il soit fournisseur ou client.

Cette façon de penser nous semblait suffisamment intuitive pour que l'utilisateur puisse naviguer sur l'application facilement. Nous avons également choisi de permettre un « retour » à ce menu lorsque nous sommes dans les sections le composant.

3. Accès aux paramètres :

Une fois sur l'onglet des paramètres, plusieurs choix s'offrent à lui :

- a) Gérer les langues : plus précisément, il pourra en ajouter, en choisir une qui sera sa langue favorite et changer la langue actuelle de l'application comme expliqué précédemment dans les Use cases diagrams.
- b) Modifier son mot de passe : pour ce faire, l'utilisateur recevra un mail de confirmation permettant de le changer en toute sécurité.

4. Les notifications :

Une fois sur cette section, l'utilisateur pourra répondre à ses notifications, c'est-à-dire, marquer comme lues les contrats, les accepter, les refuser ou encore de voir leurs détails à savoir les données relatives à ces derniers.

De surcroît, s'il le désire, ce dernier aura le droit de rafraîchir ses notifications par lui-même sans attendre que cela se réalise automatiquement.

2.3 Accès à l'application pour les clients

Lorsque d'un client voudra **voir ses portefeuilles**, il retrouvera plusieurs fonctionnalités :

1. Créer un portefeuille
2. Fermer un portefeuille
3. Visualisation des données de consommation
4. Gérer les données

- Créeer un portefeuille :

Cette première option permettra au client comme son nom l'indique de créer un portefeuille.

Dans cette optique, il devra rentrer le nom et l'adresse du portefeuille, le code EAN et ajouter les contrats liés.

- Fermer un portefeuille :

Cela donnera la possibilité au client de clôturer un portefeuille, une fois qu'il n'en aura plus l'utilité.

- Visualisation des données de consommation :

Le client pourra observer les valeurs associées à sa consommation, ainsi que les contrats liés à ces dernières.

- Gérer les données :

Il sera également apte à les modifier, c'est-à-dire, renouveler ses données de consommation et ajouter, modifier ou supprimer les contrats étant associés à un portefeuille.

Grâce à cette option, ce dernier aura pleinement la main sur ses portefeuilles et aura la faculté de les reformer à sa guise.

Le menu offre en outre la possibilité de **voir les contrats** ou plus précisément de voir ses contrats, l'utilisateur sera donc en mesure de, d' :

1. Ajouter des contrats
2. Fermer des contrats

- **Ajouter des contrats :**

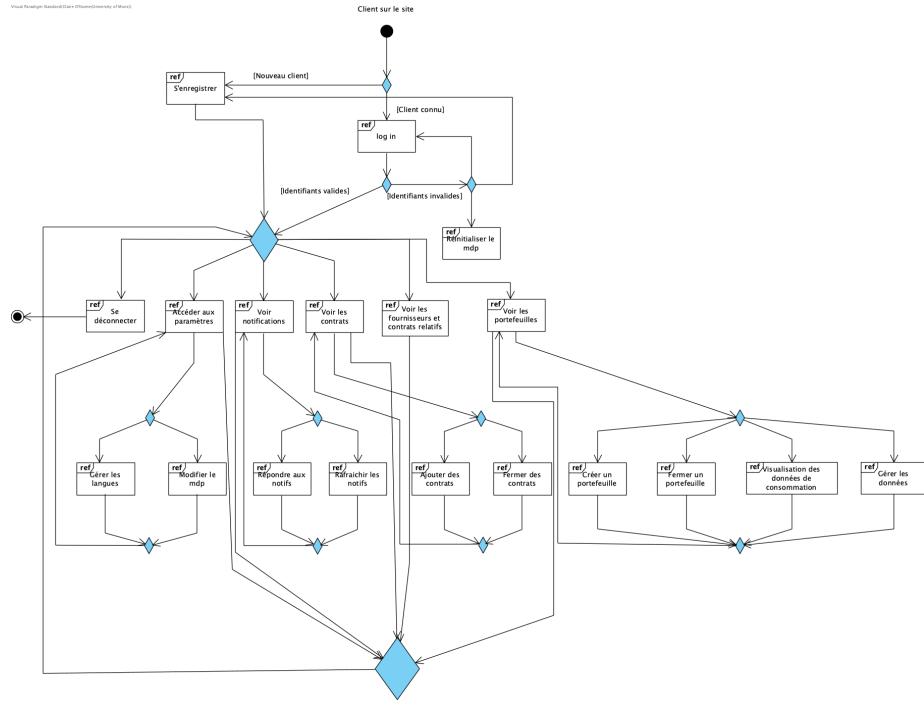
Le client se retrouvera alors sur la page des **fournisseurs et contrats relatifs** répertoriant tous les contrats qu'il ne possède pas encore et les verra en détails avant de les ajouter s'il le désire.

- **Fermer des contrats :**

Cela lui donnera la possibilité de clôturer un contrat s'il trouve cela plus judicieux.

L'ajout et la fermeture de contrats impliquera une **notification** sur l'application du fournisseur en question.

Nous avons décidé de laisser la possibilité au client de voir les fournisseurs et contrats relatifs sans forcément passer par la liste de ses contrats afin de lui permettre un accès plus rapide.



2.4 Accès à l'application pour les fournisseurs

Au moment où le fournisseur accèdera à l'onglet « **Voir les contrats clients** », plusieurs choix s'offriront à lui :

1. Ajouter des contrats
2. Fermer des contrats
3. Changer les paramètres des contrats

- **Ajouter des contrats :**

Cette option permettra au fournisseur d'ajouter de nouveaux contrats qu'il aura créés aux contrats déjà existants en spécifiant les valeurs relatives (prix, type de fourniture et compteurs).

- **Fermer des contrats :**

Cela lui donnera la possibilité de clôturer un contrat, une fois qu'il n'en aura plus l'utilité.

- **Changer des contrats :**

Le fournisseur sera apte à changer les termes du contrat s'il trouve cela approprié.

Évidemment comme pour la fermeture de contrats, le changement sera notifié au client.

De plus, le fournisseur sera à même de « **Voir ses clients** » de manière distincte, plus précisément, ce dernier retrouvera une liste de ses clients avec les contrats leur étant associés :

1. Ajouter un client
2. Supprimer un client
3. Voir les données de ses clients
4. Gérer les données de ses clients

- **Ajouter un client :**

Ce dernier aura la capacité d'ajouter un client en l'associant à un de ses contrats en particulier.

- **Supprimer un client :**

Cela donnera la possibilité au fournisseur de clôturer tous les contrats actifs avec un client, s'il est d'avis que cela est nécessaire.

- **Visualisation des données de ses clients :**

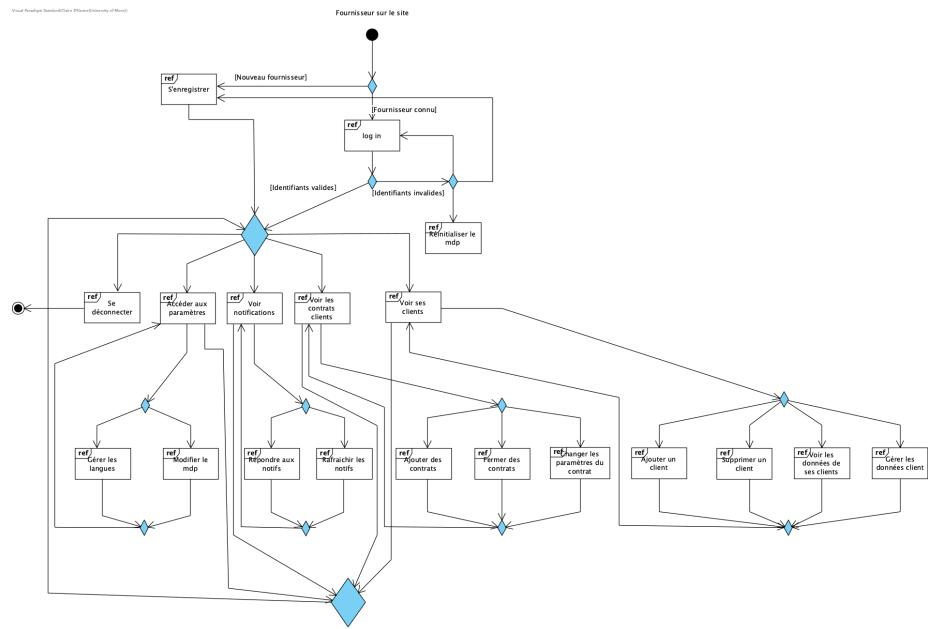
Celui-ci pourra observer les valeurs associées aux contrats de ses clients, ainsi que leur consommation.

- **Gérer les données de ses clients :**

Le fournisseur aura également la capacité de les modifier, c'est-à-dire, modifier, ajouter ou supprimer les contrats étant associés à un client.

Grâce à cette option, ce dernier aura pleinement la main sur ses clients et aura la faculté de les gérer à sa guise.

L'ajout, la modification et la fermeture de contrats impliquera une notification sur l'application du client en question.



3 Entity relationship diagram

3.1 Préambule

Dans cette partie, nous allons parler de la modélisation de la base de données. Chaque sous-section parlera d'une relation. Dans ces sous-sections, nous parlerons du fonctionnement de certains attributs et nous définirons nos clés primaires et étrangères.

De manière générale, nous avons fait en sorte d'obtenir le moins de redondance possible entre les différentes relations pour convenir au mieux à la 3NF¹.

3.2 Utilisateur

Nous partons du principe qu'un fournisseur et un client sont avant tout des utilisateurs de l'application. De ce fait, les deux personnes auront un identifiant "id" ("u" + 9 digits). Cet identifiant représente la quantième personne qui a été inscrite. Par exemple, si je suis le premier utilisateur alors j'aurai l'identifiant : u000000000.

A noter que le mot de passe sera stocké dans la base de données après avoir été "hashé" par l'algorithme bcrypt² pour éviter toute fuite dans le cas où la base de données serait compromise.

PK(utilisateur) = **id**

FK(utilisateur) REFS fournisseur = **id**
FK(utilisateur) REFS client = **id**

¹https://en.wikipedia.org/wiki/Third_normal_form
²<https://fr.wikipedia.org/wiki/Bcrypt>

3.3 Langue

Dans cette relation, nous posons que l'utilisateur peut avoir plusieurs langues. De ce fait, nous avons une clé primaire composée.

De plus, la langue enregistrée peut être une langue préférée (1 si c'est le cas, 0 sinon).

Ensuite, un attribut binaire `langue_actuelle` est stocké. Cet attribut nous permet de savoir quelle langue l'utilisateur est en train d'utiliser ou a utilisé pendant sa dernière session (1 si c'est le cas, 0 sinon).

`PK(langue) = id et langue_enregistrée`

`FK(langue) REFS utilisateur = id`

3.4 Client

Du côté de la relation "client", nous y trouvons un seul attribut. Cela nous permet d'avoir un utilisateur n'ayant aucun portefeuille.

`PK(client) = id_client`

3.5 Portefeuille

L'adresse sera définie comme : nomdeville-nomderue-numérodemaison.

`PK(portefeuille) = adresse`

`FK(portefeuille) REFS client = id_client`

3.6 Contrat du portefeuille

Dans "contrat du portefeuille" et pour toute autre relation, `id_contrat` sera défini comme ceci : "c" + quantième contrat créé.

`PK(contrat du portefeuille) = adresse et id_contrat`

`FK(contrat du portefeuille) REFS portefeuille = adresse`

3.7 fournisseur

Un fournisseur sera perçu avec son identifiant. De même que le client, un fournisseur aura la possibilité de n'avoir aucun contrat ce qui justifie le fait qu'il n'y ait qu'un seul attribut dans la relation.

`PK(fournisseur) = id_fournisseur`

3.8 Proposition

Cette section correspondra aux offres du fournisseur avec tous les paramètres qui seront associés. Nous posons qu'un nom d'offre ne peut être unique par rapport à plusieurs fournisseurs. De ce fait, nous avons une clé primaire composée. L'attribut `nom_proposition` sera du type : "p" + nom que le fournisseur aura donné.

A noter que la localisation sera définie par un binaire de taille 3: le premier bit sera la région Wallonne, le second sera la région flamande et le troisième la région Bruxelles-Capitale. Comme exemple, 101 dira que le contrat sera disponible dans la région Wallonne et Bruxelles-Capitale. Du coté des autres types binaires, 1 sera vrai et 0 faux.

De plus, certains paramètres seront dépendants d'autres paramètres. Comme exemple, nous pouvons dire qu'il ne sera pas possible d'avoir un prix différent entre les heures pleines et les heures creuses si le type d'énergie est l'eau ou qu'on a un compteur mono-horaire.

Enfin, si nous avons le même prix pendant les heures creuses et les heures pleines, il sera donc inutile de conserver une heure bien précise (`début_heures_creuses` et `début_heures_pleines`³ pourront être nuls).

`PK(proposition) = nom_proposition et id_fournisseur`

`FK(proposition) REFS fournisseur = id_fournisseur`

3.9 Contrat du fournisseur

`PK(contrat du fournisseur) = id_contrat et id_fournisseur`

`FK(contrat du fournisseur) REFS fournisseur = id_fournisseur`

3.10 Contrat

La table contrat contiendra tous les éléments typiques d'un contrat tels que le fournisseur, le client, la proposition, ect,...

A noter que la date de fermeture peut être nulle. Cela conviendra à un contrat à durée indéterminée.

`PK(contrat) = id_contrat`

`FK(contrat) REFS contrat du fournisseur = id_fournisseur et id_contrat`

`FK(contrat) REFS proposition = nom_proposition`

`FK(contrat) REFS contrat du portefeuille = adresse et id_contrat`

`FK(contrat) REFS compteur = ean`

³format: entier tel que les deux premiers digits seront l'heure et les minutes pour les derniers

3.11 Compteur

Dans cette section, nous représentons le compteur associé à un contrat. Le compteur sera identifié par le code EAN.

$\text{PK}(\text{compteur}) = \text{ean}$

3.12 Consommation

Dans la consommation, nous pouvons noter qu'il est possible qu'un client n'ait rien consommé durant la journée. De ce fait, `consommation_journalière` peut être mis à 0. On a aussi posé qu'un client ne pouvait consommer plus de 30 kw/heure par jour⁴ ce qui nous permet de restreindre la taille de `consommation_journalière` à 3 digits.

$\text{PK}(\text{consommation}) = \text{ean}$ et `date`

$\text{FK}(\text{consommation}) \text{ REFS } \text{compteur} = \text{ean}$

3.13 Notification

Les notifications seront représentées par un identifiant ("n" + quantième notification créée).

En plus de ça, un contexte relativement court sera posé jusqu'à plus ample information.

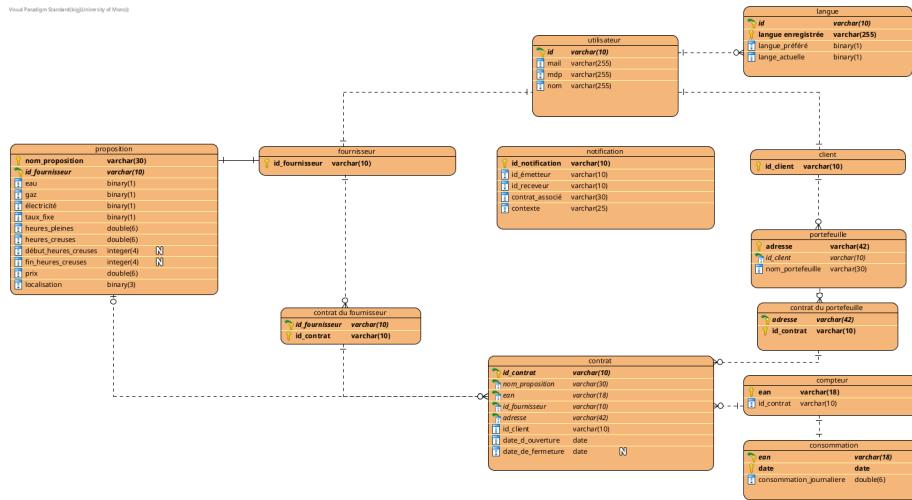
Enfin, un binaire permettra de savoir de qui vient la notification (1 si c'est le fournisseur, sinon c'est le client).

A noter que cette relation ne contient aucune clé étrangère puisqu' il est impossible de faire en sorte que `id_receveur` et `id_émetteur` référencent la table utilisateur.

$\text{PK}(\text{notification}) = \text{id_notification}$

⁴source : <https://shrinkthatfootprint.com/average-household-electricity-consumption/>

3.14 Schéma



4 Class diagram

4.1 Introduction

Étant donné la taille du projet, nous avons évidemment décidé de séparer le projet en plusieurs parties. Au niveau du serveur, nous pouvons compter 3 grandes parties.

- Le package API
- Le package database
- Le package dataObject

En outre, il y a également la classe de base nommée App. Nous en reparlerons après.

4.2 Le package API

Ce package contient toute la partie logique de l'ApiRest. Pour créer le serveur API nous avons décidé d'utiliser le framework Vertx⁵, nous expliquerons notre choix dans le rapport sur l'implémentation. Pour créer une ApiRest, nous avons d'abord besoin d'une classe (MyApi) qui hérite de AbstractVerticle (une classe abstraite venant du package vertx). MyApi permet de lancer le serveur à l'aide de la méthode start.

Avant de la lancer, il faut d'abord configurer les routes de notre API. Notre API étant conséquente, nous avons décidé de la séparer en 4 sous-routes (parties distinctes).

a) **LogApi:**

Celle-ci va être utilisée pour toutes les opérations liées au login. Autrement dit, pouvoir se connecter ou se déconnecter, créer un compte et réinitialiser son mot de passe en cas d'oubli. Nous avons aussi rajouté une méthode getCode pour envoyer un code de vérification par mail lorsque l'utilisateur veut créer un compte ou pour réinitialiser son mot de passe.

b) **ClientApi:**

Celle-ci va être utilisée pour toutes les requêtes liées aux clients lorsqu'ils seront connectés. Ils pourront effectuer toutes les opérations qu'ils souhaitent au niveau du portefeuille ainsi que voir tous leurs contrats et pour finir voir toutes les propositions des fournisseurs ou une en particulier. Le client pourra évidemment proposer de conclure un contrat en fonction des propositions précédemment vues.

c) **ProviderApi:**

Celle-ci va être utilisée pour toutes les requêtes liées aux fournisseurs lorsqu'ils seront connectés. Ils pourront effectuer toutes les opérations qu'ils souhaitent au niveau de leurs propositions mais également voir tous les clients de l'application, tous leurs clients ou un client en particulier. Contrairement au client, ils ont la possibilité de supprimer une ou toutes les données de consommations liées à un contrat. De plus, tout comme le client, ils pourront proposer de conclure un contrat à n'importe quel client à partir de l'une de ces propositions.

d) **CommonApi:**

Celle-ci va être utilisée pour toutes les requêtes qui sont communes aux clients et aux fournisseurs lorsqu'ils seront connectés. On y trouve les méthodes pour les langues, pour les notifications et pour les données de consommations. Il y a également une méthode pour voir un contrat en particulier et une qui permet de changer de mot de passe.

⁵<https://thierry-leriche-dessirier.developpez.com/tutoriels/java/creer-API-rest-vertx-5-minutes/>

Chacune de ces parties va donc créer une sous-route grâce à la méthode `getSubRouter` car elles implémentent toutes l'interface `Router`. Et c'est donc dans la méthode `start` de `MyApi` que toutes ces sous-routes vont être rassemblées pour finalement lancer le serveur API.

Notez que toutes les méthodes qui vont être appelées par l'API sont en privé car seul le serveur API peut les appeler et elles ont le même argument (`routingContext`) car c'est cette variable qui contient la requête de l'utilisateur ainsi que le corps de la requête. Et c'est également par cette variable que nous pourrons répondre à l'utilisateur.

Constatez également que ces quatre parties héritent d'une classe abstraite nommée `AbstractToken`. Comme son nom l'indique, elle va gérer les tokens. Autrement dit, lorsqu'un utilisateur se connecte, on va créer un token et c'est grâce à ce token qu'il pourra envoyer des requêtes aux classes `clientApi`, `ProviderApi` et `CommonApi`. Le token se supprimera automatiquement après un certain délai (15 minutes) ou quand l'utilisateur se déconnectera. Notez que si le token est expiré, l'utilisateur devra se reconnecter.

4.3 Le package database

Le package `database` contient toutes les méthodes qui vont communiquer avec la base de données. Une grande partie de celles-ci ont exactement le même nom que dans le package `API` puisqu'elles sont contenues dans la suite du programme. L'utilisateur envoie une requête à l'API, en fonction de la requête, la bonne méthode est appelée et appellera la méthode de `database` portant le même nom pour accéder aux données. Pour séparer le programme, nous avons divisé cette partie en trois. Nous retrouvons donc une classe mère et deux classes filles.

`CommonDB` est la classe mère de ce package, elle reprend toutes les méthodes communes aux clients et aux fournisseurs comme son nom l'indique. Outre les méthodes de l'API, nous retrouvons également les méthodes `createNotification`, `deleteNotification` et `createContract` car elles devront être appelées à la suite d'une requête et non directement. Nous avons aussi décidé d'ajouter les méthodes `createId`, `getDataOfTable` et `deleteDataOfTable` car ce sont des actions qui seront souvent utilisées et que nous pouvons donc généraliser.

`ClientDB` et `ProviderDB` sont les deux classes enfants, elles s'occupent respectivement du côté client et du côté fournisseur comme leurs noms l'indiquent. Elles héritent de la classe `CommonDB` pour avoir accès à certaines méthodes comme par exemple `createId`, `getDataOfTable` et `deleteDataOfTable`. Notez qu'au niveau du client nous avons rajouté une méthode `walletIsEmpty` qui permet de savoir si un portefeuille de client est vide.

4.4 Le package dataObject

Le package dataObject contient toutes les classes représentant un objet, c'est-à-dire:

- Les portefeuilles.
- Les contrats.
- Les clients (du point de vue des fournisseurs).
- Les propositions (que les fournisseurs ont créées).
- Les notifications.

Remarquez également que nous avons rajouté une classe TypeEnergy pour énumérer tous les types d'énergies.

Nous avons décidé de créer ces objets pour transférer facilement les données. En effet toutes les classes ne contiennent que des attributs avec des assesseurs à quelques exceptions près. Notez que nous aurions donc pu mettre les variables en final mais nous avons préféré laisser comme ça car dans le cas contraire, nous aurions eu des constructeurs vraiment longs. Ces classes seront donc instanciées lorsque nous recevrons un objet par l'API ou quand nous prendrons des données de la base de données pour facilement les envoyer à l'utilisateur en format json (io.vertx.core.json.Json).

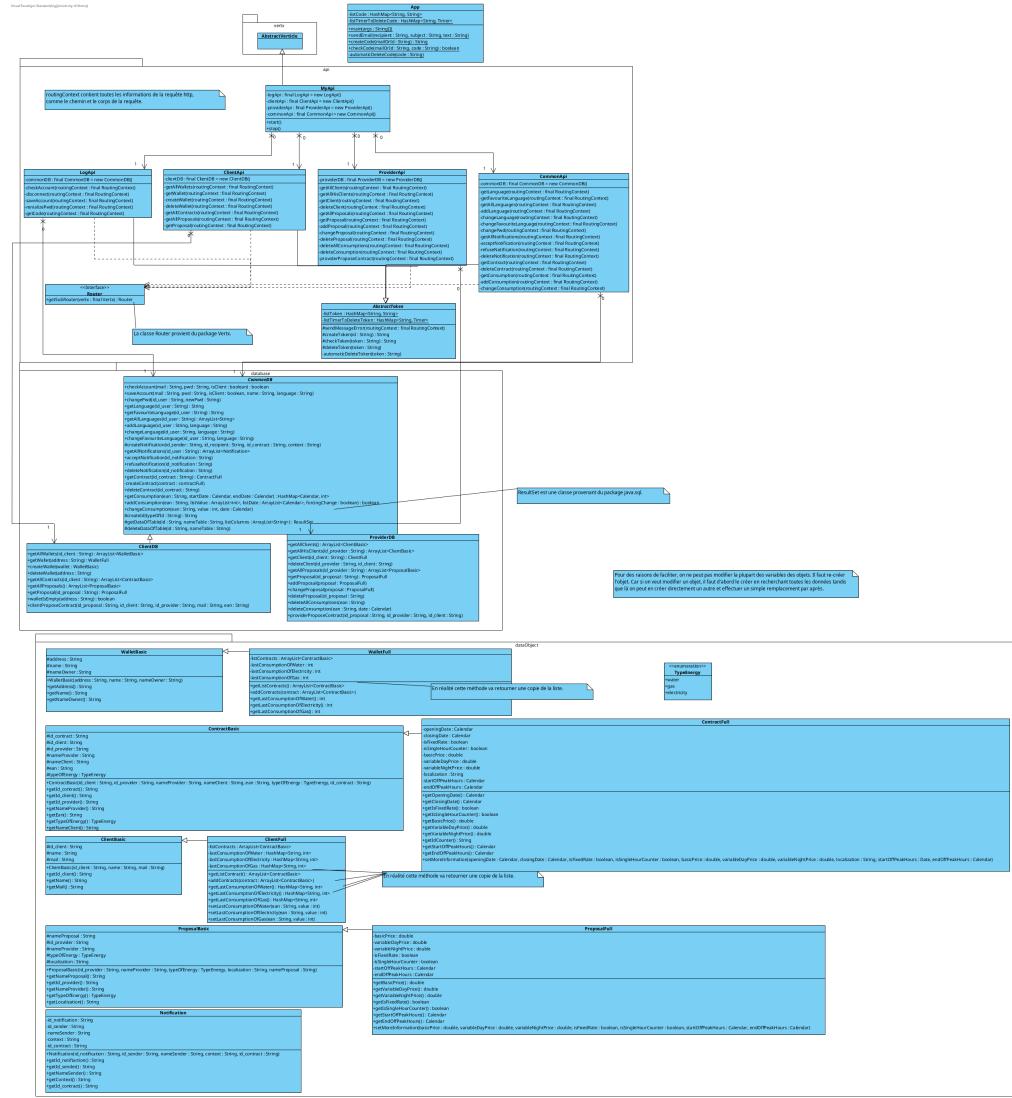
À l'exception de la classe notification, nous avons choisi de séparer chaque sorte d'objets en deux parties, une partie basique avec les informations primaires et une partie complète avec le reste des informations. Cela nous permet, lorsque l'utilisateur souhaite par exemple voir tous ses portefeuilles, d'envoyer seulement le nécessaire au lieu d'envoyer toutes les données en une fois. Ce n'est que lorsque l'utilisateur cliquera sur un portefeuille en particulier que nous enverrons toutes les données correspondant à celui-ci.

4.5 La classe App

La classe App est le début du programme. C'est elle qui est chargée de lancer le programme comme toutes les applications java grâce à la méthode nommée "main". En effet, cette méthode va lancer l'API de la classe Myapi. Notez que nous n'avons pas directement lié ces deux classes entre elles car l'API se lance indirectement par le framework Vertx.

La classe App contient également quelques méthodes statiques utilisées par les autres parties du projet. Ce sont ces méthodes qui s'occupent de la partie mail du projet, notamment "sendEmail" qui enverra un mail à un utilisateur qui souhaite créer un compte, réinitialiser son mot de passe ou bien le changer. Les autres méthodes s'occupent du "code". En effet, nous avons décidé que lorsqu'un utilisateur souhaite réaliser une des tâches précédentes, nous allons lui envoyer un code par mail avec quelques explications. Nous avons donc besoin d'une méthode pour générer le code aléatoirement, une pour vérifier que l'utilisateur a entré le bon code et pour finir une pour supprimer le code si l'utilisateur a mis trop de temps pour l'envoyer.

En outre, pour envoyer un mail, nous utilisons l'API javaxmail.



5 Sequence diagrams

5.1 Introduction

Les sequences diagrams nous permettent de détailler le chemin effectué lorsqu'un utilisateur souhaite interagir avec l'application.

Dans le but de permettre une meilleure compréhension de ces diagrammes, nous les avons séparés en trois parties.

1. Common
2. Client
3. Provider

Afin de les réaliser convenablement, nous sommes repartis de notre class diagram et nos Use cases diagrams et avons représenté l'interaction des méthodes avec l'API et la base de données.

Avant de commencer, il est important de signaler dans le but d'éviter toute répétition, que la vérification du token s'effectuera pour chaque action lorsqu'un utilisateur est connecté.

Si le token est bon, on continue le programme, sinon, on envoie un message d'erreur à l'aide de la méthode "sendMessageError".

Pour rappel, l'utilisateur n'appelle pas directement les méthodes partant de sa lifeline mais utilise des requêtes de l'API. De plus, vous pouvez constater que dans le diagramme des classes, nous n'avons pas mis de type de retour aux méthodes de l'API. Cependant, dans les diagrammes de séquences, nous avons quand-même représenté une flèche de retour. Cela est dû au fait que pour retourner quelque chose à l'utilisateur, nous utilisons une méthode de la variable `routingContext`.

5.2 Common

Pour la partie commune aux clients et fournisseurs, nous retrouvons plusieurs Use Cases à décrire.

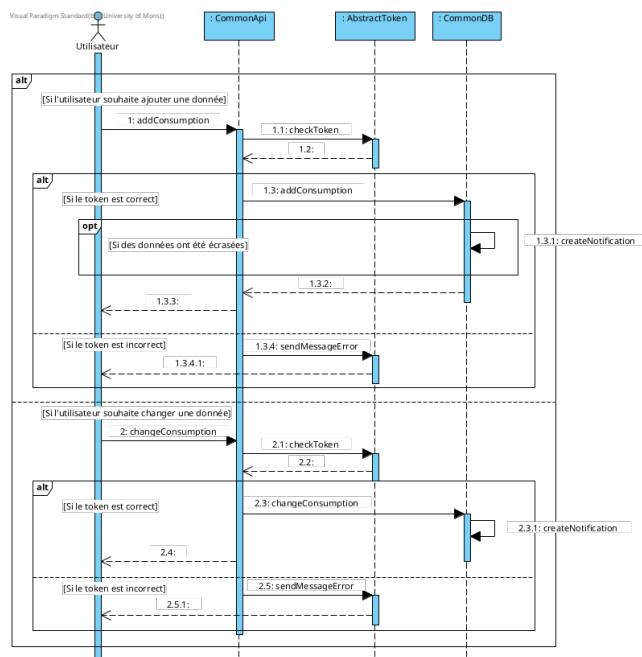
1. Gérer les données
2. Gérer les langues
3. Modifier son mot de passe
4. Répondre aux notifications
5. S'authentifier
6. Visualisation des données de consommations
7. Voir les contrats
8. Voir notifications

5.2.1 Gérer les données

Nous pouvons gérer les données de deux manières. L'utilisateur peut ajouter une donnée de consommation ou en changer une.

Pour ajouter une donnée, il suffit d'appeler la méthode addConsumption. Cette dernière peut écraser des données selon les paramètres utilisés. Lorsque des données sont écrasées, nous devons donc créer une notification pour prévenir l'autre utilisateur concerné par ce changement.

Pour changer une donnée, il suffit d'appeler la méthode changeConsumption. Celle-ci change automatiquement une donnée, nous devons donc créer une notification pour prévenir l'autre utilisateur.

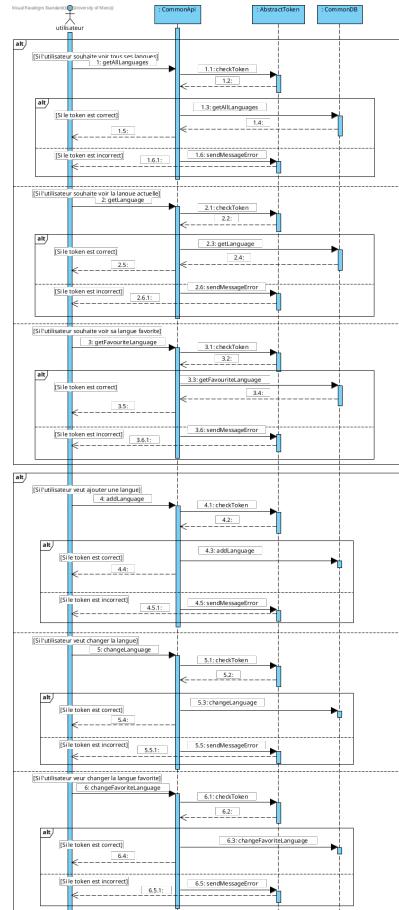


5.2.2 Gérer les langues

Nous pouvons gérer les langues à plusieurs niveaux. L'utilisateur peut voir toutes ses langues, sa langue actuelle et sa langue préférée. En plus de cela, il peut ajouter une langue, changer sa langue actuelle ou encore changer sa langue favorite.

Pour les trois premières fonctionnalités, le principe est le même. On appelle la méthode adéquate et celle-ci va être rappelée dans la partie base de données pour rechercher l'information désirée.

Concernant les autres fonctionnalités, on appelle également la méthode du même nom dans la partie base de données. Remarquez que ces méthodes ne retournent rien.

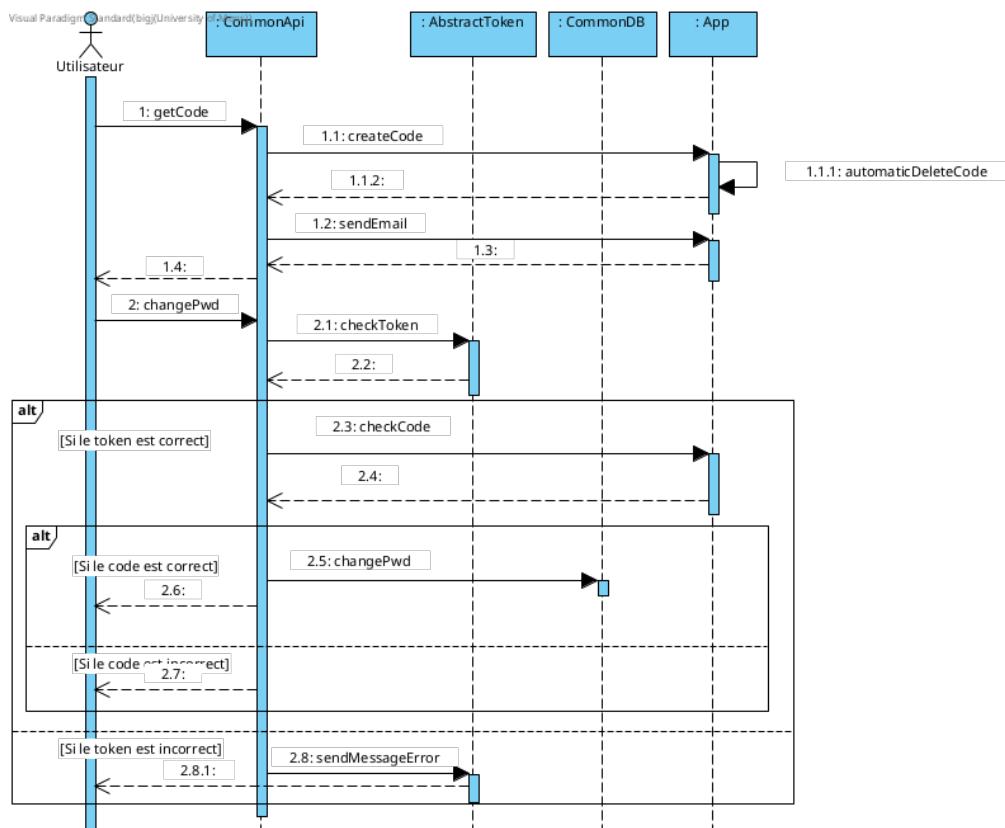


5.2.3 Modifier son mot de passe

Pour modifier son mot de passe de manière sécurisée, nous passons par l'adresse mail de l'utilisateur pour être certains que ce soit bien celui-ci qui veuille changer le mot de passe. Pour se faire, il faut tout d'abord faire appel à la méthode getCode, cette dernière va appeler les méthodes createCode et sendEmail de la classe App.

À noter que createCode va appeler automaticDeleteCode afin que le code se supprime automatiquement au bout d'un laps de temps au cas où l'utilisateur prendrait trop de temps ou n'utiliserait jamais ce code.

L'utilisateur recevra donc par mail un code de vérification. Il pourra ensuite entrer le code et son nouveau mot de passe sur le site pour finalement appeler la méthode changePwd de l'API. Avant de le changer, nous utilisons la méthode checkCode d'App pour vérifier que le code est correct. Si c'est bien le cas, alors nous pouvons appeler changePwd de CommonDB pour changer le mot de passe. Dans le cas contraire, nous renvoyons évidemment un message d'erreur.

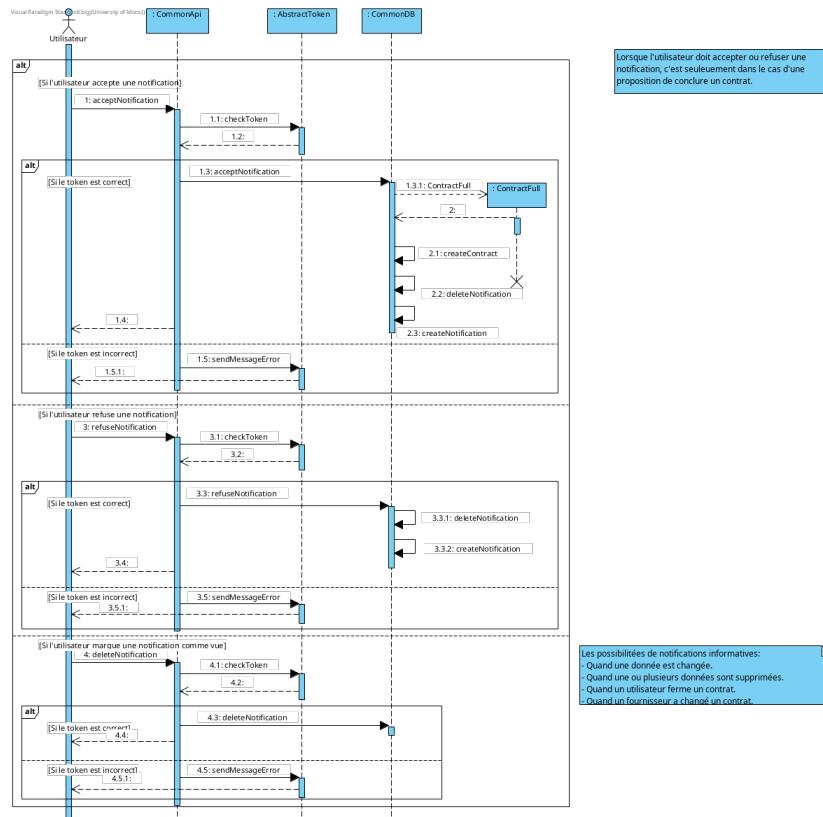


5.2.4 Répondre aux notifications

Répondre aux notifications implique trois possibilités. L'utilisateur peut l'accepter ou la refuser dans le cas d'une demande de contrat ou bien dans les autres cas, noter la notification comme lue.

Lorsqu'un utilisateur accepte une notification (une demande de contrat) dans la partie base de données, nous commençons par créer un objet contractFull à partir de la proposition acceptée, ensuite nous faisons appel à la méthode createContract pour enregistrer le contrat. Nous pouvons donc supprimer la notification actuelle étant donné qu'elle vient d'être acceptée. Pour finir, nous créons une nouvelle notification pour informer à l'autre utilisateur que sa demande a été acceptée.

Dans le cas d'une notification informative, l'utilisateur peut la marquer comme lue. Ce qui implique de la supprimer en appelant la méthode deleteNotification.



5.2.5 S'authentifier

Lorsque l'utilisateur arrive sur la fenêtre de connexion. Deux cas de base sont possibles. Si l'utilisateur n'a pas de compte ou au contraire s'il en possède déjà un.

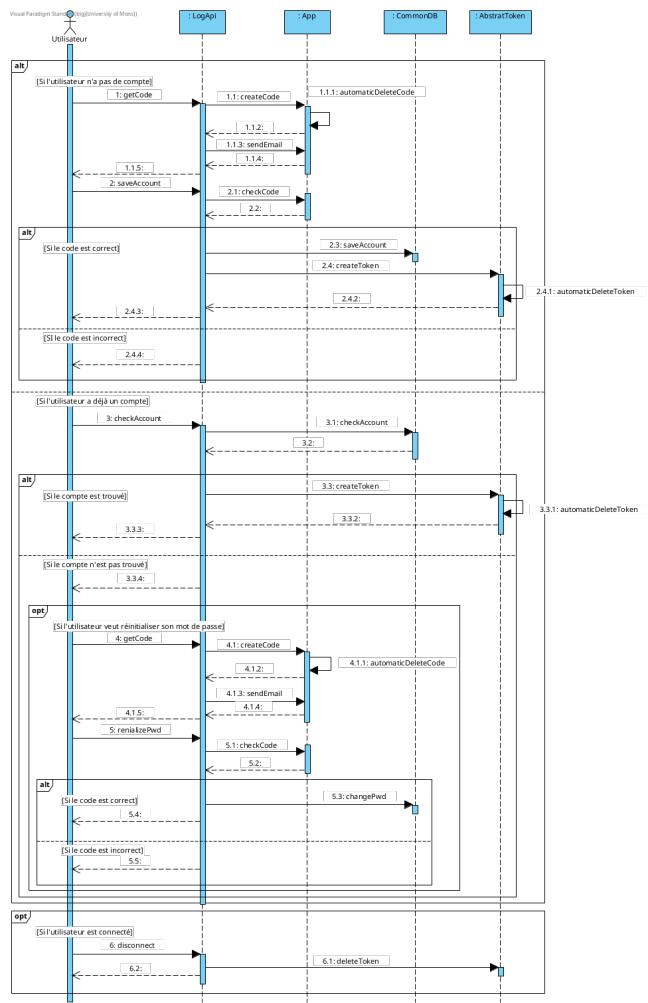
Pour créer un compte, suivant le même principe que le changement de mot de passe. Un code va être créé et envoyé à l'adresse mail que l'utilisateur vient d'entrer grâce aux méthodes déjà expliquées ci-dessus. Quand l'utilisateur souhaite sauvegarder son compte, la méthode saveAcccount est invoquée. Après vérification du code, cette dernière est ré-appelée dans CommonDB pour sauvegarder le compte.

Suite à cela, nous pouvons créer un token de connexion grâce à la méthode createToken de la classe abstraite AbstractToken et l'envoyer à l'utilisateur pour ses prochaines requêtes car en créant son compte, il s'est également connecté. Notez que lors de la création du token, nous appelons automaticDeleteToken pour limiter la durée de vie du token.

Si l'utilisateur possède déjà un compte, nous vérifions les données de connexion grâce à checkAccount. Si les données sont correctes, nous pouvons créer un token et l'envoyer comme expliqué précédemment.

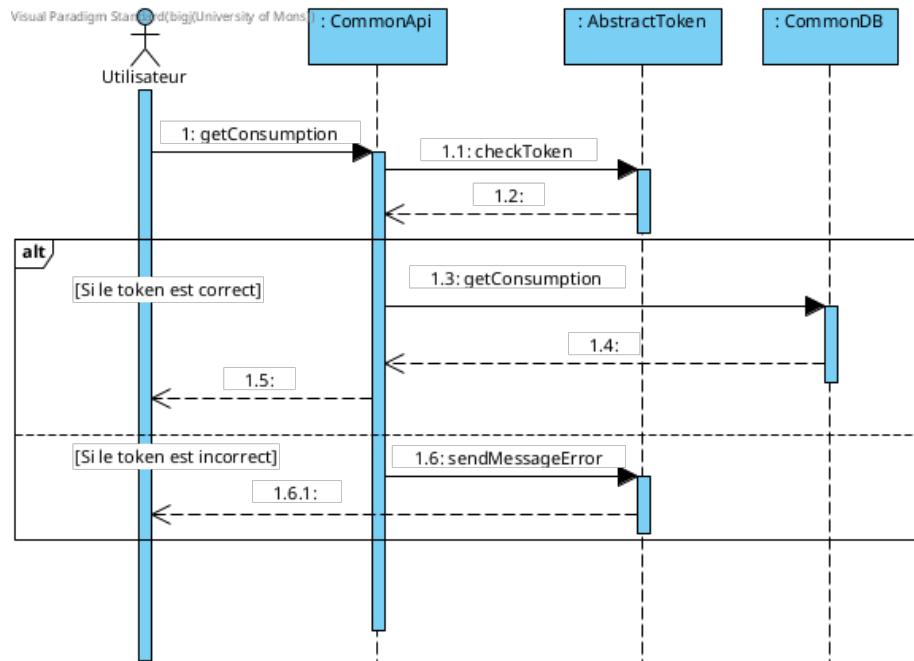
Si l'utilisateur possède un compte mais ne parvient pas à se connecter. Il a la possibilité de réinitialiser son mot de passe. Le processus utilisé est le même que lors du changement de mot de passe expliqué précédemment.

Une fois connecté, l'utilisateur peut de toute évidence se déconnecter en faisant appel à la méthode disconnect. Celle-ci servant juste à appeler deleteToken pour supprimer le token.



5.2.6 Visualisation des données de consommations

Rien de plus simple. Nous invoquons la méthode getConsumption de l'API, ensuite elle est ré-appelée dans CommonDB pour récupérer les valeurs dans la base de données pour finalement les renvoyer.

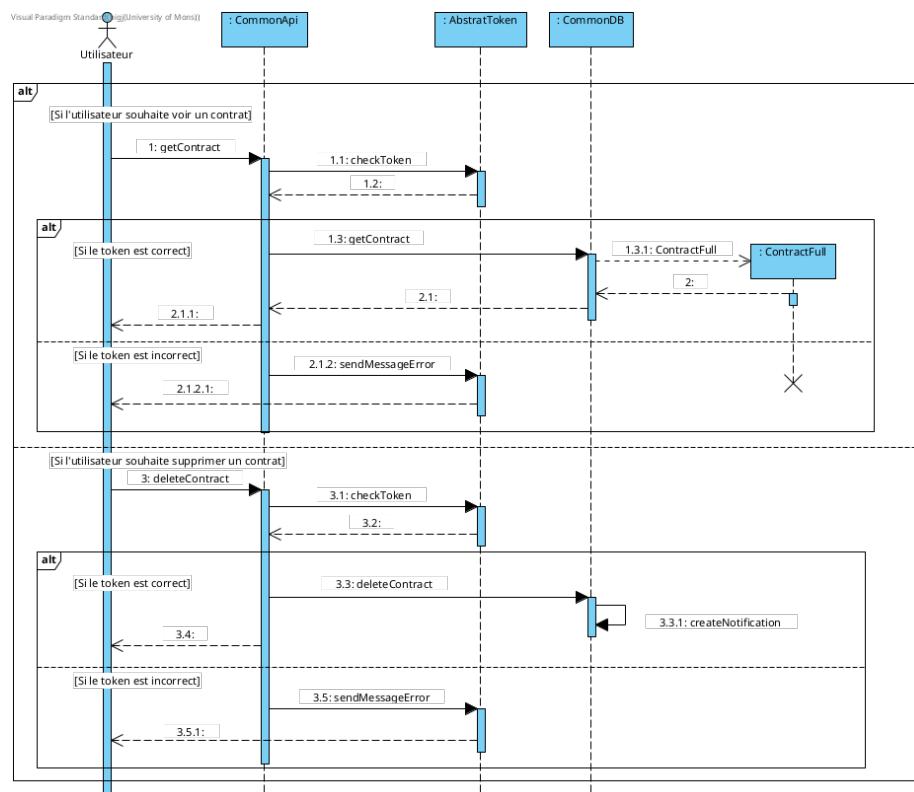


5.2.7 Voir les contrats

Au niveau des contrats dans la partie commune, il est possible de voir un contrat en particulier ainsi que d'en supprimer un.

Il suffit d'appeler la méthode getContract jusqu'à la base de données, nous pouvons ensuite créer un objet ContractFull pour facilement renvoyer les valeurs liées à cet objet. Notez qu'une fois envoyé, l'objet est détruit.

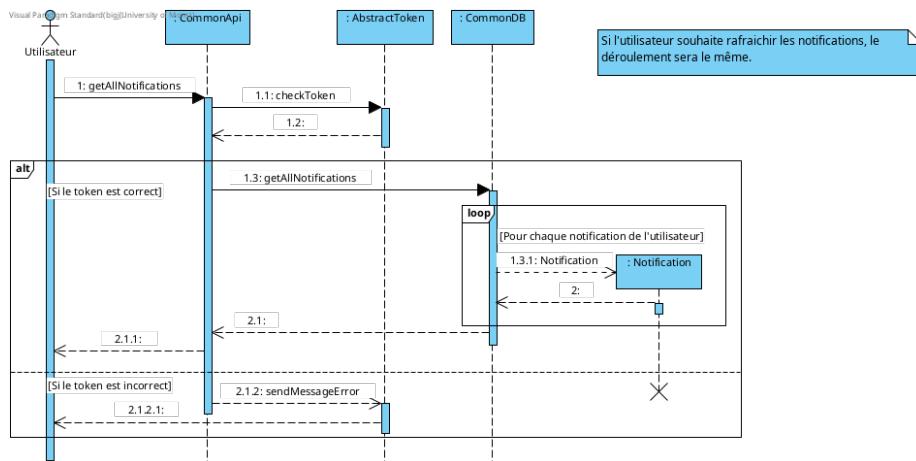
Comme son nom l'indique, deleteContract permet de supprimer un contrat. Une fois son devoir accompli, cette méthode doit également créer une notification pour avertir l'autre utilisateur impliqué par cette suppression. Il n'est pas utile que cette méthode renvoie quelque chose.



5.2.8 Voir notifications

Pour finir, l'utilisateur peut voir ses notifications. Pour cela, nous suivons toujours le même principe en appelant d'abord la méthode getAllNotifications au niveau de l'API et ensuite dans la partie base de données. Nous créons un objet Notification pour chaque notification stockée afin de les envoyer dans une liste vers l'utilisateur. Tous les objets précédemment créés seront évidemment supprimés à la suite de l'envoi de la liste.

Notez que lorsque l'utilisateur rafraîchit les notifications, c'est cette même méthode qui est appelée.



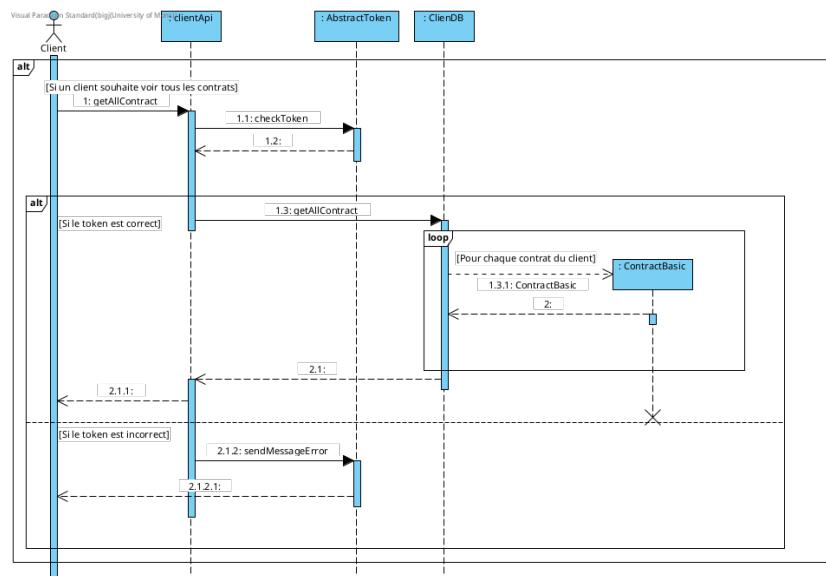
5.3 Client

5.3.1 Contrats

Commençons par le Use Case **”Voir les contrats”**, nous appelons la méthode **”getAllContract”** de **clientApi** et ensuite, la méthode du même nom de **ClientDB**.

Le but étant d’obtenir une **ArrayList** de **”contractBasic”**, nous devons effectuer une boucle afin de récupérer tous les **”contractBasic”**.

Une fois effectuée, nous renvoyons cette liste.



5.3.2 Fournisseurs et contrats relatifs

Continuons avec l'Use Case **"Voir les fournisseurs et contrats relatifs"**. A cet effet, il y aura deux méthodes :

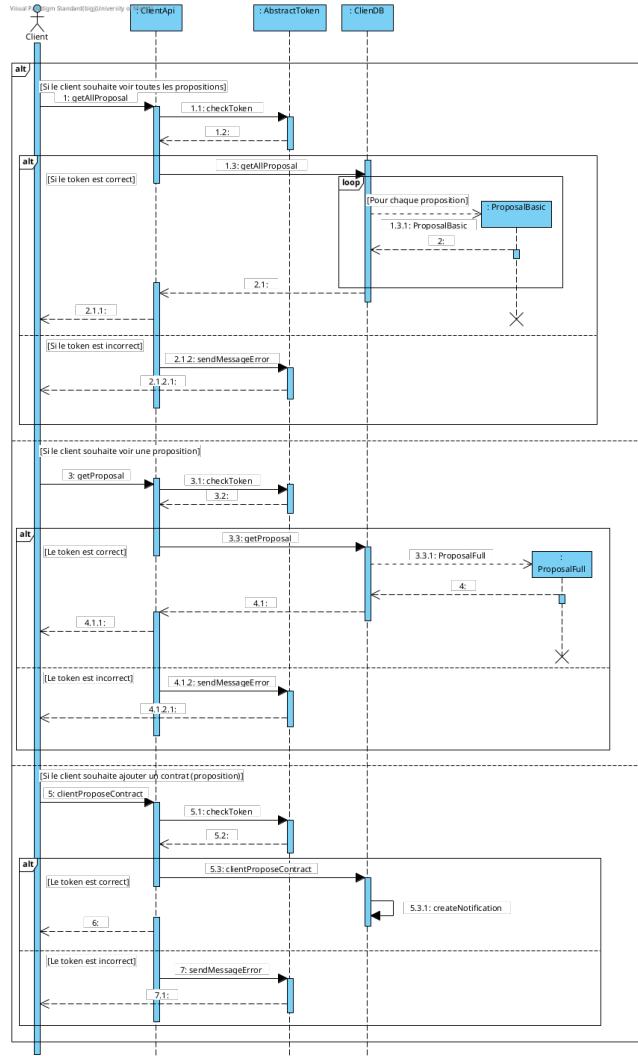
1. getAllProposal si le client souhaite voir toutes les propositions.
2. getProposal si le client souhaite voir une proposition en particulier.

Ces deux méthodes sont appelées une fois de clientApi et ensuite, une fois de ClientDB.

Dans le cas de "getAllProposal", la valeur de retour est une ArrayList de "proposalBasic" donc nous utiliserons une boucle et dans le cas de "getProposal", nous devons simplement retourner une instance de "proposalFull".

Concernant le Use Case **"Ajouter des contrats"**, nous appelons la méthode "clientProposeContract" de clientApi et ensuite, la méthode du même nom de ClientDB.

Etant donné que cette dernière ne doit rien retourner, il n'y aura pas de valeur de retour.



5.3.3 Portefeuilles

L'Use case **"Voir les portefeuilles"** fonctionne de la même manière qu'expliqué précédemment pour "Voir les fournisseurs et contrats relatifs". En effet, il y aura deux méthodes :

1. getAllWallet si le client souhaite voir tous les portefeuilles. On doit donc retourner une ArrayList de "walletBasic".
2. getWallet si le client souhaite voir un portefeuille en particulier. On doit donc retourner une instance de "walletFull".

Passons maintenant aux Use Cases **"Créer un portefeuille"** et **"Fermer un portefeuille"**.

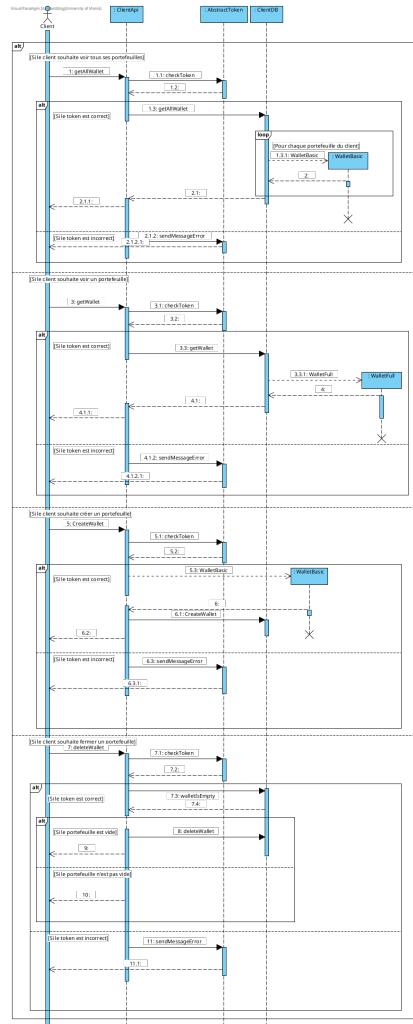
Pour **créer un portefeuille**, nous appelons donc la méthode "createWallet" de clientApi et ensuite, la méthode du même nom de ClientDB.

Nous créons une instance de "walletBasic", que nous devons enregistrer dans "ClientDB" avant de clôturer l'exécution de la méthode car il s'agit d'un nouveau portefeuille à stocker.

Pour **supprimer un portefeuille**, nous appelons la méthode "deleteWallet" de clientApi.

Cependant, avant d'appeler la méthode du même nom de ClientDB, nous devons vérifier que le portefeuille est vide.

Si c'est le cas le portefeuille sera bien supprimé, sinon, on renverra une erreur.



5.4 Provider

Tout d'abord, nous avons fait en sorte qu'un use case peut être représenté via un choix alternatif. Aussi, pour éviter de répéter la même chose dans le rapport, nous posons que toutes les méthodes liées à l'API appelleraient la méthode **checkToken** de **AbstractToken** ce qui impliquera deux choix alternatifs : "Si le token est correct" et "Si le token est incorrect". Dans le dernier cas, on renverra un message d'erreur au fournisseur.

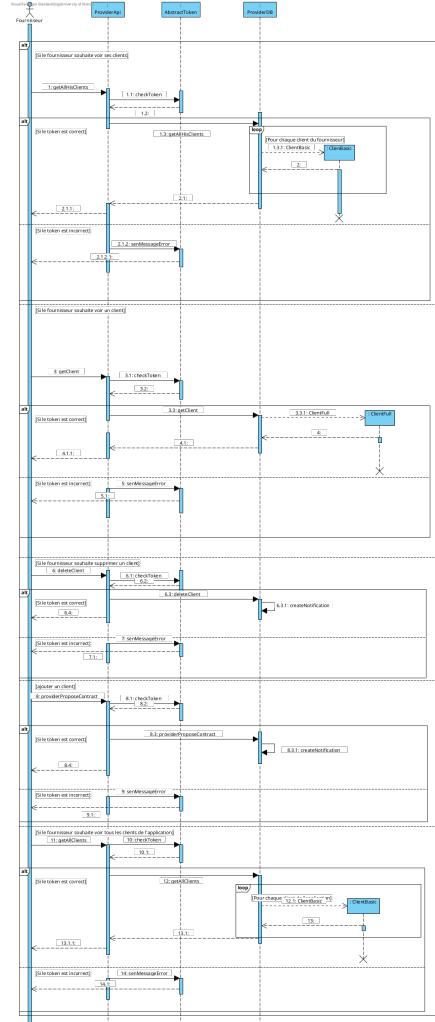
5.4.1 Gestion des clients

Commençons par le use case "voir ses clients". Afin qu'un fournisseur puisse voir ses clients, il lui suffit d'appeler la méthode **getAllHisClients** dans la classe **ProviderApi** et **ProviderDB**. Etant donné qu'on retourne une collection d'objet **ClientBasic**. Nous devons utiliser une boucle. Pour voir un seul de ses clients, il lui suffit d'appeler la méthode **getClient**.

Passons maintenant au use case "supprimer un client". Pour faire cela, la méthode **deleteClient** sera utilisée. Par conséquent, une notification sera envoyée au client ce qui nous oblige à faire appel à la méthode **createNotification**.

Pour ajouter un client, cela se fera par le biais d'une proposition du fournisseur au client. Ceci faisant appel à la méthode **providerProposeContract**. Nous posons comme hypothèse que le client n'a aucun contrat en cours avec le fournisseur.

Pour qu'un fournisseur puisse ajouter des clients il doit d'abord avoir accès à tous les clients de l'application. Ceci se fera par la méthode **getAllClients**. De la même manière que voir "ses clients", nous utiliserons une boucle.



5.4.2 Gestion des propositions

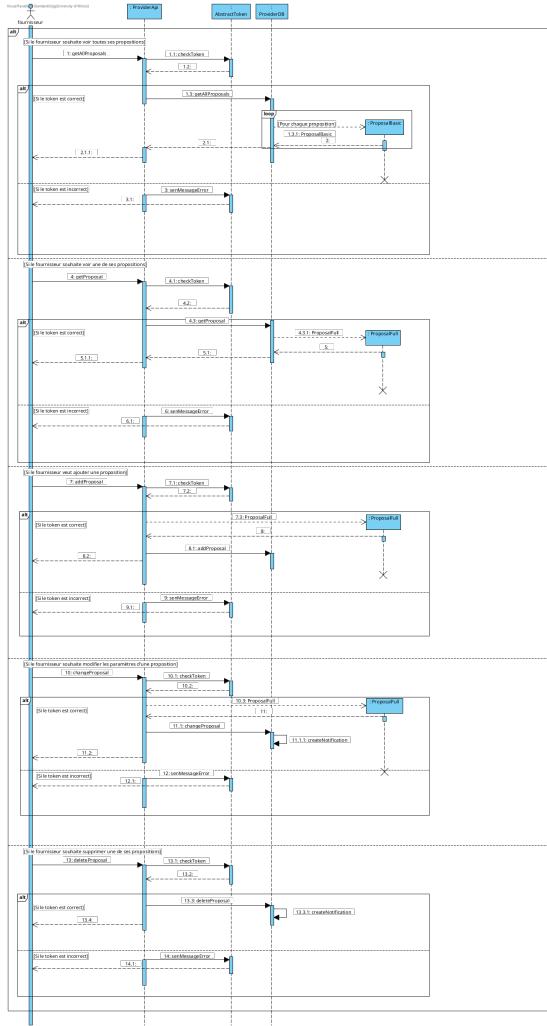
Avant de parler des diagrammes de séquence de cette partie, nous avons choisi de mettre les contrats d'un fournisseur et ses propositions dans les use cases parlant des contrats. De plus, seules les propositions seront abordées dans cette partie étant donné qu'un contrat sera commun au fournisseur et au client(section 5.3.1,page 39).

Pour commencer, nous allons regarder comment le fournisseur peut avoir accès à toutes ses propositions. Pour cela, il devra utiliser la méthode **getAllProposals**. De plus, une boucle permettra de récupérer tous les objets. Si le fournisseur veut avoir plus de précision sur une de ses propositions, alors il aura juste besoin d'appeler la méthode **getProposal** en ayant l'identifiant de la proposition au préalable.

Après ça, si le fournisseur veut ajouter une proposition, la méthode **addProposal** sera utilisée. Il faudra néanmoins créer un objet **ProposalFull** qui sera ensuite ajouté par la précédente méthode à la base de données.

Ensuite, si le fournisseur souhaite modifier les paramètres d'une de ses propositions, il n'aura qu'à faire un nouvel objet qui servira de "remplacant" à l'ancienne proposition. La méthode **changeProposal** sera appelée par la suite pour faire le changement des paramètres directement dans la base de données. A noter que modifier les paramètres d'une proposition revient à changer les contrats avec les clients qui ont pris cette proposition. De ce fait, une notification sera envoyée via la méthode **createNotification**.

Pour finir cette partie, nous allons parler de comment fonctionne la procédure pour supprimer une proposition. Nous avons juste à faire appel à la méthode **deleteProposal**. De manière similaire au fait de modifier, supprimer implique la suppression des contrats des clients ayant comme base cette proposition ce qui implique l'envoi d'une notification par la méthode **createNotification**.

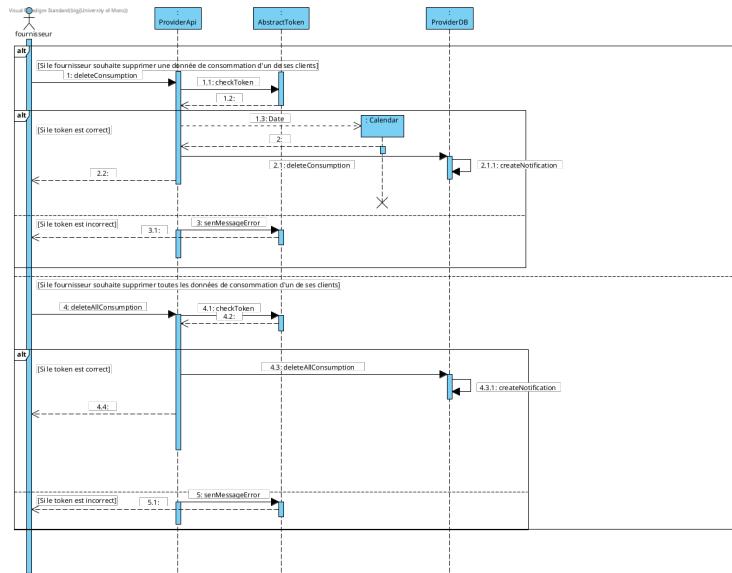


5.4.3 Gestion de la consommation

Cette sous-sous-section parlera de la gestion de la consommation d'un client par le fournisseur. A noter que le fait pour un fournisseur de voir la consommation de ses clients est la même procédure qu'un client qui va voir ses consommations à l'exception qu'un fournisseur ne peut voir la consommation créée par le client liée à son contrat.

Tout d'abord, le fournisseur souhaitant supprimer une donnée de consommation devra passer par la méthode **deleteConsumption** de l'API. De plus, il aura besoin d'un objet **Calendar** ne contenant que l'année, le mois et le jour. Après ça, il pourra supprimer la donnée qu'il souhaite avec la méthode portant le même nom que celle de l'API. Une notification sera envoyée au client avec la méthode **createNotification**.

Ensuite, si le fournisseur souhaite supprimer toutes les données de consommation d'un client. Il aura juste besoin de la méthode **deleteAllConsumption**. De même que le paragraphe du dessus, une notification sera créée.



6 Design REST Api

6.1 Introduction

Cette section représente le fonctionnement de l'API REST sous forme d'un diagramme de classes. Ce schéma montre les interactions entre l'API, le serveur et les utilisateurs. Les fonctionnalités implémentées dans le diagramme de classe de l'application sont reprises sous forme de requêtes liées à l'API. Une requête est référencée suivant un chemin appelée *route*. Celle-ci détermine les paramètres indiquant le chemin d'accès et les éléments à modifier. Tout ceci est contenu dans une requête de type "`https://www.myapplication.be/login/create_account/`". Pour toutes les requêtes (exceptée la requête de connexion), un paramètre *token* de type String est inclus pour vérifier si l'utilisateur est connecté et ce token permet de le reconnaître dans la base de données.

6.2 Gestion d'un utilisateur

Lorsqu'un utilisateur interagit avec l'application lors de sa connexion, une requête entrante est envoyée à l'API avec les informations de connexion (un mail, un mot de passe, un nom et un type⁶).

Une fois l'utilisateur créé, une requête sortante est renvoyée à l'application. La requête avec le code **201 - Created** est renvoyée. L'application ayant été mise à jour avec les nouvelles données passées en paramètres lors de la requête.

6.3 Ajout de données

Lors d'une interaction avec l'API, il est possible d'ajouter ou de créer des données grâce à une requête **POST**. Celle-ci permet notamment la création de contrats, de portefeuilles, de propositions de contrats pour les fournisseurs et l'ajout d'une méthode de consommation. Si l'interaction est validée, la requête **201 - Created** est renvoyée. Une fois la requête validée, la classe ayant appelé la requête est mise à jour avec les données passées en paramètre.

⁶Client ou fournisseur

6.4 Affichage de données

Parmi les fonctionnalités disponibles pour un utilisateur, la possibilité de voir une liste de certains objets lui appartenant en fait partie. Lors d'une requête **GET**, l'API va lister tous les éléments répondants à la requête et répondants aux critères mis en paramètres. Lors d'une requête **GET**, le dernier paramètre peut-être **NULL**. Si celui-ci est **NULL**, la requête renvoie tous les éléments recherchés. Dans le cas contraire, le paramètre est un id permettant de cibler la recherche.

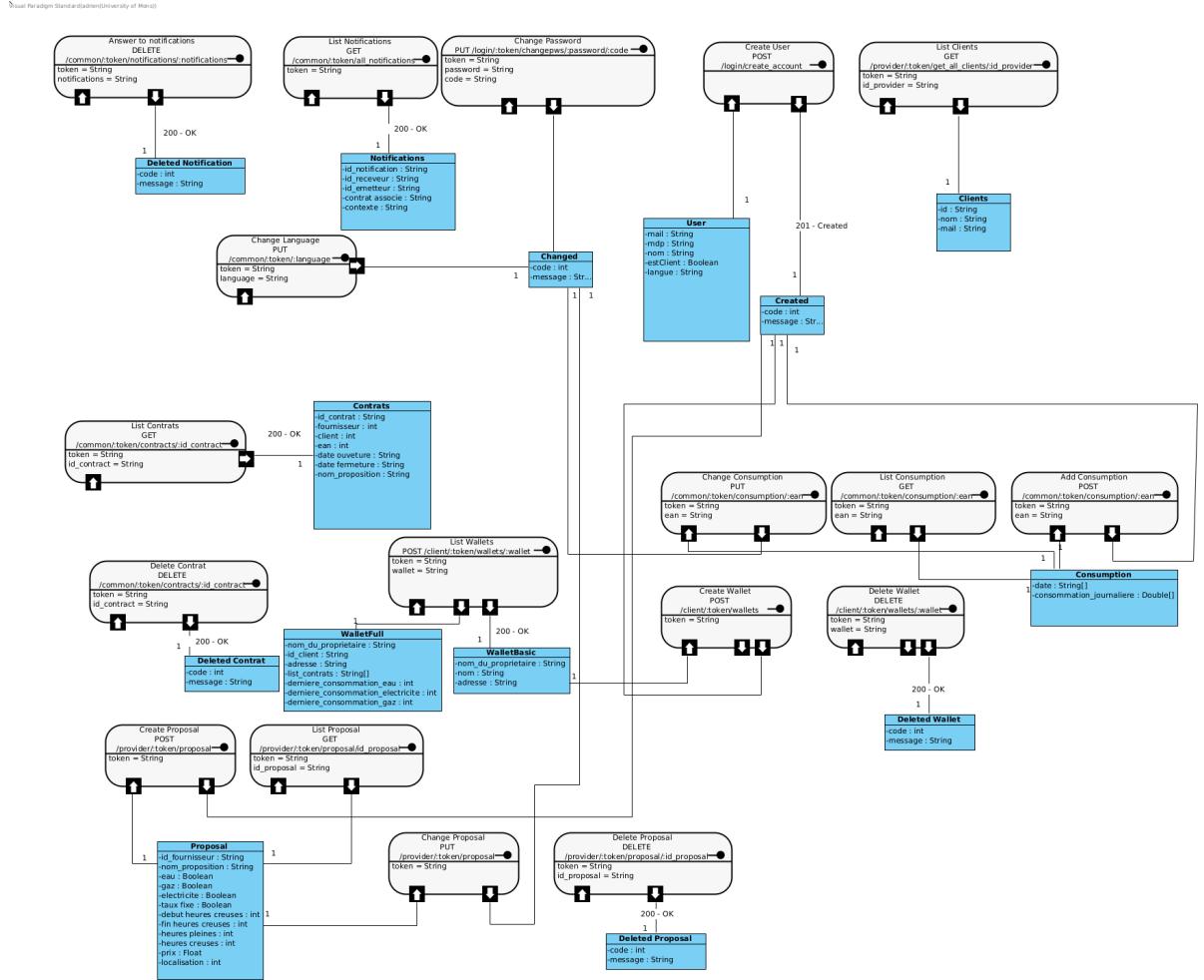
6.5 Modification/Suppression de données

Lorsqu'une modification de données est souhaitée, la classe contenant l'objet à modifier effectue une requête **PUT** correspondante. Si la modification est correctement effectuée, l'API renvoie un code **200 - OK** vers une classe *Changed*. L'élément à modifier est contenu dans la requête sous forme de paramètre. Quant il s'agit d'une suppression de données via une requête **DELETE**, le dernier servira à cibler l'élément à supprimer et ne pourra pas être **NULL** sous peine de déclencher une erreur **404 - Not Found**.

6.6 Conclusion

En conclusion, un utilisateur peut gérer ses données à l'aide de requêtes **POST**, **PUT**, **GET**, **DELETE**. Ces appels de requêtes sont faites par les classes gestionnaires des objets qu'elles contiennent. Si les critères ne sont pas remplis ou sont invalides, une erreur est renvoyée en fonction du cas.

6.7 Schéma



7 Interface

7.1 Introduction

Nous avons réalisé notre interface principalement sur base de nos usecases et overview diagrams ainsi que les choix effectués concernant la base de données et le class diagram.

Cette manière de procéder nous a permis de structurer nos idées afin de fournir l'interface la plus intuitive possible.

De plus, nous avons décidé de séparer l'interface en **trois parties** :

1. Le système de logs
2. L'interface client
3. L'interface fournisseur

ce qui nous semblait plus approprié étant donné que nous devons obtenir deux applications distinctes, une pour le fournisseur et l'autre pour le client.

Il est important de noter que nous avons utilisé le logiciel "Figma" pour réaliser ces maquettes.

7.2 Système de logs

Sur la page principale de connexion, l'utilisateur devra préciser s'il est client ou fournisseur en cochant la case appropriée et pourra également changer de langue s'il le souhaite.

Il pourra dès lors entrer ses informations et **se connecter**.

Cependant s'il a **oublié son mot de passe**, un bouton sera prévu à cet effet. Ce dernier enverra l'utilisateur sur une nouvelle page où il devra inscrire le code qu'il aura reçu par mail et son nouveau mot de passe.

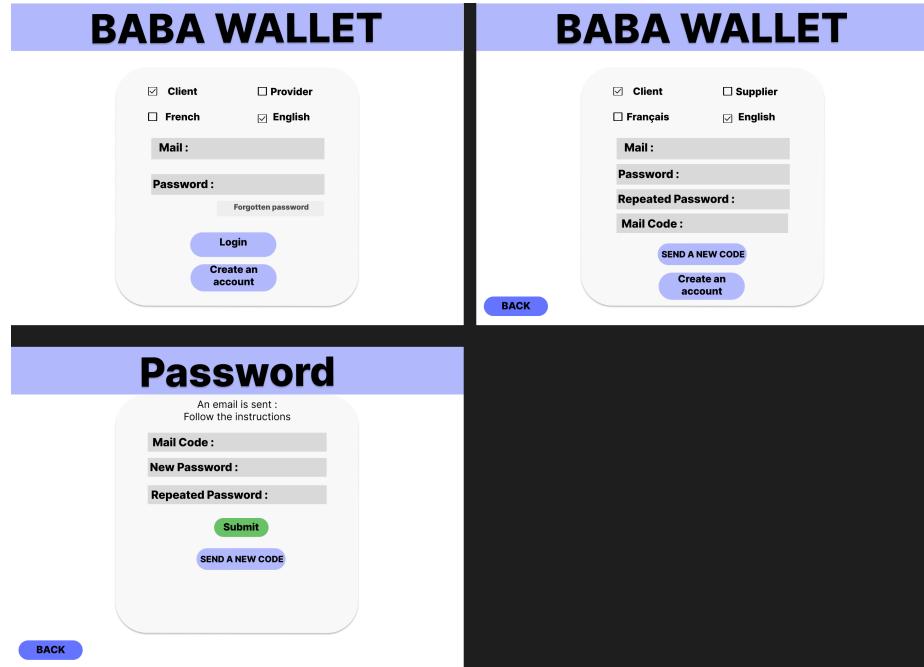
Un bouton "Submit" aura pour but de confirmer ce que l'utilisateur a entré et un bouton "Back" sera présent si jamais ce dernier veut retourner en arrière.

S'il n'a pas encore de compte, l'utilisateur aura la possibilité d'en **créer un nouveau**. Le système fonctionnera comme la connexion expliquée précédemment.

La seule différence réside lorsque ce dernier appuiera sur le bouton "Create an account". L'utilisateur sera alors envoyé sur une nouvelle page où on lui demandera de confirmer la création de son compte en entrant le code qu'il aura reçu par mail pour plus de sécurité.

Pour finir, de la même manière que pour le changement de mot de passe, un bouton "Finish" aura pour but de confirmer ce que l'utilisateur a entré et un bouton "Back" sera présent si jamais ce dernier veut retourner en arrière.

En outre, l'utilisateur pourra demander qu'on lui renvoie un mail si le précédent n'est pas reçu.



7.3 Points communs

Les points communs entre l'interface client et fournisseur sont

1. Les notifications :

L'utilisateur verra le fournisseur ou client en question, le contexte et le contrat lié à la notification.

Il pourra également observer plus en détail le contrat avec le bouton "See", marquer comme lues les notifications grâce au bouton "OK" et les accepter ou refuser dans le cadre d'une demande.

En outre, la possibilité d'actualiser lui-même s'il le désire lui est laissée.

2. Les paramètres :

Concernant le changement de mot de passe, cela se déroule exactement de la même manière que dans le système de logs. Pour la langue, l'utilisateur pourra en ajouter, choisir sa préférée et changer celle étant active.

Dans tous les cas, ce dernier sera conduit sur une page où il aura la possibilité de chercher une langue et ensuite soit de la télécharger, soit de la choisir en fonction du contexte.

7.4 Interface client

Lorsqu'un client se connectera, il arrivera sur la page d'accueil ou plus précisément "Home".

Ce dernier aura alors les possibilités suivantes :

1. Voir ses portefeuilles ("Wallets")
2. Voir ses contrats ("Your contracts")
3. Voir ses notifications
4. Voir les fournisseurs et contrats relatifs ("See new contracts")
5. Se déconnecter
6. Aller sur la page des paramètres

Il est important de noter que ce n'est qu'à partir de ces six pages que le client pourra revenir sur la page "Home" étant donné le choix effectué dans notre overview diagram.

Néanmoins, ce dernier aura la possibilité de revenir en arrière à l'aide du bouton "Back" lorsqu'il sera sur des "sous-pages" de ces six sections principales.

Nous pouvons maintenant commencer par l'option **"Voir ses portefeuilles"** (**"Wallets"**) :

Le client pourra à partir d'ici voir une liste reprenant tous les portefeuilles qu'il aura créés précédemment.

Chaque rectangle représentant un portefeuille comprend :

1. Le nom du portefeuille
2. Le nom du propriétaire du portefeuille
3. L'adresse associée au portefeuille

Le bouton **"Go"** permettra de voir plus en détail un portefeuille et le bouton **"+"** sur le rectangle vide donnera la possibilité au client d'ajouter un nouveau portefeuille.

Lorsqu'il souhaitera **voir plus en détail un portefeuille**, le client aura accès au nom du portefeuille, au nom du propriétaire du portefeuille et à l'adresse associée au portefeuille comme sur la page précédente.

Toutefois d'autres actions lui seront possibles :

1. Voir ses dernières consommations
2. Voir les contrats associés
3. Voir les contrats associés en détail (Bouton **"Go"**)
4. Ajouter des consommations (Bouton **"Add consumptions"**)
5. Fermer définitivement le portefeuille (Bouton **"Close the Wallet"**)

Le bouton **"Go"** concernant les contrats associés l'amènera sur une page montrant le contrat que nous expliquerons prochainement.

Le bouton **"Add consumptions"** l'entraînera vers une page où il pourra voir ses données de consommations en graphique ou en tableau à l'aide des deux boutons placés en haut de la page.

Le rectangle gris les représentent et il pourra également sélectionner dans ce dernier s'il souhaite voir les données mensuellement, hebdomadairement,...

Le client pourra également s'il le désire exporter ses données. Nous pouvons noter que ce dernier exportera les données présentes dans le tableau gris au moment où il appuiera sur le bouton **"Export"**.

En ce qui concerne le changement et l'ajout de consommation, celui-ci devra entrer la date correspondante, le type d'énergie et la valeur associée et devra ensuite confirmer ses modifications grâce aux boutons **"Change"** et **"Add"**.

Le bouton ”+” mentionné précédemment servant à ajouter un nouveau portefeuille est connecté à une page qui lui demandera d'écrire le nom et l'adresse du nouveau portefeuille.

Cette manière de procéder est liée au fait que nous avons décidé qu'un portefeuille doit être créé avant d'associer un contrat à ce dernier.

La deuxième option est la suivante : **”Voir ses contrats”** (**”Your contracts”**)

Une fois sur cette page, le client aura accès à sa liste de contrats et il pourra rechercher un contrat spécifique en tapant le code ”EAN” ou le nom du fournisseur.

Les informations basiques qu'il verra sur chaque ligne de la liste sont le fournisseur et le type d'énergie.

Lorsqu'il cliquera sur le **bouton ”Go”** la nouvelle page lui offrira les informations suivantes : le type d'énergie et le fournisseur de la même manière que sur la page précédente.

Les nouvelles informations disponibles sont les suivantes :

1. Le nom du contrat
2. Le portefeuille associé (Le code ”EAN” ainsi que l'adresse)
3. La localisation (du contrat)
4. Le prix basique
5. Le prix dépendant du jour et la nuit
6. Le taux fixe ou variable
7. Les heures creuses
8. La date d'ouverture du contrat
9. La date de fermeture du contrat

Il pourra aussi fermer le contrat par le bouton ”Close the contract”.

La troisième et dernière section est "**Voir les fournisseurs et contrats relatifs ("See new contracts")**"

Sur cette page, le client pourra voir une liste reprenant les noms des fournisseurs, les types d'énergie et la localisation pour chaque contrat.

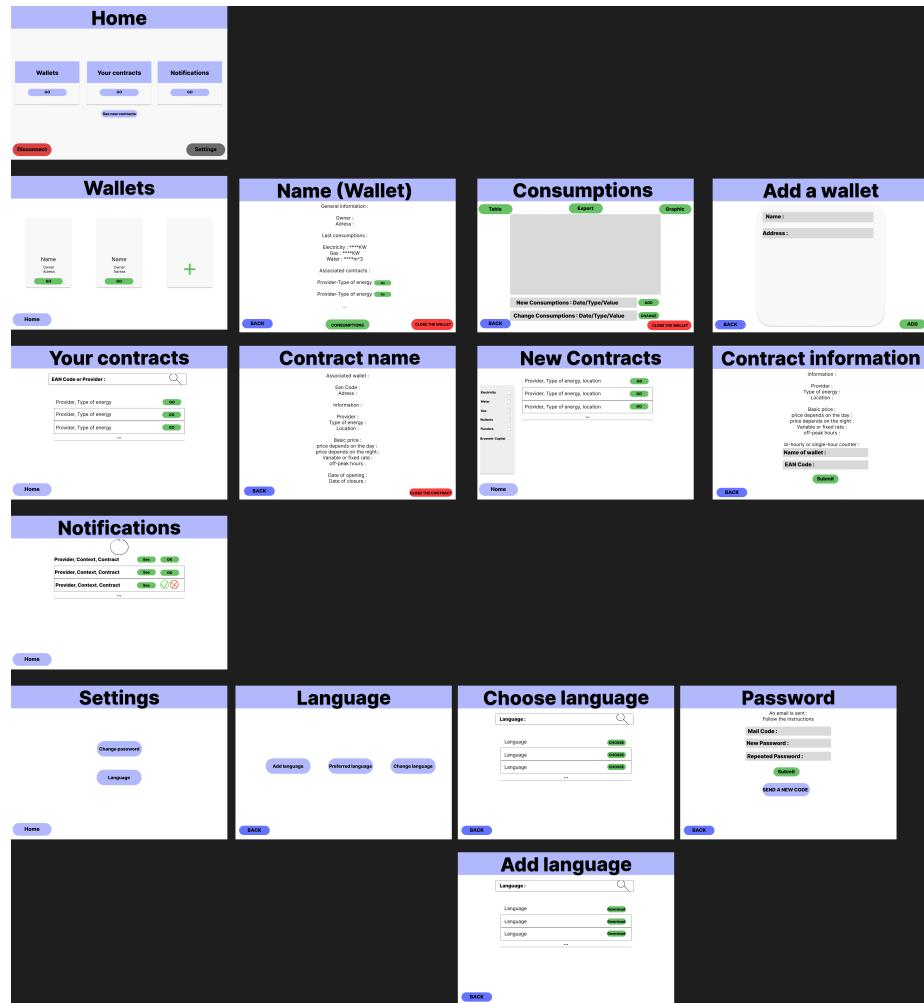
Pour lui rendre la tâche de recherche plus facile, nous avons mis à sa disposition une sélection sur le côté droit. Cette dernière permettra de choisir le type d'énergie souhaité (Electricité, gaz et eau) ainsi que la localisation.

Le **bouton "Go"** l'amènera sur les informations liées à un nouveau contrat :

1. Le fournisseur
2. Le type d'énergie
3. La localisation (du contrat)
4. Le prix basique
5. Le prix dépendant du jour et la nuit
6. Le taux fixe ou variable
7. Les heures creuses
8. Le type de compteur

Pour confirmer sa demande de contrat, le client devra entrer le nom du portefeuille auquel il souhaite l'associer ainsi que le code "EAN" et appuyer sur le bouton "Submit".

Nous ne reviendrons pas sur les points "Voir les notifications" et "Les paramètres", ces derniers étant expliqués dans les points communs.



7.5 Interface fournisseur

Lorsqu'un fournisseur se connectera, il arrivera sur la page d'accueil ou plus précisément "Home".

Ce dernier aura alors les possibilités suivantes :

1. Voir ses clients ("Your clients")
2. Voir ses contrats ("Your contracts")
3. Voir ses notifications
4. Se déconnecter
5. Aller sur la page des paramètres

Tout comme pour l'interface client, il est important de noter que ce n'est qu'à partir de ces six pages que le fournisseur pourra revenir sur la page "Home" étant donné le choix effectué dans notre overview diagram.

Néanmoins, ce dernier aura la possibilité de revenir en arrière à l'aide du bouton "Back" lorsqu'il sera sur des "sous-pages" de ces six sections principales.

La première section est **Voir ses clients ("Your clients")**

Le fournisseur pourra à partir d'ici voir une liste reprenant tous les noms de ses clients ainsi que leurs adresses mails.

Le bouton "Go" lui permettra de voir plus en détail un client, il aura accès à :

1. Son nom
2. Son adresse mail
3. Les contrats lui étant associés (Nom du contrat, code "EAN", le type d'énergie, la dernière consommation) :
Le bouton "Go" lui donnera la possibilité de voir plus en détail le contrat de la même manière que lorsqu'un client souhaite voir ses contrats.
Il aura la faculté de supprimer un contrat et voir les consommations.
Contrairement au client, il ne pourra pas ajouter de nouvelles données de consommations mais simplement les modifier et les supprimer et ne pourra pas exporter les données mais en importer.
4. La suppression d'un client (Bouton "Delete Client")

Le bouton "Add clients" le mènera sur une page où il pourra sélectionner un contrat à envoyer à un possible nouveau client.

Passons à la deuxième section : **Voir ses contrats ("Your contracts")**

Le fournisseur pourra à partir d'ici voir une liste reprenant tous les noms de ses contrats, leurs types d'énergie ainsi que leurs localisations.

Le bouton "Go" lui donnera la possibilité de voir plus en détails ses contrats avec :

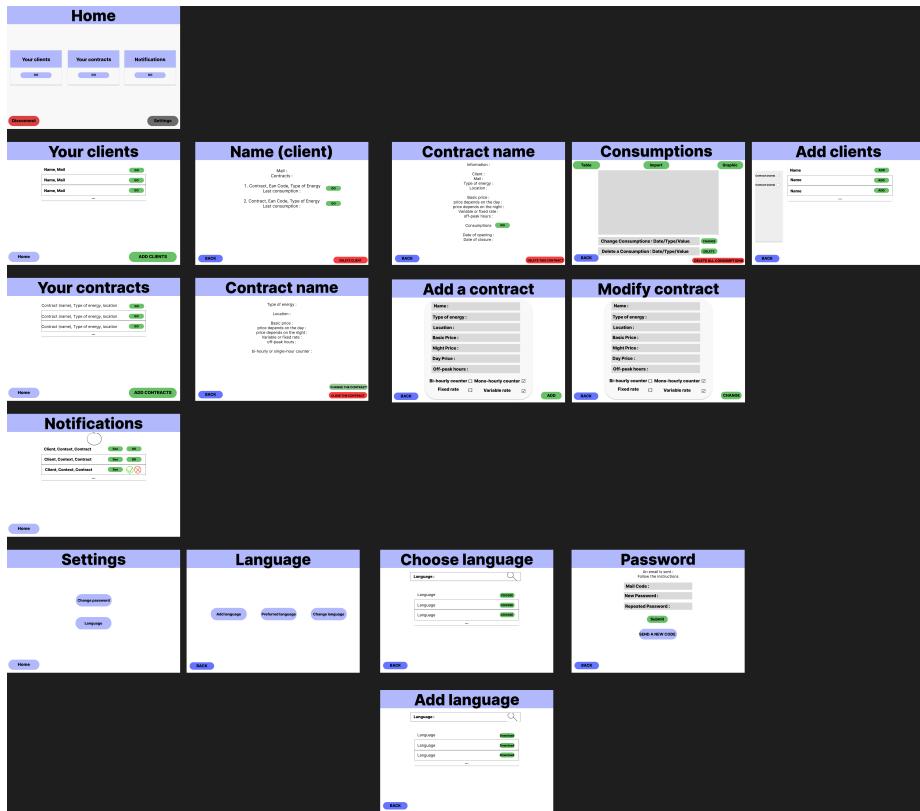
1. Le nom du contrat
2. Le type d'énergie
3. La localisation (du contrat)
4. Le prix basique
5. Le prix dépendant du jour et la nuit
6. Le taux fixe ou variable
7. Les heures creuses
8. Le type de compteur

Ce dernier aura également accès à la suppression du contrat et au changement du contrat fonctionnant comme **l'ajout de contrats (Bouton "Add contracts")** que nous allons expliquer ci-dessous.

Lorsqu'un fournisseur souhaitera ajouter un contrat il devra entrer :

1. Le nom du contrat
2. Le type d'énergie
3. La localisation (du contrat)
4. Le prix basique
5. Le prix dépendant du jour et la nuit
6. Les heures creuses
7. Le taux fixe ou variable
8. Le type de compteur

Le bouton "Add" lui servira à valider la création du nouveau contrat.



8 Introduction-extension

Pour plus de clarté, nous avons décidé de proposer un schéma de la base de données et de l'API pour les fonctionnalités de base.

Ensuite, nous expliquerons comment ces schémas seront étendus respectivement pour chaque extension.

Notez donc que lors de l'implémentation, nous rassemblerons tout en une seule base de données et une seule API.

9 Extension 1.6.1 : Gestion d'utilisateurs par Claire D'Haene

Cette extension concerne **uniquement** les clients, de ce fait, je ne vais pas réafficher les diagrammes des fournisseurs pour plus de clarté.

Il me semblait également approprié de reprendre les diagrammes de la base du projet et d'y rajouter les cas nécessaires à cette extension.

10 Use case diagram

10.1 Introduction

La plus grande différence réside dans **les accès aux portefeuilles**.

Pour cette raison, j'ai choisi de **séparer** la partie « Voir les portefeuilles » où il s'agit de portefeuilles appartenant entièrement au client « **gestionnaire** », de la partie « voir les portefeuilles où il est invité » pour permettre une meilleure visibilité des cas.

10.2 Client

Lorsque le client ou plus précisément, le gestionnaire du portefeuille est sur la section « Voir les portefeuilles », il pourra créer ou fermer un portefeuille et voir les données de consommation du portefeuille comme sur la base du projet.

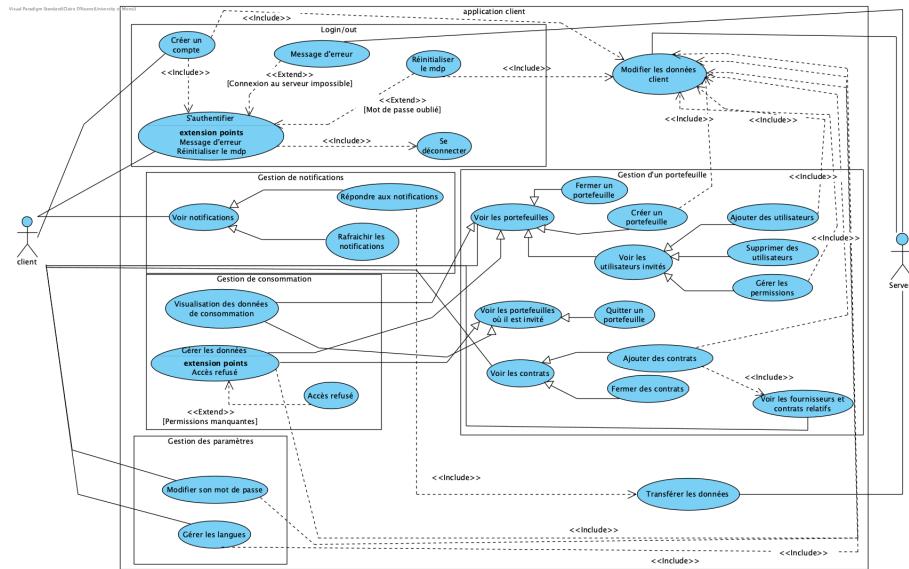
Ce dernier sera également apte à gérer ces données et à voir les utilisateurs invités sur ses portefeuilles étant donné son « grade » de gestionnaire.

A partir de cette option, le gestionnaire aura la capacité d'ajouter, supprimer des utilisateurs ou gérer les permissions de ces derniers.

Une fois sur la section « Voir les portefeuilles où il est invité », le client aura accès à la visualisation des données en lecture seule et à cette même option ainsi qu'à gérer les données en lecture et écriture. S'il n'a pas les permissions requises pour gérer les données, l'accès sera **refusé**.

Celui-ci aura aussi la possibilité de quitter un portefeuille lorsqu'il le souhaite.

10.3 Diagramme



11 Interaction overview diagram

11.1 Introduction

La réalisation de mon interaction overview diagram s'est faite sur base de mon use case diagram.

Ce qui m'a permis encore une fois, de détailler la structure de ce diagramme intuitivement de la même manière que nous l'avions fait pour la base du projet.

Nous allons donc repasser ensemble sur le nouveau parcours s'offrant au client.

Notamment pour la gestion de portefeuilles et de consommations suite aux nouvelles permissions et possibilités lui étant octroyées.

11.2 Accès à l'application pour les clients

Lorsqu'un client « gestionnaire » voudra voir ses portefeuilles à partir du menu, il retrouvera plusieurs fonctionnalités :

1. Créer portefeuille
2. Fermer un portefeuille
3. Visualisation des données de consommation
4. Gérer les données
5. Voir les utilisateurs invités

Les quatre premières options étant identiques à la base, ne repassons pas sur ces dernières.

Cependant, nous pouvons noter que l'accès à l'option « gérer les données » ne sera **jamais refusée** puisque le client est « gestionnaire » et maître de ses portefeuilles dans ce cas.

- Voir les utilisateurs invités :

- a) Ajouter des utilisateurs
- b) Supprimer des utilisateurs
- c) Gérer les permissions

- Ajouter des utilisateurs :

Le client « gestionnaire » se retrouvera alors sur une page où il pourra rechercher l'utilisateur qu'il souhaite inviter et ainsi, il pourra l'ajouter en inscrivant les permissions qu'il désire lui associer (lecture ou lecture et écriture).

- Supprimer des utilisateurs :

Cela lui donnera la possibilité de supprimer un utilisateur invité s'il trouve cela plus judicieux.

- Gérer les permissions :

Le gestionnaire aura la capacité de permettre une lecture seule ou une lecture et écriture à l'utilisateur invité.

La lecture seule comme son nom l'indique, permettra à ce dernier une « visualisation des données de consommation » et **la lecture et écriture** autorisera en plus de cela, de « gérer les données ».

L'ajout et la suppression d'utilisateurs ainsi que la gestion des permissions concernant les utilisateurs entraîneront **une notification** sur l'application des autres clients concernés.

Le menu offre désormais également la possibilité au client de **voir les portefeuilles où il est invité**, le client sera donc en mesure de :

1. Voir des données de consommation
2. Gérer les données
3. Quitter un portefeuille

- **Visualisation des données de consommation :**

En cas de lecture seule, le client pourra voir les données de consommation et donc garder un œil sur la gestion du portefeuille.

- **Gérer les données :**

En cas de lecture et écriture, ce dernier aura la capacité de voir les données de consommation d'un portefeuille et de les modifier.

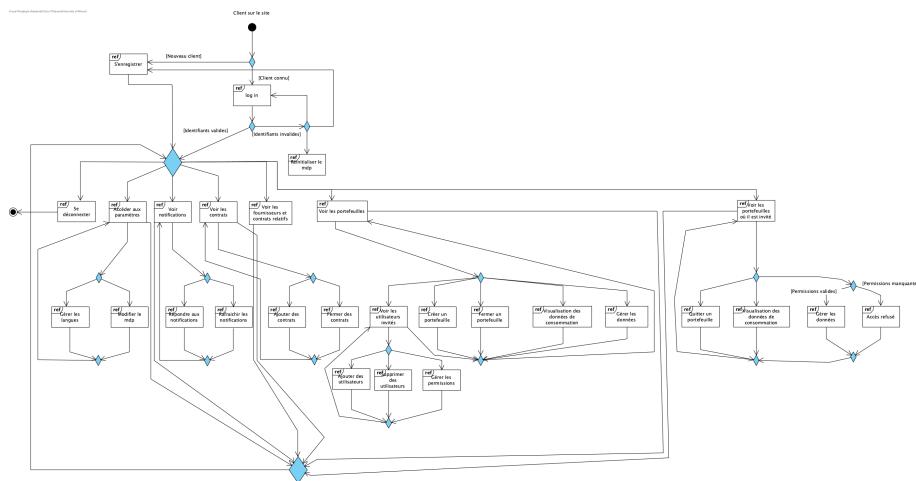
Si les **permissions** du client ne sont **pas valides**, autrement dit, si le client n'a pas accès à la lecture et à l'écriture dans le portefeuille, alors, l'**accès** lui sera **refusé**.

- **Quitter un portefeuille :**

Cela lui donnera la possibilité de quitter un portefeuille s'il trouve cela plus approprié.

Ce qui entraînera une **notification** du côté du client <**gestionnaire**>.

11.3 Diagramme



12 Entity relationship diagram

12.1 Introduction

Dans cette partie, je vais vous expliquer les ajouts apportés à la base de données de la base du projet.

De ce fait, il me semblait plus judicieux de ne garder dans ce rapport que les sections jouant un rôle primordial dans la base de données pour l'extension.

Vous pourrez donc y retrouver le fonctionnement de certains attributs ainsi que les clés primaires et étrangères.

Comme vous le remarquerez, le choix de séparer "portefeuilles" de "portefeuilles invités" joue encore un rôle important dans la manière de réaliser ce diagramme.

12.2 Utilisateur

De la même manière que pour la base du projet, partons du principe qu'un fournisseur, un client et un client "propriétaire" (gestionnaire) sont des utilisateurs de l'application.

De ce fait, utilisons donc l'identifiant "id".

On aura donc :

1. PK(utilisateur) = id
2. FK(utilisateur) REFS fournisseur = id
3. FK(utilisateur) REFS client = id
4. FK(utilisateur) REFS client_proprietaire = id

12.3 Portefeuille invité

Vous remarquerez que ce qui le différencie principalement du "portefeuille" sont les attributs suivants:

1. **les permissions :**

Les permissions sont donc celles attribuées à un client (invité), c'est à dire, "lecture" ou "lecture et écriture".

Ce qui explique donc la présence du binary(1) représentant les deux possibilités.

2. **l'id du propriétaire :**

Son ajout est lié au fait qu'il s'agit d'un élément non négligeable.

Le but étant de le récupérer pour obtenir la totalité des informations essentielles.

En outre, son id est différent de l'id client.

Il est important de noter que l'id client dans ce cas est en fait le client invité ayant la possibilité de lire ou de lire et écrire dans les portefeuilles invités et d'être donc gestionnaire dans la partie portefeuille.

On a donc en terme de clés :

1. PK(portefeuille) = adresse et id_client :

La table va stocker toutes les invitations de tous les portefeuilles et il y aura donc plusieurs invités pour une adresse.

2. FK(portefeuille) REFS client = id_client

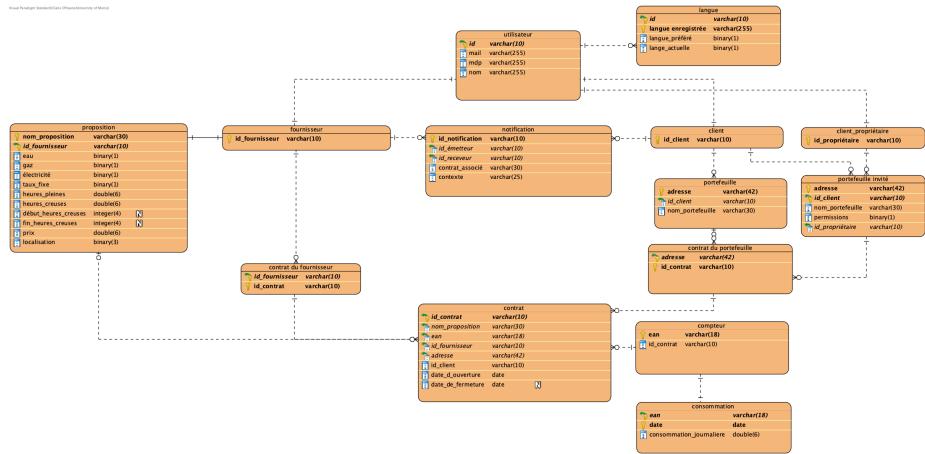
3. FK(portefeuille) REFS clientPropriétaire = id_Propriétaire

12.4 Contrat du portefeuille

Cette section ne sera pas affectée, la seule différence avec la base réside dans le fait que cette dernière est également liée aux portefeuilles invités.

En effet, tout comme les "portefeuilles", les "portefeuilles invités" doivent être aussi liés aux contrats.

12.5 Diagramme



13 Class diagram

13.1 Introduction

Cette extension ne change en rien la logique de base que nous avons construite dans la base du projet.

En effet, les packages :

1. Api
2. Database
3. DataObject
4. App

sont conservés avec la même logique.

Repassons donc sur les quelques classes et méthodes essentielles rajoutées pour la conception de l'extension.

13.2 Le package api

Dans ce package, vous pouvez trouver l'ajout des méthodes liées aux clients dans **"ClientApi"** en prenant en compte les nouvelles possibilités liées aux clients invités, aux portefeuilles invités ainsi qu'aux permissions étant accordées.

Cette classe sera toujours utilisée pour les requêtes liées aux clients lorsqu'ils seront connectés.

Ils pourront effectuer les opérations qu'ils souhaitent au niveau des :

1. portefeuilles
2. **portefeuilles invités**

et auront également la possibilité de voir tous leurs contrats ainsi que les propositions des fournisseurs.

13.3 Le package DataBase

Ce dernier contient toutes les méthodes qui vont communiquer avec la base de données de la base du projet ainsi que celles ajoutées précédemment pour l'extension.

Comme pour la base du projet, vous pouvez noter que celles-ci ont exactement le même nom que dans le package API puisqu'elles sont contenues dans la suite du programme.

Vous pouvez donc observer l'ajout des méthodes dans la classe "**ClientDB**".

13.4 Le package dataObject

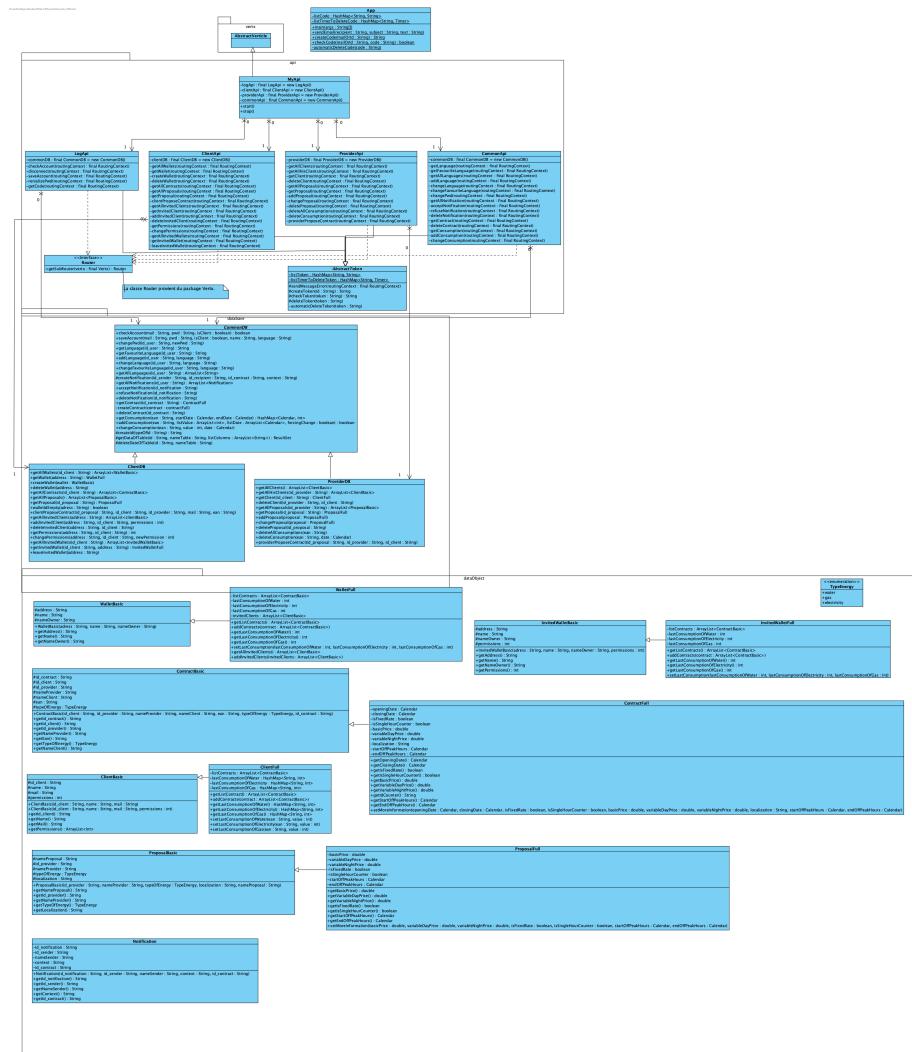
Ce package reprend les classes représentant un objet, c'est-à-dire pour l'extension :

1. Les clients
2. Les portefeuilles
3. **Les portefeuilles invités**
4. Les propositions
5. Les notifications

Vous pouvez remarquer l'ajout des permissions dans la classe "**ClientBasic**" et la gestion des clients invités dans "**WalletFull**".

En outre, la création des classes "**InvitedWalletBasic**" et "**InvitedWalletFull**" permettent de respecter les choix effectués concernant le fait de séparer les portefeuilles "classiques" des portefeuilles invités.

13.5 Diagramme



14 Sequence diagram

14.1 Introduction

Tout d'abord, il était important de revenir sur la partie client du sequence diagram de la base du projet étant donné les méthodes et Use Cases ayant été ajoutés pour cette extension.

Comme pour la base du projet, repartons de mon class diagram et de mon use case diagram et représentons l'interaction des méthodes avec l'API et la base de données.

Avant de commencer, il est important de signaler que les explications concernant le token dans le class diagram de la base du projet sont aussi valables pour l'extension afin d'éviter les répétitions.

De plus, vous pouvez remarquer l'appel à la méthode "createNotification" lorsque cela est nécessaire dans les deux sequence diagrams ci-joints.

En outre, pour permettre une meilleure compréhension et lisibilité, nous repasserons ensemble uniquement sur les Use Cases et méthodes ajoutés.

14.2 Client

14.2.1 Gestion des portefeuilles invités

Le premier ajout est en partant de l'Use Case **"Voir les portefeuilles où il est invité"**. A cet effet, il y aura deux méthodes :

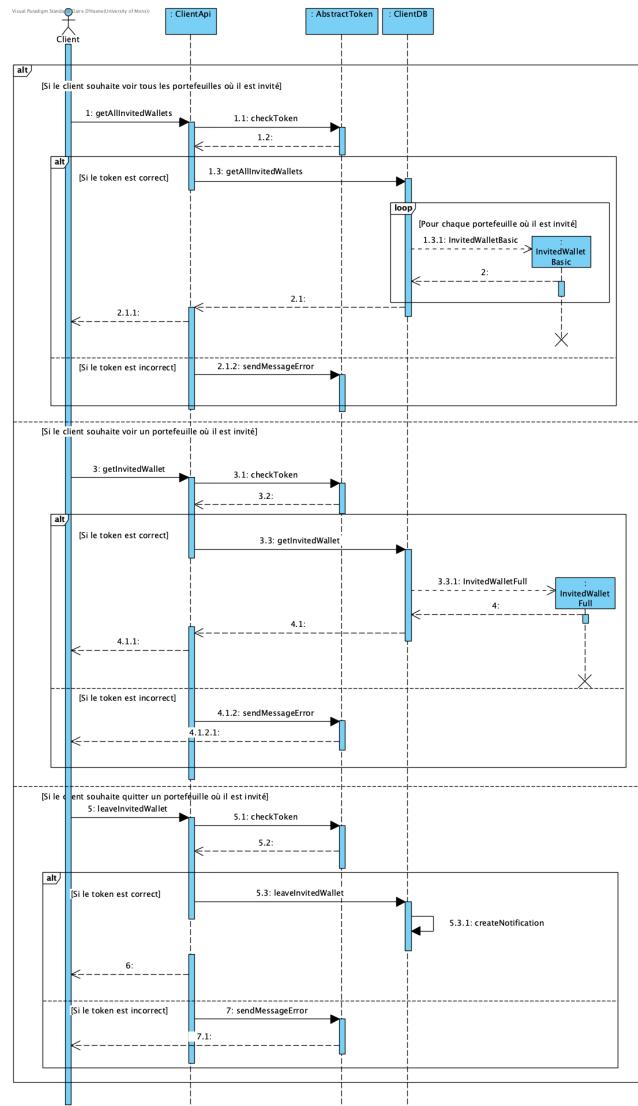
1. getAllInvitedWallets si le client souhaite voir tous les portefeuilles où il est invité.
2. getInvitedWallet, si le client souhaite voir un portefeuille où il est invité en particulier.

Ces deux méthodes sont appelées une fois de clientApi et ensuite, une fois de ClientDB.

Dans le cas de "getAllInvitedWallets", la valeur de retour est une ArrayList de "InvitedWalletBasic" donc j'utiliserai une boucle et dans le cas de "getInvitedWallet", je dois simplement retourner une instance de "InvitedWalletFull".

Concernant le Use Case **"Quitter un portefeuille où il est invité"**, j'appelle la méthode "leaveInvitedWallet" de clientApi et ensuite, la méthode du même nom de ClientDB.

Il est important de remarquer que pour quitter un portefeuille où il est invité, il n'y a pas de condition particulière, contrairement à lorsqu'un client souhaite supprimer un portefeuille étant donné que ce dernier sera gestionnaire dans ce cas.



14.2.2 Gestion des utilisateurs invités

Le deuxième ajout est en partant de l'Use case "**Voir les utilisateurs invités**". Ce dernier fonctionne de la même manière qu'expliqué précédemment pour "Voir les portefeuilles où il est invité".

En effet, il y aura la méthode :

1. getAllInvitedClients si le client souhaite voir tous les utilisateurs invités.
Je dois donc retourner une ArrayList de "ClientBasic".
2. mais getInvitedClient, si le client souhaite voir un utilisateur invité en particulier, ne devra pas être instanciée car toutes les informations dont un client "gestionnaire" à besoin se trouvent dans un ClientBasic envoyé au préalable par la méthode getAllInvitedClients.

Passons maintenant aux Use Cases "**Ajouter un utilisateur**" et "**Supprimer un utilisateur**".

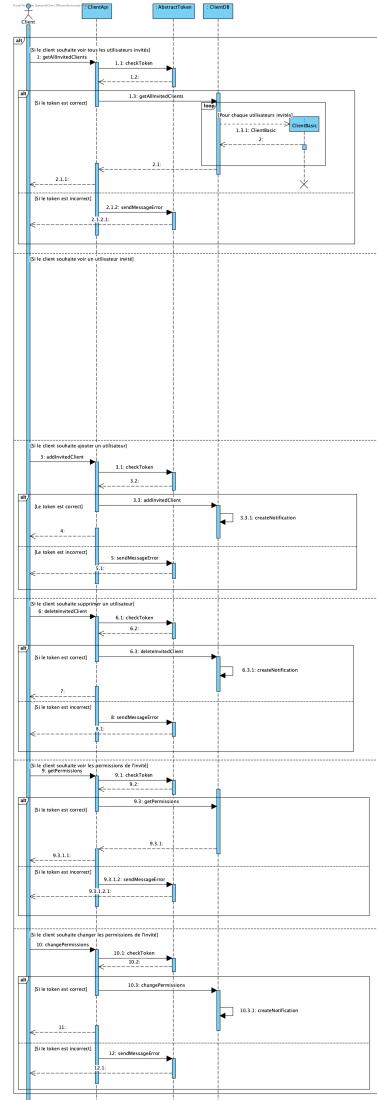
Pour **ajouter un utilisateur**, j'appelle donc la méthode "addInvitedClient" de clientApi et ensuite, la méthode du même nom de ClientDB.

Pour **supprimer un utilisateur**, je procède de la même manière que pour ajouter un utilisateur avec la méthode "deleteInvitedClient".

Vous pouvez remarquer que ces deux méthodes ne doivent rien retourner.

Au sujet de l'Use Case "**Gérer les permissions**", si le client souhaite voir les permissions d'un invité, je ferai appel à la méthode "getPermissions" qui me retournera les permissions et s'il souhaite changer les permissions d'un invité, j'utiliserai la méthode "changePermissions".

Ces deux méthodes sont d'abord appelées depuis clientApi et ensuite, de ClientDB.



15 API REST

15.1 Introduction

Dans cette partie, je vais vous expliquer les ajouts apportés à l'API-REST de la base du projet.

De ce fait, il me semblait plus judicieux de garder dans ce rapport que les nouvelles sections concernant mon extension.

15.2 Les utilisateurs invités

Vous pouvez remarquer l'ajout des "Rest Services" suivants :

1. **List InvitedClients :**

Le GET permet de renvoyer les utilisateurs invités sur un portefeuille.

2. **Add InvitedClient :**

Le POST permet d'ajouter un nouvel utilisateur à un portefeuille.

3. **Delete InvitedClient :**

Le DELETE permet de supprimer un utilisateur invité sur un portefeuille.

4. **Change Permissions :**

Le PUT permet de modifier les permissions d'un utilisateur invité.

5. **Get Permissions :**

Le GET permet de renvoyer les permissions d'un utilisateur invité.

15.3 Les portefeuilles invités

Vous pouvez également remarquer l'ajout des "Rest Services" suivants :

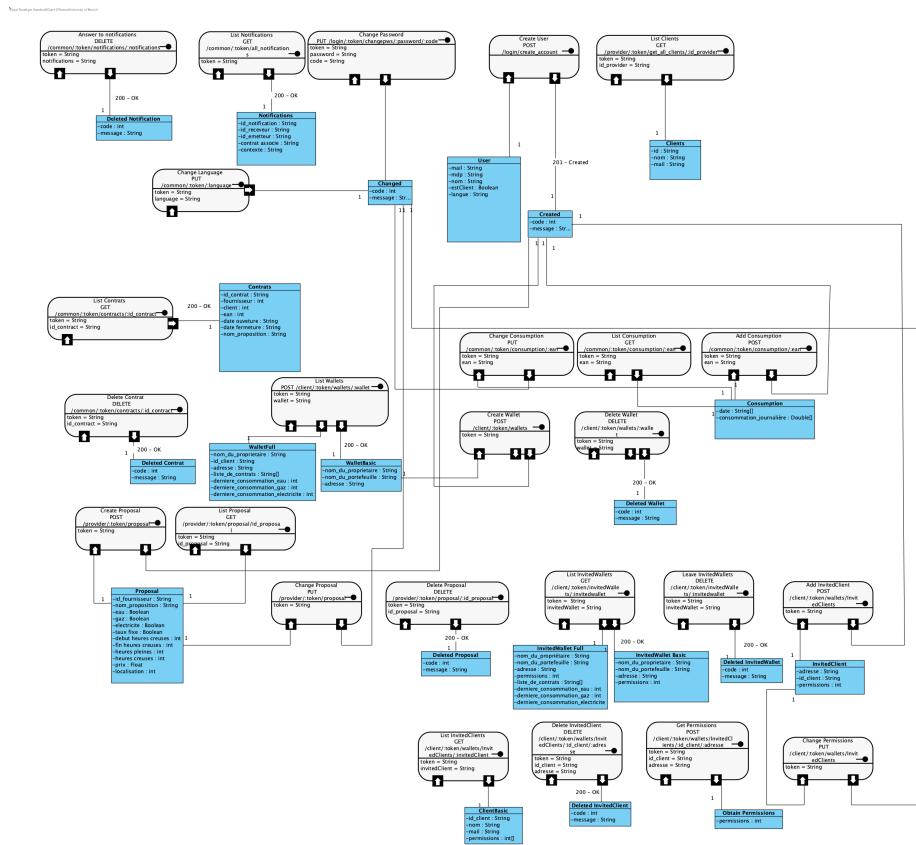
1. **List InvitedWallets :**

Le GET donne la possibilité de renvoyer les portefeuilles invités que possède un client.

2. **Leave InvitedWallets :**

Le DELETE donne la possibilité au client de supprimer un portefeuille où il est invité.

15.4 Diagramme



16 Interface

16.1 Introduction

J'ai réalisé l'interface principalement sur base de l'usecase et overview diagram ainsi que les choix effectués concernant la base de données et le class diagram.

Cette manière de procéder m'a permis de structurer mes idées afin de fournir l'interface la plus intuitive possible.

De plus, il me semblait plus judicieux de n'inclure que la partie qui diffère de l'interface de base afin de permettre une meilleure lisibilité.

Les **parties** suivantes seront donc **conservées** même si elles ne sont pas affichées dans le pdf lié :

1. Le système de logs
2. L'interface fournisseur
3. La partie "Contrats" de l'interface client.
4. La partie "Notifications" de l'interface client.
5. La partie "Paramètres" de l'interface client.

Les **nouvelles parties** pour l'interface client sont donc essentiellement les suivantes :

1. Les portefeuilles "invités"
2. Les clients invités

Il est important de noter que le logiciel utilisé pour réaliser cette maquette est "Figma".

16.2 L'interface client

Lorsqu'un client se connectera, il arrivera sur la page d'accueil ou plus précisément "Home" comme sur l'interface de base.

Ce dernier aura alors les mêmes possibilités ainsi qu'une nouvelle :

1. Voir ses portefeuilles ("Wallets")
2. Voir les portefeuilles où il est invité ("Invited wallets")
3. Voir ses contrats ("Your contracts")
4. Voir ses notifications
5. Voir les fournisseurs et contrats relatifs ("See new contracts")
6. Se déconnecter
7. Aller sur la page des paramètres

Il est important de noter que ce n'est qu'à partir de ces sept pages que le client pourra revenir sur la page "Home" étant donné le choix effectué dans mon overview diagram.

Néanmoins, ce dernier aura la possibilité de revenir en arrière à l'aide du bouton "Back" lorsqu'il sera sur des "sous-pages" de ces sept sections principales.

Commençons par expliquer les modifications survenues pour l'option **voir ses portefeuilles** ("Wallet").

Vous pouvez constater l'ajout d'un **bouton "See guests"** lorsque le client souhaitera voir un portefeuille plus en détail.

Ce dernier l'emmènera sur une **page "Guests"** où le client pourra voir la liste des clients qu'il a invités sur chaque portefeuille.

Cette liste reprendra :

1. Le nom du client invité
2. Le portefeuille où il est invité

En outre, le client propriétaire pourra rechercher un client invité en inscrivant son nom ou son adresse mail.

Le bouton **"add guests"** le mènera à une page fonctionnant de la même manière que celle contenant la liste d'invités.

En effet, il pourra rechercher un client à inviter à l'aide de son nom ou son adresse mail et devra sélectionner un (seul) portefeuille avant de cliquer sur le **bouton "ADD"** pour ajouter ce nouvel invité à un de ses portefeuilles.

Ce bouton est relié à la page "Choose permissions" permettant à un client propriétaire de choisir les permissions qu'il souhaite associer à un client.

Le bouton "Choose" validera l'ajout du client avec ses permissions.

Si nous revenons à la liste des clients invités expliquée précédemment, vous pouvez apercevoir également un **bouton "Go"**.

Ce dernier permet de voir plus en détail le client invité :

1. Le nom du client invité
2. Le portefeuille où il est invité
3. Ses permissions

Sur cette même page, le client propriétaire aura la possibilité de voir le portefeuille associé concerné en détail (bouton "See" menant à son portefeuille), de retirer son invité de ce dernier (bouton "Delete from this wallet") ainsi que de changer ses permissions (Bouton "Change" menant à la page "Choose permissions").

Nous pouvons maintenant passer à notre nouvelle section "**Invited Wallets**".

Cette page affichera de la même façon que la page "Wallet" une liste des portefeuilles où le client est invité.

La seule différence réside dans le fait qu'il ne pourra pas ajouter de portefeuilles par lui-même et qu'il peut voir ses permissions.

Ce dernier doit attendre une invitation dans ses notifications par le client propriétaire.

Le bouton "Go" le mènera sur une page montrant en détail le portefeuille comme pour le client propriétaire sans oublier ses permissions.

Remarquez que lorsque le client cliquera sur le bouton "Go" permettant de voir plus en détail les contrats, il n'aura pas le droit de supprimer un contrat lié.

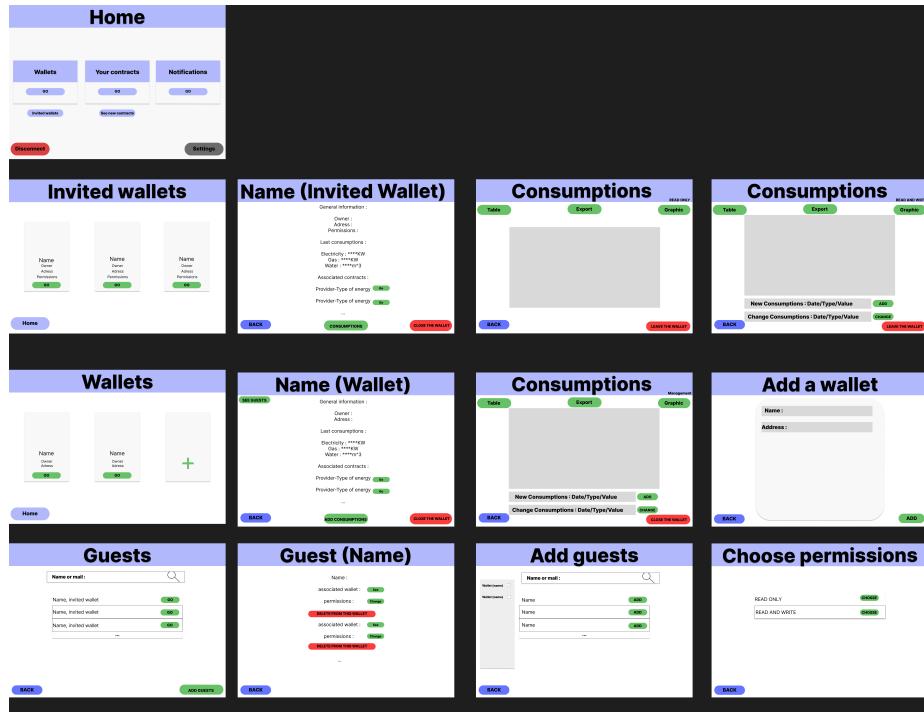
En cliquant sur le **bouton "Consumptions"**, le client pourra soit intéragir avec les consommations de la même manière qu'un gestionnaire en cas de "Read and write" c'est à dire qu'il pourra ajouter et changer des données de consommation, soit il n'aura pas accès à ses options mais pourra uniquement visualiser les données en cas de "Read only".

Il est important de préciser que le niveau d'accès :

1. Read only
2. Read and write
3. Management (pour un client propriétaire)

est indiqué en haut de chaque page concernant la gestion de données de consommation.

16.3 Image de l'interface



17 Extension 1.6.4: Analyse statistique de la consommation énergétique par Adrien Fiévet

Le but de cette extension est surtout de permettre au client de comparer ces données de consommation avec d'autres. Plusieurs possibilités seront proposées.

D'abord, l'application pourra calculer des statistiques par rapport aux données du client. Ce dernier pourra choisir un intervalle de temps et en connaître les statistiques, en autre la moyenne, l'écart type, la médiane, les quartiles, l'écart interquartile ainsi que le minimum et le maximum. Le client pourra ensuite comparer toutes ces données avec les données de consommations qu'il souhaite.

Ensuite, le client pourra comparer ses données en fonction de la date. En effet, il aura l'occasion d'afficher un deuxième tableau ou graphique et de choisir un autre intervalle de temps. Ainsi, il pourra par exemple regarder la différence de statistique entre sa consommation en hiver et en été.

Finalement, il aura également l'occasion de comparer ses données de consommations avec les données d'autres clients ayant les mêmes caractéristiques (en pratique, ces données seront en réalité une simulation et non les données d'autres utilisateurs).

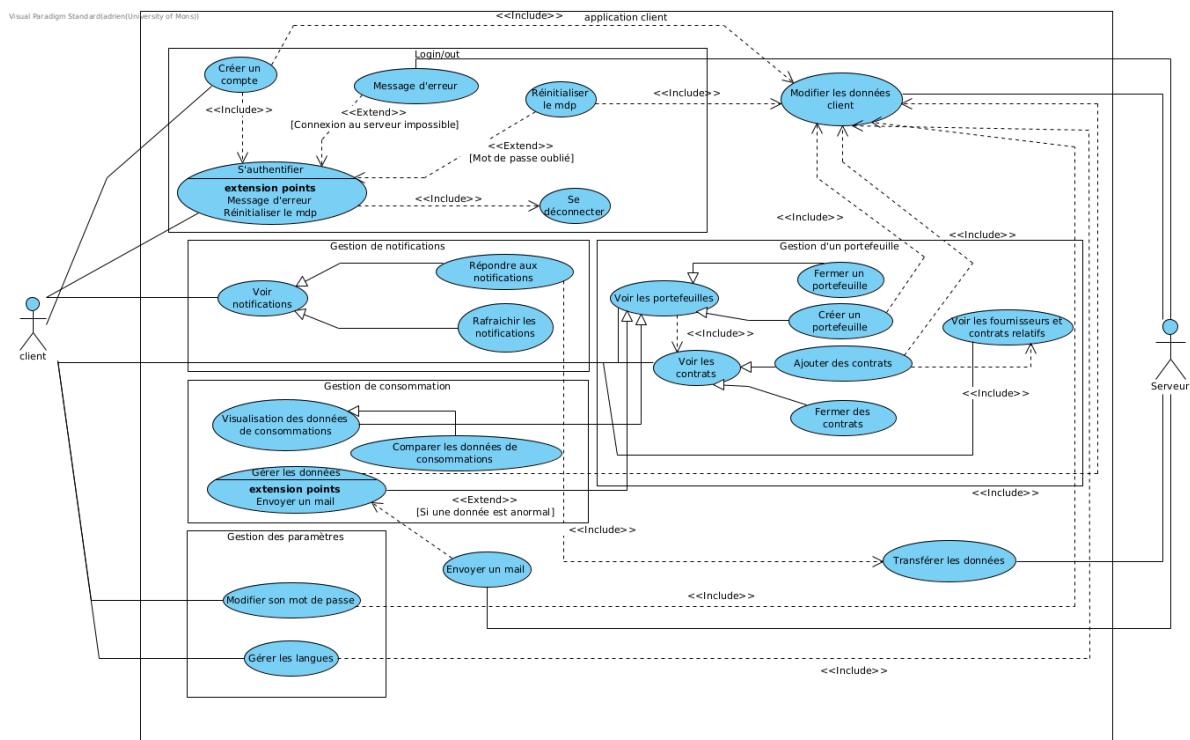
Notez que cette extension implique une autre fonctionnalité. En effet, l'application surveillera chaque donnée de consommation introduite pour prévenir le client si une valeur est anormalement élevée. Un mail sera donc tout simplement envoyé pour expliquer que la donnée entrée est étrange. Le client pourra donc agir en conséquence.

18 Use cases diagrams

Concernant les Use cases pour cette extension, j'ai dû en rajouter deux.

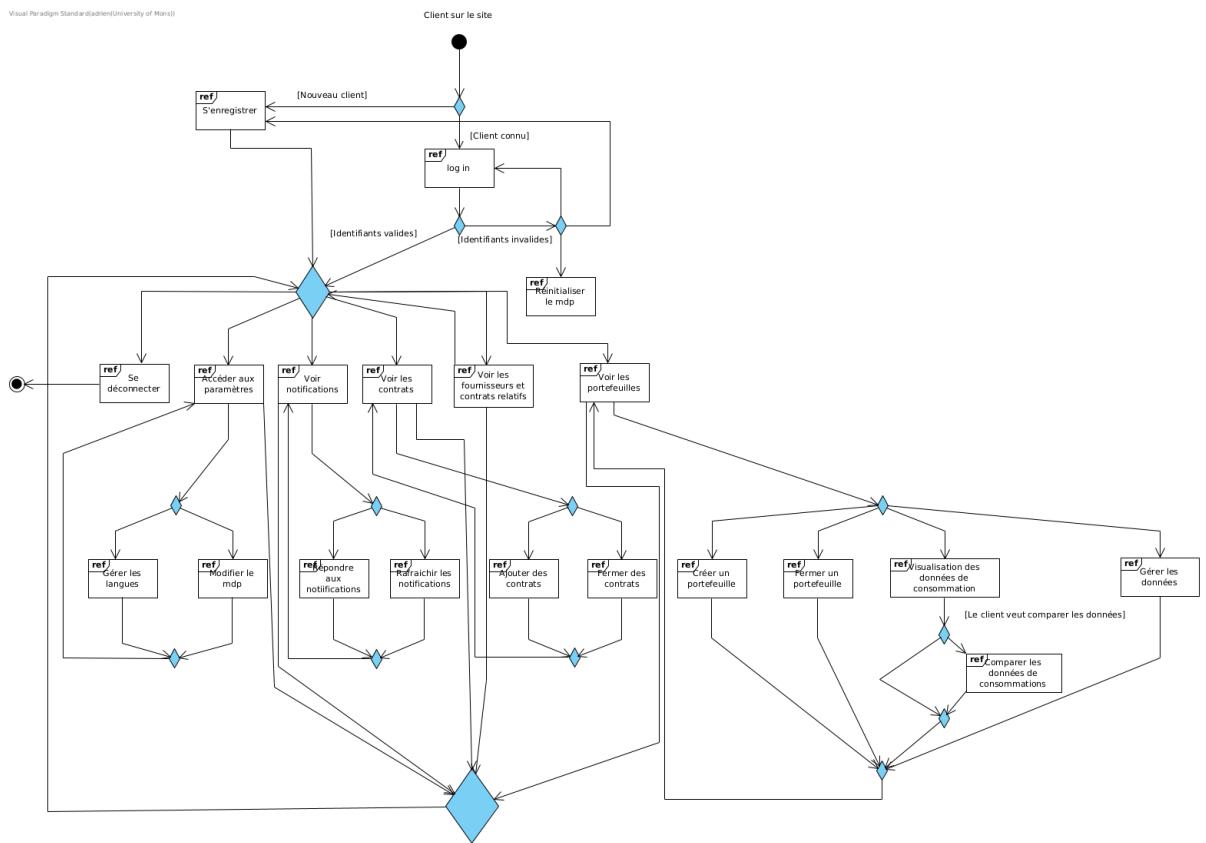
Comparer les données de consommations, celui-ci va permettre à l'utilisateur comme son nom l'indique de comparer ses données, cette faculté a déjà été expliquée dans l'introduction.

Envoyer un mail, quant à celui-ci, il spécifie que si une donnée d'un client est anormalement élevée, alors le serveur enverra un mail pour l'avertir, comme expliqué également dans l'introduction.



19 Interaction overview diagram

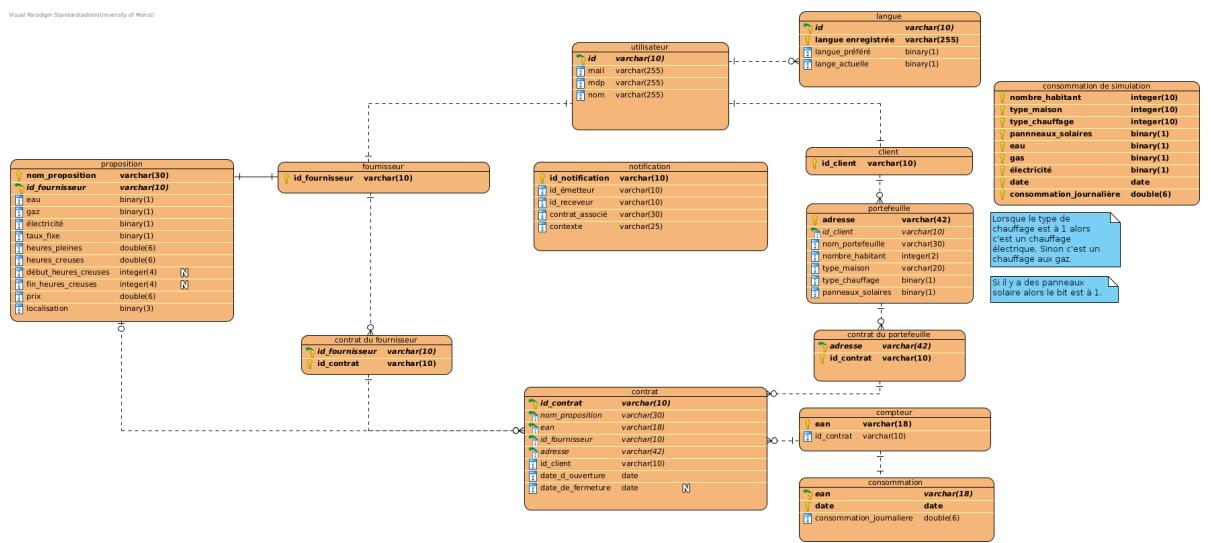
Au niveau de l'overview, j'ai donc ajouté un "interaction use" après la **Visualisation des données de consommation** car l'utilisateur peut décider de comparer ses données une fois qu'il est dans la visualisation s'il le souhaite.



20 Schéma de la base de données

Au niveau de la base de données, j'ai besoin dans un premier temps de stocker plus d'informations sur l'habitation des clients, c'est-à-dire dans la table **portefeuille**; notamment le nombre de personnes habitant dans cette dernière, le type de maison, le type de chauffage et finalement s'il y a une présence de panneaux solaires.

Dans un second temps, j'ai stocké les données de consommations qui serviront de comparaison. Pour cela, une nouvelle table a été créée, à savoir **consommation de simulation**. Nous devons donc y retrouver des données de consommations, un type d'énergie et une date mais il faut aussi pouvoir les distinguer en fonction des caractéristiques de la maison, c'est pourquoi nous y retrouvons également les mêmes types de données que dans la table portefeuille. C'est-à-dire le nombre d'habitants, le type de maison, le type de chauffage et s'il y a des panneaux solaires.

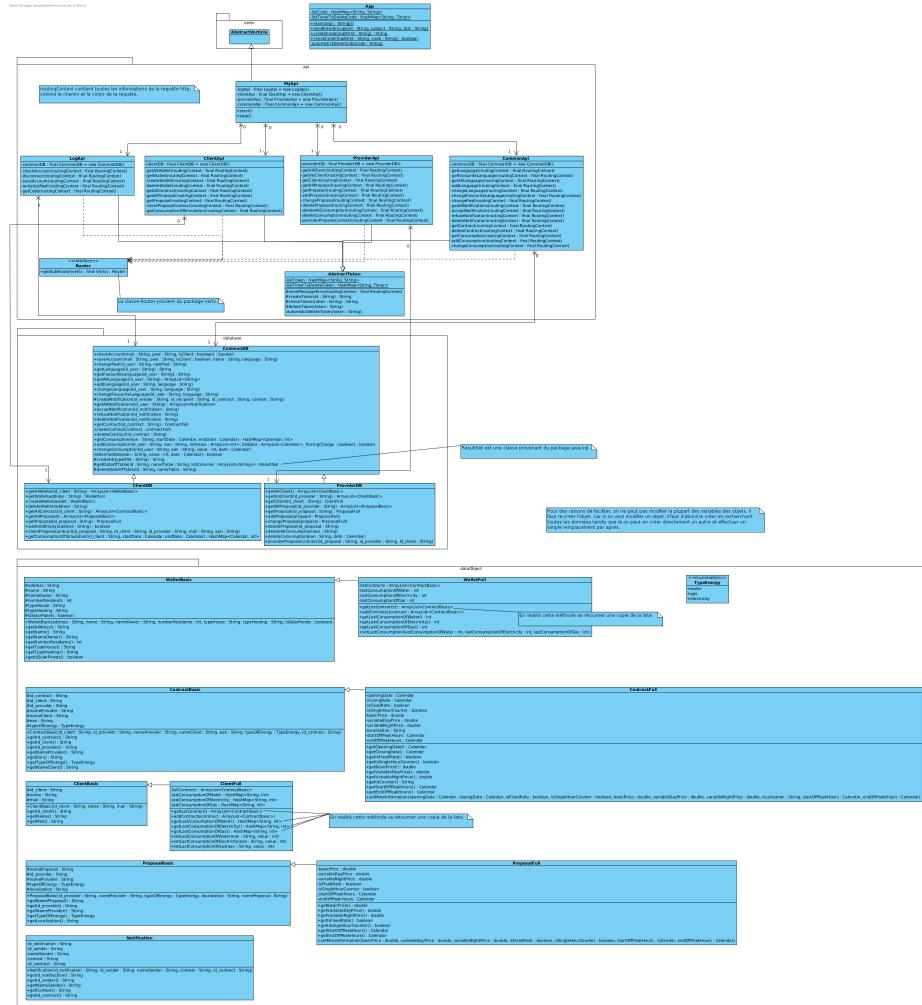


21 Class diagram

Au niveau des méthodes contenues dans la partie serveur, il a d'abord fallu changer l'objet **WalletBasic** afin de stocker toutes les informations nécessaires pour l'extension, à savoir celles citées dans la partie base de données (premier paragraphe).

Et ensuite, sachant que la plupart de l'implémentation se fera côté client, il a fallu rajouter deux méthodes. Une pour recevoir les données de la simulation que l'on va appeler **getConsumptionOfSimulation** et une autre pour vérifier que les données entrées sont normales, que l'on va appeler **isNormalData**.

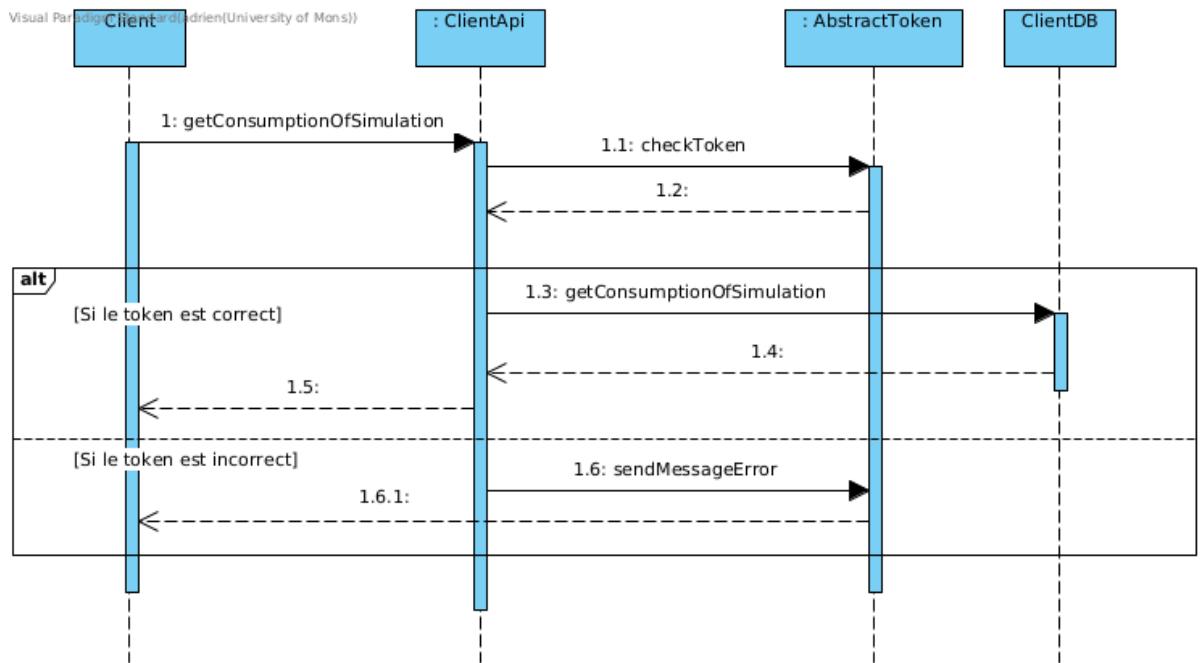
Finalement, j'ai dû rajouter l'id user en paramètre de **addConsumption** et **changeConsumption** afin de prévenir par mail si la donnée est anormale.



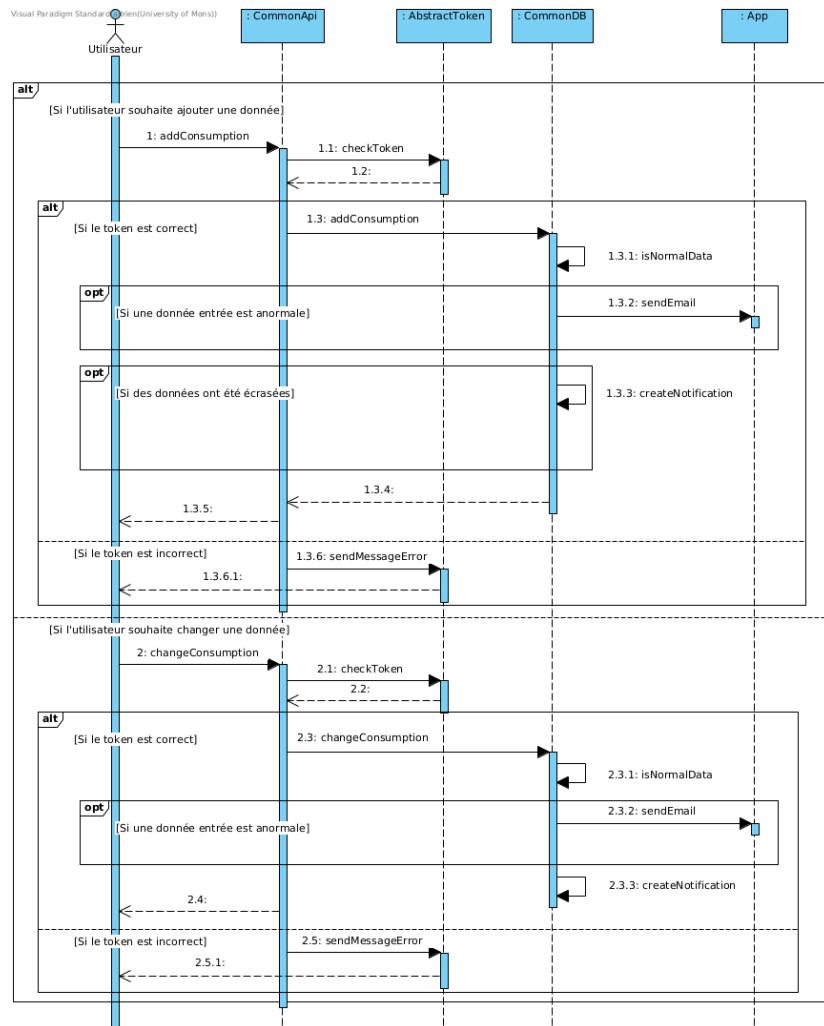
22 Séquence diagram

Afin de compléter mon extension, j'ai introduit deux diagrammes de séquences, un premier nommé **Comparer les données de consommations** et un deuxième nommé **Gérer les données** qui est le même que celui de la base à quelques différences près.

Dans le cas du premier, nous retrouvons un cas de base. c'est-à-dire que le client n'a qu'à appeler la méthode **getConsumptionOfSimulation**, le token est vérifié et si ce dernier est correct alors, dans ce cas, nous appelons **getConsumptionOfSimulation** dans la classe **ClientDB**.



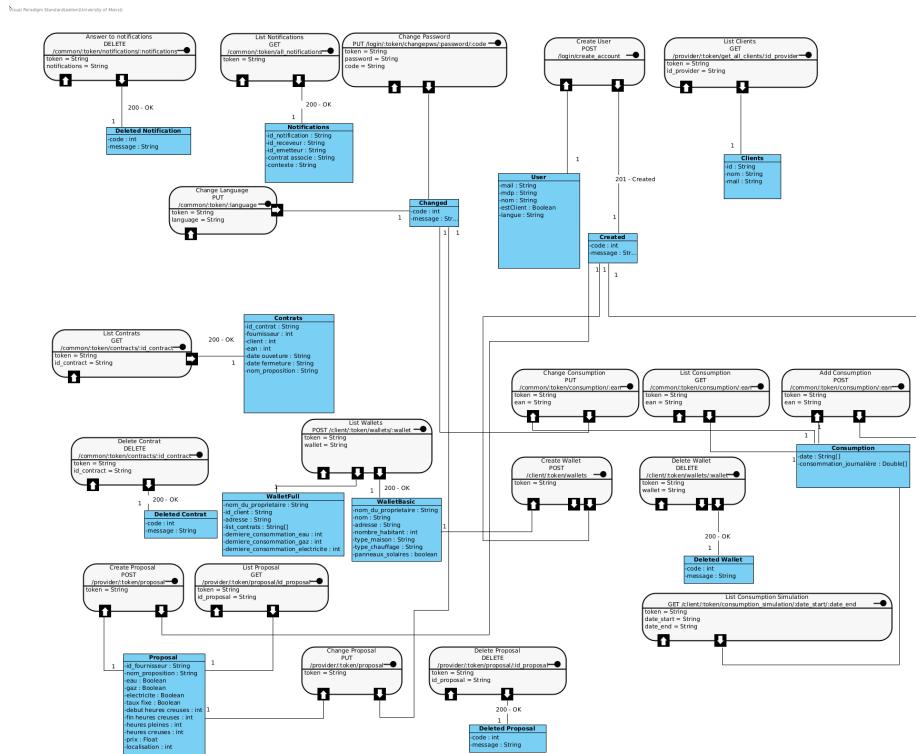
Dans le deuxième cas, que ce soit pour ajouter des données ou les changer, nous vérifions dans la classe **CommonDB** que la donnée est normale grâce à **isNormalData**. Si cette donnée est anormale, nous appelons la méthode statique **sendEmail** de la classe **App** pour envoyer un mail d'information.



23 Api rest

Pour terminer cette extension, il a fallu ajouter une requête API comme illustrée dans le diagramme de séquence sur **Comparer les données de consommations**. Nous retrouvons donc une requête API nommée **List Consumption Simulation** qui appellera la méthode **getConsumptionOfSimulation**. Pour cela, nous envoyons le token ainsi que la date de départ et la date de fin. Nous recevons en retour une liste de dates et de données de consommations journalières.

Notez qu'il a fallu ajouter les paramètres concernant l'extension lors de la création d'un portefeuille et que nous pouvons également envoyer en plus lors d'une requête **List Wallets** pour que le client puisse voir toutes les informations liées à celui-ci.



24 Schéma de l'interface

Pour inclure l'extension dans l'interface, j'ai dû changer 3 pages.

- Add a wallet
- Name (wallet)
- Consumptions

Suivant les explications de la base de données, le client devra entrer plus d'informations dans la fenêtre **Add a wallet** et nous pouvons également les afficher dans **Name (wallet)** vu que le client souhaite voir toutes les informations du portefeuille dans celui-ci.

Dans le cas de **Consumptions**, j'ai d'abord rajouté trois boutons dans le bas de page. Ils représentent les différents modes, c'est-à-dire:

- Voir sa consommation **Just see**
- Comparer ses consommations **Compare**
- Comparer sa consommation avec quelqu'un d'autre (simulation) **Compare with other**

Finalement, j'ai placé un tableau en plus pour le cas où l'on veut comparer des données. Comme vous pouvez le voir, il y a également deux boutons au-dessus de chaque tableau qui permettent de choisir si l'on veut voir les données pures ou les statistiques.



25 Extension 1.6.10 : Application android par Julien Ladeuze

Pour rappel, l'extension consiste à refaire les applications de base sous android. Des fonctionnalités telles que l'écran tactile, le basculement automatique de l'écran doivent être prises en compte.

26 Aspect pratique

Etant donné la nature de l'extention, il est judicieux de ne travailler que sur l'interface graphique de l'application. Tous les autres composants de l'application tels que les uses cases, les interaction overview diagrams, le schéma de la base de données, class diagram du serveur, sequence diagrams, et le schéma de l'API rest seront les mêmes que celles de base.

27 Aspect technologique

Pour programmer une application android, le choix s'est porté sur Kotlin⁷. Une des raisons de cette décision est que toutes les fonctionnalités dites dans ?? seront supportées par Kotlin. En plus de ça, l'apprentissage de ce langage sera plus simple car celui-ci provient de java. Et enfin, Kotlin est très bien pris en charge par JetBrains.

28 Interface

Vu que l'interface sera le seul changement, il est important de respecter les autres schémas. Cela implique que la structure de l'application mobile reste la même que l'application web.

Ensuite, la maquette de l'interface a été faite sur figma. De ce fait, des templates graphiques d'une application android ont permis de rendre l'application la plus conviviale possible.

Après ça, l'interface a été faite de manière à ce que celle-ci en mode paysage soit quasiment la même que celle de base. Nous avons donc jugé qu'il était inutile de représenter l'interface en mode paysage.

Et enfin, il est important de souligner que le bouton permettant de faire un retour en arrière, afficher les applications ouvertes et de retourner dans le home de l'appareil android ne se trouvent pas dans la maquette puisqu'ils n'appartiennent pas à l'application. Néanmoins ils seront tout de même présents lors du lancement de l'application.

⁷<https://kotlinlang.org/>

29 Ressources utilisées

-[https://www.figma.com/file/RYrvHTvHeIzX7KoNq7fAWK/Android-UI-kit-\(Community\)?node-id=0%3A1&t=m1zHQU1n8zkHg6uK-0](https://www.figma.com/file/RYrvHTvHeIzX7KoNq7fAWK/Android-UI-kit-(Community)?node-id=0%3A1&t=m1zHQU1n8zkHg6uK-0)

-[https://www.figma.com/file/rsIFgSQHIKIJ8Eu1Bz8rMN/Interactive-Checkbox-\(Community\)?node-id=0%3A1&t=Pu0V4GNyamIJBSnx-0](https://www.figma.com/file/rsIFgSQHIKIJ8Eu1Bz8rMN/Interactive-Checkbox-(Community)?node-id=0%3A1&t=Pu0V4GNyamIJBSnx-0)

30 Schéma

Pour des raisons de lisibilité, les différents schémas : "log", "client", "fournisseur" sont mis en annexe (voir [Annexe/extension10/](#)). Contrairement à l'interface de base, des noms fictifs ont été mis pour que la maquette soit un peu plus représentative de la réalité. De plus, des flèches entre les différentes images sont mises pour simuler les différents scénarios possibles. A noter qu'il sera toujours possible de revenir en arrière jusqu'à l'image "home" et l'interface de connection pour la partie "log".

31 Extension 1.6.7 : Facturation et paiement d'acomptes par Maxime Dupuis

Dans cette partie, nous implémenterons un système de factures et de paiement d'acomptes. Pour ce faire, nous repartirons des diagrammes de base et nous les étendrons aux cas qui nous intéressent. Les parties ajoutées à chaque diagrammes figurent en rouge afin de distinguer la base de l'extension.

32 Use cases diagram

32.1 Description

Pour commencer, nous regardons les différentes fonctionnalités à implémenter via ce diagramme. Tout d'abord, l'application permet à un client de visualiser l'historique de ses factures ainsi que les acomptes mensuels et leur statut de paiement.

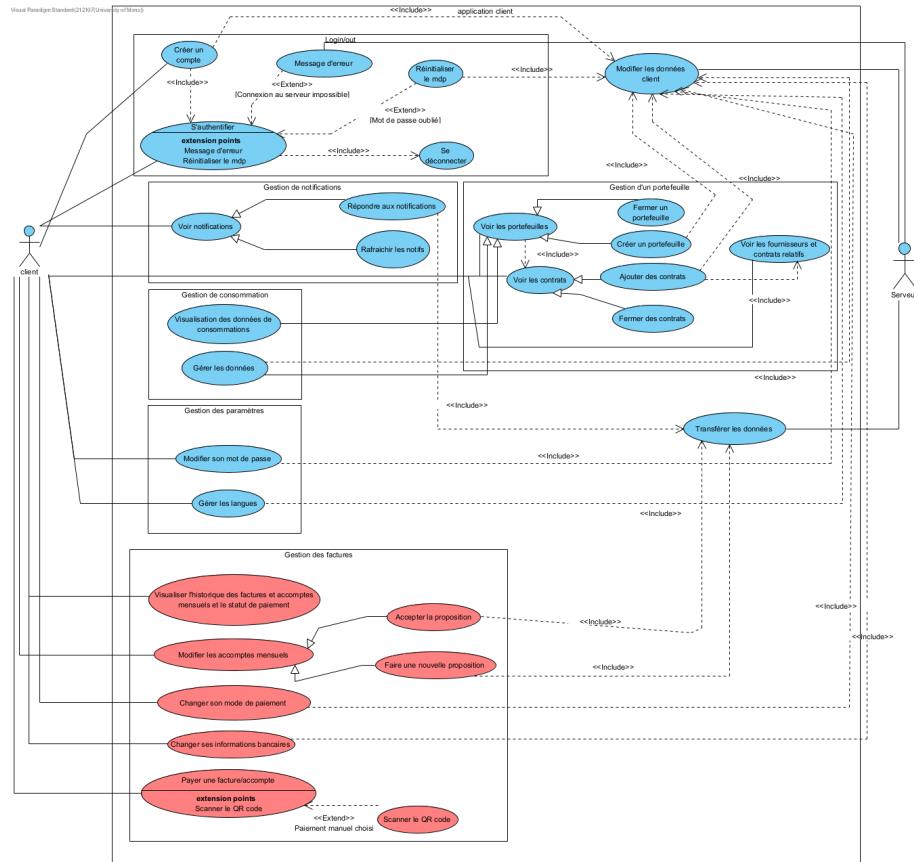
Ensuite, un client peut accéder au montant de l'acompte mensuel proposé par l'application. Celui-ci est calculé automatiquement, basé sur les rapports énergétiques annuels du client, du coût de l'énergie actuel et des installations du client. L'utilisateur peut donc, soit accepter la proposition ou la refuser en proposant une alternative.⁸

De plus, lorsque celles-ci sont erronées, le client peut modifier ses informations bancaires. Dans le cas où le client a choisi une domiciliation, il est possible pour celui-ci de modifier les informations reliées à son compte bancaire. D'autre part, le client peut modifier ce mode paiement et basculer du mode automatique à manuel (ou inversément).

Enfin, le client peut procéder au paiement de sa facture. Celui-ci pourra donc générer un QR code lui permettant d'effectuer un virement bancaire plus simple.

⁸Comprise en +-20 pourcent en fonction de la modification

32.2 Schéma



33 Interaction overview diagram

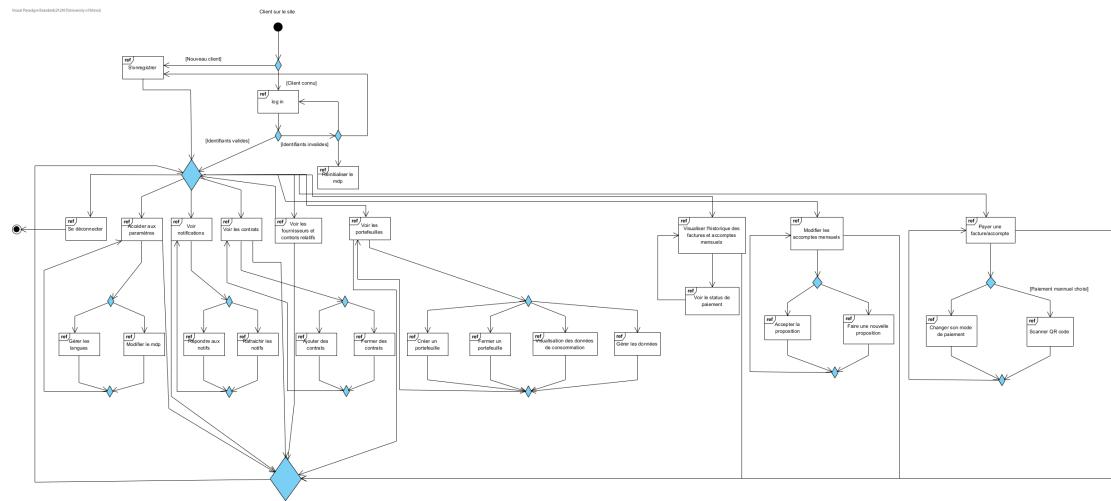
33.1 Description

Basé sur le *use cases diagram*, ce diagramme nous permet de visualiser l'ordre des opérations de l'application. Nous verrons donc ici l'ordre des différentes fonctionnalités de l'extension.

Une fois que l'utilisateur est connecté, la visualisation des factures peut se faire via le menu principal ainsi que le statut de paiement. Dans ce menu, le client peut visualiser les informations d'une facture en particulier. L'utilisateur peut soit modifier l'acompte proposé en faisant une nouvelle proposition, soit changer son mode de paiement, soit changer ses informations bancaires, soit effectuer le paiement.

Dans chaque menu, le client a la possibilité de revenir en arrière jusqu'à la page principale. Dans ce schéma, il n'y a pas mention d'une vérification du paiement. Dans ce cas-ci, ceci pourra être implémenté ramenant le client à la page principale et modifiant le statut de la facture comme étant *payed*.

33.2 Schéma



34 Class diagram

34.1 Description

Dans cette section, nous découvrons les différentes classes liées à l'implémentation des différentes fonctionnalités.

Tout d'abord, le schéma ci-dessous ne reprend que la classe de l'extension (étant donné que le schéma est suffisamment grand et est déjà disponible plus haut). La classe **Facture** est reliée à la classe **ClientApi** puisque que cette extension n'est disponible que pour les clients.

34.2 Les variables

Dans la classe **Facture**, plusieurs arguments sont répertoriés.

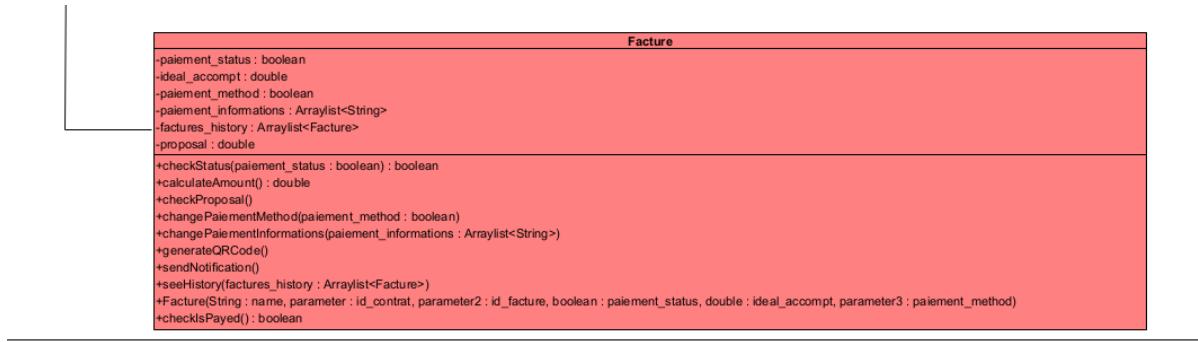
- La variable *paiement-status* permet de voir le statut de paiement de la facture via un boolean (payed=true).
- La variable *ideal-accompt* affiche l'acompte calculé automatiquement par l'application via un double.
- La variable *paiement-method* affiche la méthode de paiement via un boolean (manual=true).
- La variable *paiement-information*s affiche, sous forme d'une ArrayList, les informations de paiement.
- La variable *factures-history*, de la même manière que la variable précédente, affiche un historique des factures du client.
- La variable *proposal* contient, sous forme d'un double, la proposition du client lorsqu'il modifie la proposition faite par l'application.

34.3 Les méthodes

Plusieurs méthodes de cette classe permettent l'implémentation des factures.

- La méthode *checkStatus* permet de vérifier le statut de paiement d'une facture.
- La méthode *calculateAmount* est la méthode par laquelle l'application va calculer le montant idéal.
- La méthode *checkProposal* permet à l'application de vérifier les conditions de la proposition du client.
- La méthode *changePaiementMethod* permet au client de modifier son mode de paiement.
- La méthode *changePaiementInformations* modifie les informations bancaires du client à sa volonté.
- Dans le cas où le paiement manuel est choisi, la méthode *generateQRCode* est appelée afin de générer un QR Code correspondant au paiement.
- La méthode *sendNotification* envoie une notification (via une application externe) lorsque qu'une facture est disponible.
- La méthode *seeHistory* permet au client de voir la liste de ses factures. Chaque facture sera ajoutée dans la liste des factures.
- La méthode *checkIsPayed* est une méthode liée au paiement de vérifier si le paiement est validé ou pas.
- La class est munie d'un constructeur *Facture* créant l'objet Facture pouvant être ajouté dans l'ArrayList.

34.4 Schéma



35 Entity relationship diagram

35.1 Description

Afin de sauvegarder, modifier et supprimer des données, il est nécessaire d'implémenter la notion de factures dans la base de données. C'est pourquoi, ce diagramme illustre l'ajout de factures.

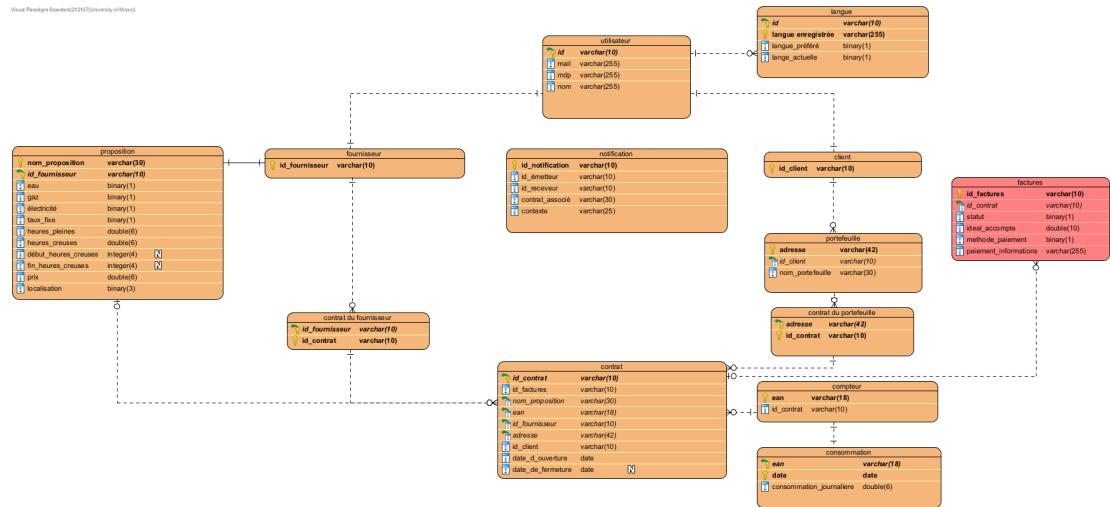
Tout d'abord, une facture est liée à un contrat. C'est pourquoi, nous retrouvons une liaison entre un contrat et une facture. En outre, un élément *id-factures* a été ajouté à l'entité *contrat* afin de relier les 2.⁹

Ensuite, l'entité **factures** regroupe plusieurs variables :

- *id-contrat* relie la facture au contrat avec un varchar de 10 caractères.
- *statut* contient une valeur binaire du statut de paiement.
- *ideal-accompte* contient une valeur double de l'acompte proposé par l'application.
- *methode-paiement* contient une valeur binaire du mode de paiement.
- *paiement-information* contient un varchar avec les informations bancaires du client.

⁹ *id-factures* étant une Primary Key

35.2 Schéma



36 Sequence diagram

36.1 Description

Dans cette section, nous aborderons le déroulement des opérations effectuées lors d'utilisation de factures. Ce diagramme reprend le diagramme en commun avec la base.

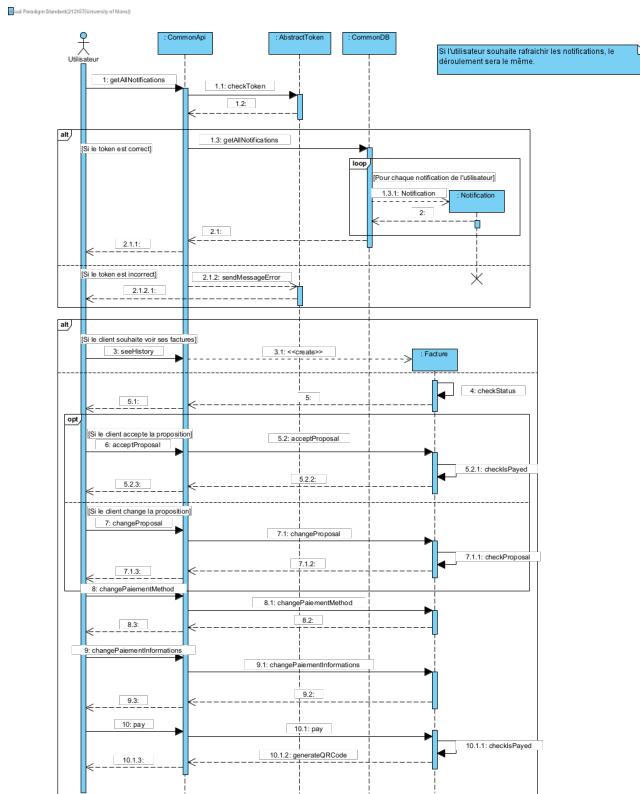
Tout d'abord, le client peut voir l'historique de ses factures. L'utilisateur va donc appeler *CommonApi* via la méthode *seeHistory*. Grâce à cette méthode, l'Api va appeler le constructeur de factures, celui-ci va vérifier le statut de chacune des factures et va retourner la liste des factures.

Ensuite, le client a la possibilité d'accepter ou de modifier la proposition du montant de l'application. Si le client accepte, l'Api va vérifier (grâce à la classe Facture) si la facture est payée pour ensuite retourner à l'utilisateur. Si le client modifie la proposition, la nouvelle proposition va être vérifiée avec la méthode *checkProposal* et retournera à l'utilisateur.

L'utilisateur a la possibilité de changer son moyen de paiement en appelant l'Api avec la méthode *changePaiementMethod*. Le client peut également changer ses informations bancaires via la méthode *changePaiementInformations*.

Enfin, le client peut effectuer le paiement en appelant la classe facture (via l'API) avec la méthode *pay* qui elle-même appellera la méthode *checkIsPayed* afin de vérifier si la facture n'a pas été payée entre-temps. Si ce n'est pas le cas, la classe Facture générera un QR Code avec la méthode *generateQRCode* envoyé au client.

36.2 Schéma



37 Design API Rest

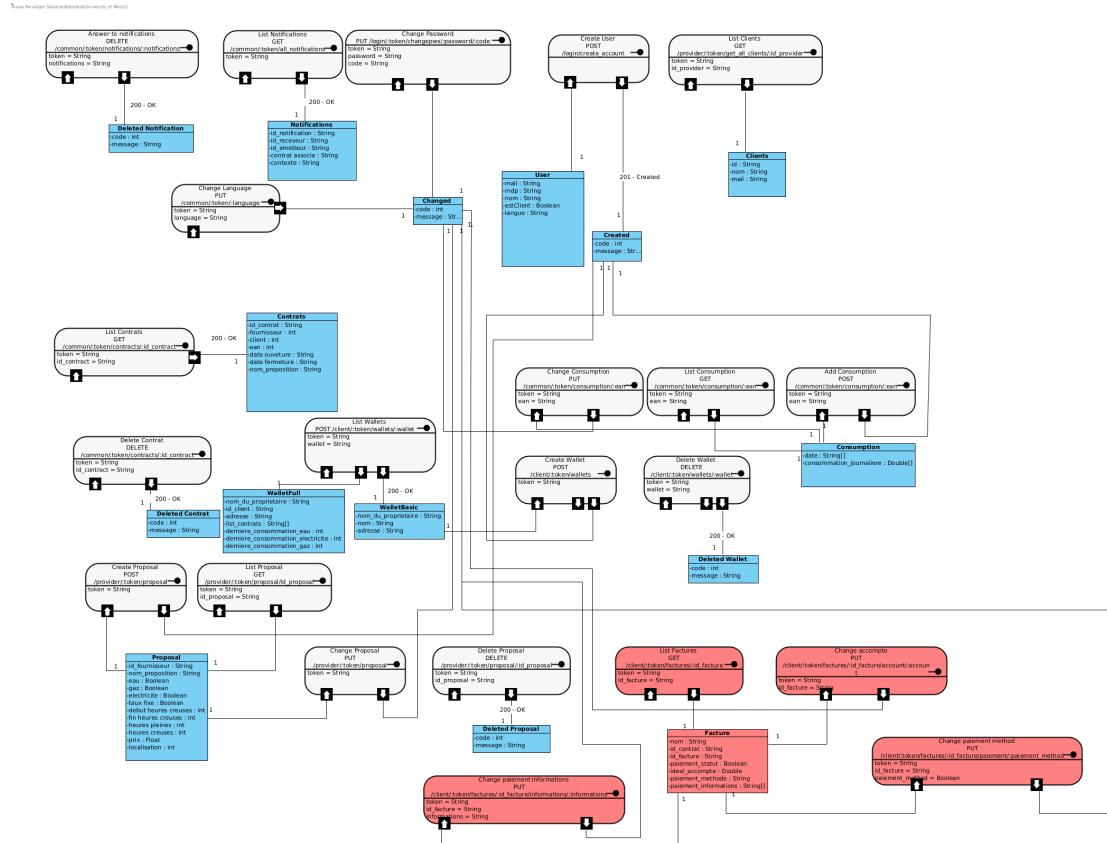
37.1 Description

Afin d'interagir entre l'application et la base de données, des requêtes sont envoyées via l'API Rest permettant cette communication. C'est pourquoi certains services ont été ajoutés à la base commune de l'API. Pour les mêmes raisons que la partie commune, le **token** de connexion est repris à chaque fois

Pour qu'un client puisse voir la liste de ses factures, une requête **GET** doit être envoyée. Dans cette requête, un paramètre *id-facture* est nécessaire pour voir une facture en particulier. Si ce paramètre vaut **NULL**, toutes les factures sont affichées. Cette requête renvoie donc un objet *Facture* comprenant les mêmes éléments cités précédemment.

Un client peut modifier l'acompte proposé par l'application. C'est pourquoi, une requête **PUT** peut être envoyée contenant un paramètre *account* spécifiant la nouvelle valeur de l'acompte. De la même manière, les informations de paiement et informations bancaires peuvent être modifiées à l'aide du même type de requête. Il faut cependant noter que le paramètre *id-facture* ne peut plus être **NULL** sous peine de générer une erreur.

37.2 Schéma



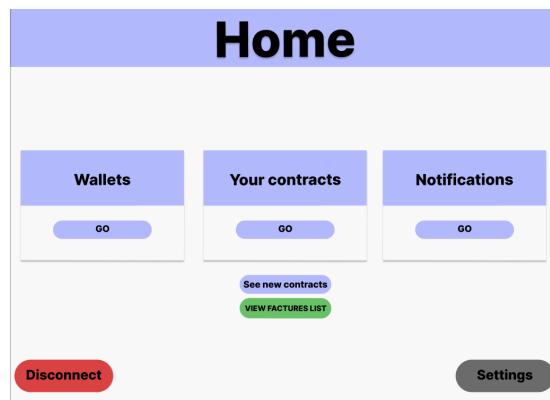
38 Interface

38.1 Description

Pour le design de l'interface graphique de l'application, nous retrouvons un croquis répondant aux demandes de l'extension.

38.2 Page d'accueil

La page d'accueil est la même que l'interface commune si ce n'est qu'un bouton permettant de voir la liste des factures et d'accéder à l'extension a été ajouté.



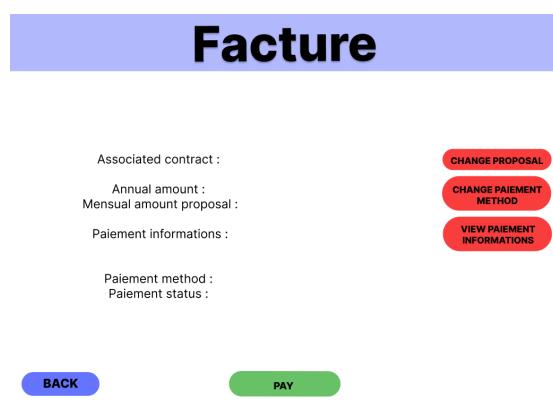
38.3 Liste des factures

Lorsque que le client a cliqué sur le bouton mentionné précédemment, celui-ci se retrouve face à sa liste des factures. L'interface contient donc la liste des factures payées et en attente de paiement. L'application affiche le numéro de la facture et le montant de l'acompte restant à payer.



38.4 Facture

Lorsque que le client a accédé, via le menu précédent, à une facture, l'application affiche les détails correspondant à cette facture. Nous y retrouvons l'id du contrat associé, le montant annuel, le montant mensuel proposé/modifié, les informations de paiement, la méthode de paiement et son statut.



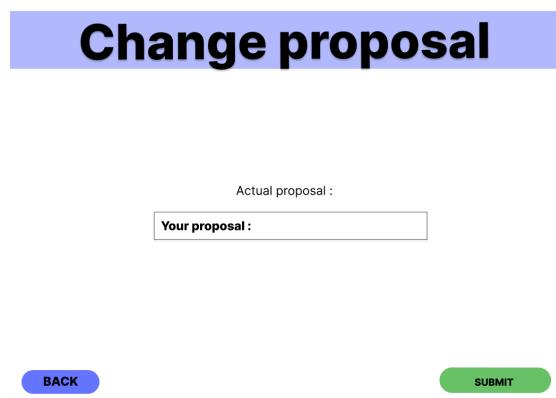
38.5 Paiement

Si toutes les informations conviennent au client, celui-ci peut (dans le cas d'un paiement manuel) procéder au paiement où l'application affichera le QR Code correspondant au paiement.



38.6 Modification de la proposition

Dans le cas où le client n'est pas satisfait de la proposition, celui-ci peut modifier la proposition en proposant un montant +- élevé. Une fois que le client clique sur le bouton "SUBMIT", celui-ci est renvoyé à la page précédente.



38.7 Changement de méthode de paiement

Si le mode de paiement ne convient pas au client, celui-ci peut le modifier et choisir entre mode "AUTOMATIC" ou "MANUAL" en cliquant sur les boutons correspondants. Une fois le choix fait, le client est ramené à la page précédente.

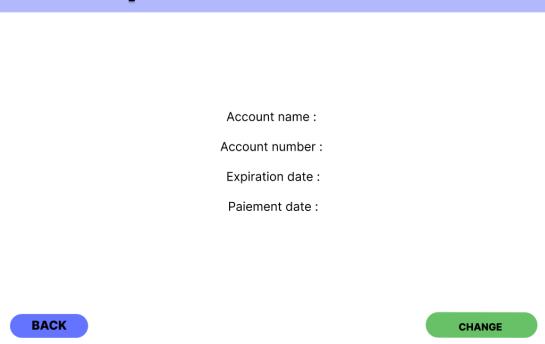
Change paiement method



38.8 Changement des informations bancaires

Le client peut également voir ses informations bancaires et les modifier. L'application affiche le nom du compte, son numéro, la date d'expiration ainsi que la date du prélèvement (dans le cas automatique). Ces informations peuvent être modifiées en cliquant sur le bouton "CHANGE". La page est alors actualisée avec les nouvelles informations du client.

View paiement informations



39 Conclusion

En conclusion, un système de gestion et de paiement de factures est désormais disponible grâce à cette extension. Le client peut donc gérer les acomptes, ses informations bancaires, son moyen de paiement et visualiser l'évolution de ses factures.