PROJET GENIE LOGICIEL (SINFO015) :
Rapport de développement

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

UMONS
Université de Mons

Faculté
des Sciences

Table des matières

1 Base du projet

1.1 Introduction

Cette partie concerne la base du projet.

Les sections suivantes expliqueront nos réflexions sur la façon de parvenir au résultat attendu ainsi que les choix effectués et les problèmes rencontrés.

Lien vers la playlist comprenant les vidéos explicatives :
https://www.youtube.com/playlist?list=PLxRYTywU1xA21Mn7suwrCl8Hv8BMNdeTz

Lien vers le site : https://babawallet-site.alwaysdata.net

1.2 Base de données

1.2.1 Intoduction

Dans cette partie, il s'agira de discuter de la mise en oeuvre de la base de données. Nous allons entre autre parler des choix des technologies utilisées, des changements faits par rapport au diagramme de classe initial et des problèmes rencontrés.

1.2.2 Choix technologiques

Premièrement, il a été décidé de stocker les données sur une base de données **MySQL/MariaDB**. Ceci se fait en adéquation avec les conaissances déjà acquises lors du cours de Base de données 1 (partie SQL).

Deuxièmement, la librairie **java.sql** a été utilisée pour gérer les interactions avec la base de données. La raison principale est que java.sql est assez intuitif et possède une courbe d'apprentissage assez courte qui se résume juste à sa documentation principale. En dépit du peu de possibilité que possède cette api, elle a convenu à notre méthode d'implémentation.

Troisièmement, des **variables d'environnement** ont été mises en place pour ne pas laisser les identifiants de la base de données en dur dans le code.

Quatrièmement, la testabilité de l'application a été permise grâce à **Junit5**. A noter que les tests vérifient "grosso modo" si la requête SQL a bien été écrite. De plus, l'utilisation de mock object est impossible étant donné la nature du framework utilisé pour communiquer avec la base de données. C'est pour cela qu'une base de données test a été créée pour permettre d'éxécuter les requêtes.

Cinquièmement, l'encryption des mots de passe dans la base de données a été rendue possible via le framework **springframework.security** qui utilise l'algorithme Bcrypt (déjà expliqué dans la phase de modélisation).

1.2.3 Changements effectués

Pour convenir aux remarques faites lors de la phase d'analyse, il a été decidé d'effectuer plus de distinctions dans le package **database**. Effectivement, il a été décidé de faire correspondre une classe à une ou au plus 3 tables de la base de données. Par exemple, **ConsumptionManager** va se charger principalement des tables : `consumption` et `counter`.

Ensuite, des classes ont été rajoutées. Nous avons tout d'abord la classe **DB** qui s'occupe de générer la connexion en ne possédant qu'une instance (Singleton pattern). De plus, **Table** et **Query** ont été rajoutées pour avoir une meilleure abstraction. Enfin, **CommonDB** a été remodelée pour ne posséder que des instances des classes "Manager" ce qui permet une meilleure cohésion avec le package **api**.

En ce qui concerne l'ajout de certaines méthodes non présentes dans notre diagramme de classe initial, nous n'avions pas pu prévoir ces méthodes lors de notre analyse.

1.2.4 Problèmes rencontrés

Effectuer du test-driven-development était quelque chose d'extrêmement compliqué à cause du fait d'avoir comme erreur dans la console : "the syntax of the query at line 1" sachant que la requête de se fait en une ligne.

Les clés étrangères ont été plus ennuyantes qu'utiles étant donné qu'aucune requête ne prend de terme "DELETE ON CASCADE".

Lors des différents tests, il se pouvait que la base de données "crash" sans aucune raison.

1.3 API

1.3.1 Intoduction

Nous allons voir dans cette section le choix du framework pour lancer un serveur API ainsi que les modifications effectuées pour la partie API. Nous allons ensuite évoquer un petit programme pour simuler un compteur numérique et finalement nous allons expliquer les plus gros problèmes rencontrés.

1.3.2 Choix du framework

Comme expliqué dans le rapport de modélisation, nous avons choisi d'utiliser le framework Vertx pour gérer l'API. Nous sommes partis sur cette solution car nous avons trouvé un excellent tutoriel très complet pour apprendre le fonctionnement de Vertx, en sachant qu'il y a également une très bonne documentation ainsi qu'un serveur discord si jamais on avait vraiment rencontré des gros problèmes. Ce framework étant facile à utiliser, nous n'avons eu aucun problème à lancer nos premiers serveurs testes.

1.3.3 Modification apportées

Avant de passer en revue les différentes modifications apportées, nous précisons dans un premier temps que nous avons gardé la structure des fichiers énnoncée dans le rapport de modélisation.

1. Nous avons rajouté une nouvelle méthode pour obtenir la liste des contrats entre un client et un fournisseur, cette dernière est appelée getAllClientsOfContract.
2. Nous avons ajusté quelques chemins API afin de convenir à nos besoins.
3. Nous avons finalement manipulé les dates comme des objets String et non des Calendar.
4. Nous avons ajouté la gestion de pagination. Notamment grâce à la méthode getSlice se trouvant dans le fichier MyApi.
5. Nous avons rajouté une nouvelle méthode pour obtenir les consommations par mois.
6. Nous avons totalement changé la gestion des tokens, la classe AbstractToken a été supprimée et tout est maintenant géré dans la classe MyApi grâce à Vertx.
7. Nous avons ajouté une protection par rapport à l'autorisation des requêtes. En effet, seules les requêtes de notre site et d'always data sont autorisées.
8. Nous avons dû faire de la surcharge de constructeur pour createNotification car si c'est le client qui accepte, il doit également spécifier son adresse et son code ean.

9. Pour vérifier les tokens, le rôle de l'utilisateur ainsi que la source de la requête, nous avons ajouté une classe interne dans MyApi qui se nomme HandlerUtils. Elle regroupe les handlers utilisés pour effectuer les vérifications.
10. Comme notre api a dû être lancée sur always data comme un site, nous n'avons pas pu utiliser les timers de java. Pour régler le problème, nous avons ajouté des timer task dans always data qui effectuent des requêtes à intervalles réguliers pour supprimer les codes email et les contrats.
11. Nous avons découvert qu'il fallait rajouter des en-têtes CORS pour pouvoir effectuer correctement les requêtes, il y a un donc un handler qui rajoute tous les en-têtes nécessaires.
12. Le token contient la date d'expiration ainsi que l'identifiant et le rôle de l'utilisateur. Pour pouvoir récupérer toutes ces informations, nous avons rajouté une méthode nommée getDataInToken qui retourne la valeur que l'on souhaite en String.
13. Nous avons retiré la méthode changeProposal car elle était superflue. En effet, dorénavant, pour changer une proposition, nous re-appelons la méthode addProposal avec le même nom de proposition pour écraser l'ancienne.
14. Auparavant, nous avions une méthode pour réinitialiser le mot de passe lorsque nous l'avions oublié et une autre changePwd pour changer le mot de passe depuis la page paramètre une fois connecté. Nous avons donc supprimé cette dernière pour appeler dans tous les cas renitializePwd.
15. Nous avons appliqué, pour changer une consommation, le même principe que pour changer une proposition. Nous avons donc supprimé la méthode changeConsumption.
16. Pour finir, nous avons rajouté une requête API et une méthode pour supprimer son compte, elle a été placée dans LogApi et elle appelle une méthode de la base de données également rajoutée à cet effet.

1.3.4 Compteur numérique

Pour simuler un compteur numérique, nous avons donc créé un programme annexe se trouvant dans le dossier numeriqueMeter. Dans ce dossier se trouve un programme java géré par gradle, vous pouvez donc le lancer. Ce dernier a besoin de plusieurs arguments :

- L'adresse mail du compte
- Le mot de passe du compte
- Le code ean du contrat
- Le type d'énergie (e—g—w)

Une fois ce programme lancé, il ajoutera une consommation plus ou moins aléatoire en fonction du type d'énergie toutes les 5 secondes et affichera dans la console la date d'ajout ainsi que la valeur. Pour l'arrêter, il suffit d'entrer n'importe quel caractère et ensuite appuyer sur "entrer" pour fermer le programme correctement.

1.3.5 Problèmes rencontrés

Au niveau des API, il y a essentiellement deux problèmes que nous avons eu beaucoup de difficultés à régler.

D'un côté nous voulions lancer notre serveur comme un vrai serveur always data. Cependant, il était donc en http et comme le port d'always data n'était pas le port par défaut, nous n'avons pas réussi à faire vérifier notre serveur pour le faire passer en https. Pour régler ce problème, nous avons été obligé de lancer notre serveur comme un site mais l'inconvénient est donc que les timers de java n'étaient plus utilisables et nous avons donc mis en place des timer task pour contourner cet inconvénient.

D'un autre côté, le plus gros problème que nous avons rencontré est qu'il fallait ajouter des en-têtes CORS car il y a une protection internet qui bloque nos requêtes entre le site et le serveur car ils ne sont pas dans le même domaine. Nous n'avions aucune idée des en-têtes qu'il nous manquait ni comment les ajouter correctement.

1.4 Front-end

1.4.1 Intoduction

Dans cette partie, nous allons vous expliquer la façon dont nous avons implémenté le front-end.

Le fichier "index.js" permet d'identifier plus clairement les parties, nous avons travaillé en prenant en compte ces différences :

- **Pages liées à la gestion des comptes**
- **Pages liées aux clients**
- **Pages liées aux fournisseurs**
- **Pages liées aux fonctionnalités communes**

1.4.2 Pages liées à la gestion des comptes

1. **Login** :
   Cette page permet aux utilisateurs de se connecter.
2. **CreateAccount** :
   Cette page permet aux utilisateurs de créer un compte.
3. **ForgottenPassword** :
   Cette page donne la possibilité aux utilisateurs de changer de mot de passe s'ils le souhaitent ou s'ils ne se souviennent plus de celui-ci.

1.4.3 Pages liées aux clients

1. **HomeClient** :
   La page d'accueil des clients.
2. **Wallets** :
   Cette page comprend tous les portefeuilles du client.
3. **WalletFull** :
   Cette page permet de voir les informations liées à un portefeuille en particulier en partant de la page Wallets.
4. **AddWallet** :
   Cette page donne la possibilité au client d'ajouter un portefeuille.
5. **NewContracts** :
   Cette page permet au client d'envoyer une notification au fournisseur afin d'essayer d'avoir un nouveau contrat avec ce fournisseur en question.
6. **ContractInformation** :
   Cette page permet de voir les informations liées à une nouvelle proposition se trouvant dans NewContractsPage.
7. **ContractPage** :
   Cette page permet de voir les contrats en cours qu'un client a en commun avec un fournisseur.

1.4.4 Pages liées aux fournisseurs

1. **HomeSupplier** :
   La page d'accueil des fournisseurs.
2. **Clients** :
   Cette page comprend tous les clients du fournisseur.
3. **ClientFull** :
   Cette page permet de voir les informations liées à un client en particulier en partant de la page Clients.
4. **AddClient** :
   Cette page donne la possibilité au fournisseur d'ajouter un client avec la proposition qu'il désire.
5. **CreateContract** :
   Cette page permet au fournisseur de créer des nouvelles propositions.
   Notez que si on entre "baba" en durée, une fois le contrat associé à un client, celui-ci durera une heure.
6. **SupplierContractPage** :
   Cette page permet au fournisseur de voir les propositions qu'il a créées.
7. **ProposalFull** :
   Cette page permet de voir les informations liées à une proposition en particulier en partant de la page SupplierContractPage.
8. **SupplierModifyContract** :
   Cette page donne la possibilité au fournisseur de modifier ses propositions ou ses contrats en cours s'ils sont variables.

1.4.5 Pages liées aux fonctionnalités communes

1. **Notifications** :
   Cette page donne la possibilité aux utilisateurs de voir leurs notifications ainsi que les accepter ou les refuser.
2. **ContractFull** :
   Cette page permet aux utilisateurs de voir un contrat en cours de manière détaillée en partant de la page ClientFull ou WalletFull.
3. **ConsumptionPage** :
   Cette page permet aux utilisateurs d'observer la consommation liée à un contrat.

1.5 Modifications apportées

Quelques modifications par rapport à la maquette fournie ont été apportées. En effet, lors de la conception de la maquette, nous ne connaissions pas les banques CSS mises à notre disposition et donc nous n'avions pas beaucoup pensé au design [1].

Dans l'ensemble, le design du site respecte tout de même la thématique que nous avions apporté lors de la réalisation de la maquette. Cependant quelques modifications de design comme : Les notifications, l'affichage des wallets/contrats/proposal, la suppression d'une barre de recherche, les formulaires et la page de login, ... Toutes ces choses ont reçu des modifications au niveau du design.

1.5.1 Problèmes rencontrés

Nous avons rencontré un souci avec notre router, l'ensemble des redirections fonctionnait en local mais pas sur alwaysdata en ligne.

Nous n'avions pas compris que nous envoyions des fichiers statiques (venant du dist) sur le serveur.

De ce fait, la configuration se faisait en nodejs mais pas en fichier statique, ce qui était la source du problème.

1. Toutes les banques de CSS utilisées ont été citées dans le readme.

1.5.2 Les langues

Pour respecter les normes d'internationalisation, nous avons du implémenter le module *i18n*. Ce module nous permettait de traduire l'ensemble de notre site dans des langues que nous aurions prédéfinies. Ici, les langues ajoutées sont le français et l'anglais. C'est pourquoi, il n'y a aucun texte codé en "dur" mais nous faisons l'usage de *clés de traduction* pour rediriger le site vers la bonne traduction en fonction de la langue stockée en base de données. La langue est chargée en base de donnée et est stockée localement dans la variable *i18n.locale*.

Pour ajouter une langue, il suffit d'ajouter un fichier *.json*, de rajouter cette langue (via un ajout de ligne) dans le fichier json des langues configurées et ajouter les différentes clés de traduction.

1.5.3 Modules importés

- **VueCookies** :
  Ce module est importé afin de gérer les cookies.
- **VueSweetalert2** :
  Ce dernier est importé afin d'obtenir diverses pop-ups.
- **i18n** :
  Celui-ci est importé dans le but de gérer les langues comme expliqué dans la section précédente.
- **bluebird** :
  Ce module est importé pour permettre de laisser un temps d'attente.
  Celui-ci est utile notamment lorsqu'on souhaite afficher une pop-up et ensuite rediriger.

1.6 Conclusion

Le développement de ce projet nous a permis d'acquérir de l'expérience concernant le fonctionnement d'un site comprenant une API et une base de données.

Ce projet nous a beaucoup appris et nous a permis de mettre en pratique plus concrètement certaines notions [2].

2. Le readme contient la documentation manipulée.

1.7 Readme

Les tests unitaires relatifs à la partie backend fonctionnent parfaitement dans l'IDLE intellij. Malheureusement, il semble avoir un problème quand on tente d'effectuer `gradle test` dans le terminal. Ceci viendrait d'un problème d'accès à la base de données. Il a été tenté par différents moyens de résoudre le problème sans succès. De ce fait, les preuves de réussites des tests ont été déposées dans l'archive.

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

2.1 Introduction

Afin de réaliser cette extension, j'ai dû apporter des modifications au projet de base. Les sections suivantes expliqueront ceci et mes réflexions sur la façon d'atteindre le résultat.

Notez que les emplacements du code modifié par mon extension porte le commentaire "Extension Claire"

2.2 Base de données

2.2.1 Introduction

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

La base de données se voit ajouter deux tables :

1. **invitedTable** :
   Cette table permet de regrouper de manière claire les informations nécessaires à la gestion des clients invités.
2. **invitation** :
   Celle-ci permet de gérer les informations échangées pour les invitations aux portefeuilles.

Cependant, il n'y a pas de modifications apportées aux tables de la base du projet.

2.2.2 Package dataobject

Vous pouvez observer l'ajout des classes :

- **Invitation**
- **InvitedClient**

L'une permet de regrouper les informations des clients invités sur un portefeuille, l'autre permet de regrouper les informations sur les invitations envoyées d'un client à l'autre.

Vous pouvez également remarquer l'ajout d'une liste de clients invités dans "WalletFull".

En effet, c'est dans la page de "WalletFull" que la gestion des clients invités se fera plus tard.

De plus, l'ajout de la possibilité d'avoir des permissions dans "WalletBasic" permet de séparer les invités pouvant lire ou lire et écrire des propriétaires.

Effectivement, il suffit d'obtenir une fois les permissions côté front-end grâce au "WalletBasic" pour prendre connaissance de si la page "WalletFull" ou la page de consommations doit s'afficher de manière à répondre à un propriétaire, à un client invité en lecture ou à un client invité en lecture et écriture.

Il n'est donc pas nécessaire de l'inclure à la classe "WalletFull".

2.2.3 Package database

Dans ce package, nous avons l'ajout de deux "Manager" ainsi qu'une légère modification de "WalletManager".

1. **InvitationManager** :
   Ce "Manager" vient principalement du choix de séparer les notifications des invitations.
   Cette décision me permet de gérer de manière à part entière les ajouts dans mes deux tables notamment dans invitedTable.
   Cette manière de procéder permet aussi de garder une séparation entre les notifications (échangées entre les fournisseurs et les clients) et les invitations (échangées uniquement entre clients).
   La table "invitation" contient :
   - **invitationId** :
     L'identifiant de l'invitation.
   - **senderId** :
     L'identifiant du client émetteur.
   - **receiverId** :
     L'identifiant du client receveur.
   - **address** :
     L'adresse du portefeuille concerné.
   - **permission** :
     La permission accordée.
   - **nameSender** :
     Le nom de l'émetteur.
   - **type** :
     Le type quant à lui permet de départager les invitations, les acceptations et les refus d'invitations. Ce qui est utile côté front-end afin de déterminer s'il s'agit d'une invitation à accepter ou refuser ou d'un simple retour négatif ou positif à marquer comme lu.

   Vous pourrez donc retrouver dans cette classe les méthodes suivantes :
   (a) **createInvitation** :
   Cette méthode permet d'envoyer une invitation à un autre client ou simplement de répondre positivement ou négativement à cette dernière.
   Notez que cette méthode permet aussi de vérifier si le client a entré un identifiant existant et s'il n'a pas essayé de s'ajouter lui-même.
   (b) **refuseInvitation** :
   Cette méthode donne la possibilité de refuser une invitation en renvoyant à l'émetteur que sa demande a été refusée.
   (c) **acceptInvitation** :
   Cette méthode donne la possibilité d'accepter une invitation en renvoyant à l'émetteur que sa demande a été acceptée.

De plus, on prend en compte le retour de "addInvited" de la classe "InvitedClientManager".

(d) **getAllInvitation** :
Cette méthode permet d'obtenir toutes les invitations d'un client. Récupérer toutes ces informations permet en front-end de mieux gérer les explications données à ce dernier.

(e) **deleteInvitation** :
Cette méthode permet de supprimer une invitation ou une réponse positive ou négative.

2. **InvitedClientManager** :
Ce "Manager" est utilisé pour les actions à effectuer sur la table **"invitedTable"** en ce qui concerne les clients invités sur un portefeuille.
La table "invitedTable" contient :

- **address** :
L'adresse du portefeuille.
- **invitedId** :
L'identifiant du client invité.
- **ownerId** :
L'identifiant du client propriétaire.
- **permission** :
La permission accordée au client invité.

Vous pourrez donc retrouver dans cette classe les méthodes suivantes :

(a) **getAllInvitedClients** :
Cette méthode permet d'obtenir la liste de tous les clients invités sur un portefeuille.
Elle est utilisée dans "WalletManager" pour le "getWallet".

(b) **deleteInvitedClient** :
Cette méthode donne la possibilité de supprimer un invité d'un portefeuille.

(c) **addInvited** :
Cette méthode donne la possibilité d'ajouter un invité sur un portefeuille.
Notez que cette dernière permet aussi de vérifier si le client n'était pas déjà invité sur le portefeuille.

(d) **changePermission** :
Cette méthode permet de modifier la permission donnée à un client invité sur un portefeuille.

L'ajout d'une méthode nommée **"getAllInvitedWallets"** dans **WalletManager** permet de corréler les éléments cités plus tôt dans le package dataobject. Il y a maintenant une méthode pour obtenir tous les portefeuilles sous forme de "WalletBasic" sans permission pour le propriétaire et une autre avec permission pour les invités.

Ces permissions étant reprises à l'aide de la table **"invitedTable"**.

En outre, pour supprimer un portefeuille, il faut maintenant que le client n'ait plus de contrats mais aussi plus d'invités.

2.3 API

2.3.1 Introduction

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

Notez que les ajouts se trouvent dans ClientApi de ce package étant donné que cette extension ne concerne que les clients.

2.3.2 ClientApi

Vous pouvez remarquer l'ajout des méthodes suivantes afin de faire la passerelle entre le front-end et la base de données.

1. **getAllInvitedWallets** :
   Cette méthode permet d'appeler la méthode de "WalletManager" expliquée précédemment dans le but d'obtenir les portefeuilles "invités" avec la permission liée.
2. **deleteInvitedClient** :
   Cette méthode permet d'appeler la méthode de "InvitedClientManager" afin de supprimer un invité.
3. **changePermission** :
   Cette méthode permet d'appeler la méthode de "InvitedClientManager" afin de changer la permission d'un invité.
4. **getAllInvitation** :
   Cette méthode permet d'appeler la méthode de "InvitationManager" dans le but d'obtenir toutes les invitations et refus ou acceptations de ces dernières.
5. **proposeInvitation** :
   Cette méthode permet d'appeler la méthode de "InvitationManager" afin d'envoyer une invitation à un autre client pour espérer l'ajouter à la liste des invités. Cette méthode renvoie un code 500 si les conditions expliquées précédemment ne sont pas remplies.
6. **acceptInvitation** :
   Cette méthode permet d'appeler la méthode de "InvitationManager" afin d'accepter une invitation. Cette méthode renvoie aussi un code 500 si les conditions expliquées précédemment ne sont pas remplies.
7. **refuseInvitation** :
   Cette méthode permet d'appeler la méthode de "InvitationManager" afin de refuser une invitation.
8. **deleteInvitation** :
   Cette méthode permet d'appeler la méthode de "InvitationManager" afin de supprimer une invitation. Cette dernière est également utile pour le front-end pour les invitations à marquer comme lues.

2.4 Front-end

2.4.1 Introduction

Dans cette partie, je vais vous expliquer les modifications et ajouts apportés au front-end de la base du projet.

Vous pouvez constater l'ajout de quatres fichiers :

1. **AddInvited**
2. **ChangePermissions**
3. **InvitationMessage**
4. **InvitedWallet**

2.4.2 AddInvited

Cette page permet à un client d'ajouter un invité en saisissant l'identifiant de l'invité et en sélectionnant la permission qu'il souhaite lui accorder. Cette manière de procéder semblait plus intuitive pour l'utilisateur, il suffit que la personne que ce dernier souhaite inviter aille dans ses paramètres pour voir son identifiant et ainsi lui donner.

2.4.3 ChangePermissions

Cette page permet simplement de changer la permission d'un invité.

2.4.4 InvitationMessage

Cette page répertorie toutes les demandes d'invitations et permet de les accepter ou refuser et de voir les demandes qui leur ont été refusées ou acceptées.

2.4.5 InvitedWallet

Cette page permet comme expliqué précédemment de récupérer tous les portefeuilles où le client est invité ainsi que la permission correspondante.

Notez que cette fois, lorsqu'on se dirige vers "walletFull", les permissions sont encodées dans le sessionStorage.

2.4.6 WalletFull et consumptionPage

Ces deux pages reposent sur le même principe, je récupère la permission associée et grâce aux directives de condition, j'affiche les possibilités pour le client en fonction qu'il soit propriétaire ou s'il est invité en lecture ou lecture et écriture.

2.4.7 Modules importés

- **jwt-decode** :
  Ce module est importé afin d'obtenir l'identifiant de l'utilisateur.

2.5 Conclusion

Le développement de cette extension et du projet en général m'a donné l'occasion de manipuler et mieux comprendre le principe d'une API et d'une base de données.

J'ai pu également voir comment les diverses parties communiquent.

Ce projet fut enrichissant.

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

3.1 Introduction

Dans cette partie, je vais expliquer les changements que j'ai dû faire afin de réaliser l'extension sur l'analyse des données de consommations.

Les explications seront divisées en 3 parties :

1. Frontend
2. API
3. Base de données

3.2 Base de données

Finalement, au niveau de la base de donnée, j'ai simplement ajouté toutes les nouvelles caratéristiques des portefeuilles que ça soient dans la vrai bdd et l'objet WalletBasic. Je n'avais besoin de rien d'autre si ce n'est une petite méthode pour obtenir l'adresse mail d'utilisateur à partir de son id pour pouvoir lui envoyer un mail (LogManager) et une autre pour obtenir l'adresse qui est lié à un contrat (ContractManager).

3.3 API

Au niveau de l'API, voici la liste des modifications effectuées :

1. Ajout de la gestion des informations supplémentaires pour les portefeuilles
2. Ajout d'une méthode dataIsNormal qui permet d'envoyer un mail si une donnée paraît étrange
3. Ajout d'une méthode genereValue qui permet de générer des données pour servir de simulation
4. Ajout d'une méthode getOtherConsumptions ainsi que son chemin API pour retourner les valeurs générées

Notez que contrairement à la partie modélisation, j'ai décidé de ne pas sotcker de valeur de simulation étant données le nombre conséquent de valeur que j'aurais eu besoin pour toutes les possibilitées. J'ai donc opté pour une méthode qui calcul elle-même des valeurs fictives sur le moment en fonction des caractéristiques ajoutées dans le portefeuille ainsi que la saison.

3.4 Front-end

Au niveau frontend, il y a princalement 2 gros changements que j'ai dû effectuer. Notez que j'ai également dû rajouter des traductions pour mon extension dans les fichiers fr.json et en.json.

1. Les informations du portefeuilles
2. La page pour voir les consommations

3.4.1 Les portefeuilles

Pour cette partie ci, j'ai rajouté des inputs lors de la création des portfeuilles afin d'y mettre plus d'information. On y retrouve :

1. Le nombre d'habitant
2. La taille de l'habitation
3. Si c'est une maison ou bien un appartement
4. Si l'habitation est chauffé au gaz ou à l'électricité
5. S'il y a des panneaux solaires

J'ai donc également affiché toutes ces nouvelles informations lorsqu'on affiche toutes les données d'un portefeuille.

3.4.2 Consommations

Cette seconde partie étant le centre de mon extension, c'est ici qu'il y a eu les plus gros changements. Afin de rescpecter les consignes de mon extension, j'ai rajouté à cette page un bouton à option en bas de page qui permet de choisir le mode d'affichage que l'on souhaite. On y retrouve :

1. (Just See) : le mode de base
2. (Compare with other) : Pour comparer ses données avec quelqu'un d'autre (la simulation)
3. (Compare over time) : Pour pouvoir comparer ses données à deux endroits en même temps
4. (Statistic) : Pour voir les statistiques des consommations de l'utilisateur par rapport au données affiché actuellement sur le graphique

Notez donc que j'ai finalement opté pour un seul nouveau bouton reprenant les différentes possibilités de mon extension, je n'ai pas gardé le fonctionnement emis lors de la partie modélisation.

3.5 Conclusion

En conclusion, afin de réaliser mon extension, j'ai dû changer plusieurs choses dans les différents parties du projet. Ceci ma permis d'en apprendre vraiment un maximum sur le fonctionnement d'un site et une API. En comparaison avec la partie modélisation, j'ai changé la manière d'obtenir les données de comparaison ainsi que le design lorqu'on se trouve sur la page des consommations.

4 Extension 1.6.10 : Application android par Julien Ladeuze

4.1 Introduction

Dans cette section, il s'agira de parler de l'implémentation du côté front-end de l'application sous android. Les choix technologiques, difficultés rencontrés et ressentis seront abordés.

A noter que seulement la partie log a été faite et les fonctionnalités tels que le swipe et le basculement en portrait paysage (possible mais mauvais rendu) sont indisponibles .Il a tout d'abord été choisi d'implémenter quelque chose de fonctionnel et pas uniquement de l'UI. Les raisons supplémentaires seront expliquées par après.

4.2 Choix technologiques

Tout d'abord, le language du language s'est porté sur le **java**. En dépis du fait qu'apprendre le Kotlin aurait été quelque chose d'extrèmement instructif, il aurait été impossible de fournir une application android.

Ensuite, pour ce qui est de l'envoi/réception de requête. Il a été choisi de prendre le framework **retrofit**. Ce framework possède de multiples avantages tels que : apprentissage facile, dé/sérialisation automatique des objets, "type-safe", supporte plusieurs clients HTTP et il est extensible et customisable. En somme, il est très puissant.

De plus, un design pattern d'architecture a été pris pour rendre le code le plus compréhensible possible. Celui-ci est le **Model-View-ViewModel** (MVVM). Celui-ci sépare la partie graphique, donnée et logique de l'application. Ceci nous permet d'avoir un code qui peut être facilement extensible.

Pour générer l'icone de l'application, un site web a été utilisé : `https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html#foreground.type=clipart&foreground.clipart=android&foreground.space.trim=1&foreground.space.pad=0.25&foreColor=rgba(96%2C%20125%2C%20139%2C%200)&backColor=rgb(68%2C%20138%2C%20255)&crop=0&backgroundShape=circle&effects=none&name=ic_launcher`

Bien évidemment, le format du code de l'application est adapté à une app android.

4.3 Difficultés rencontrées

La raison principale du peu de contenu produit a été évidemment le temps. En effet, l'extension a commencé à être produite une semaine avant la remise de celle-ci. Bien qu'il aurait été possible d'y consacrer la majeure partie de la journée, la santée et l'union des cours priment.

Ensuite, le fait de programmer en android n'est pas du tout une tâche aisée. Même si le java était déjà acquis depuis bien longtemps, de nouveaux aspects tels que les layouts, framework android, retrofit... se sont introduits dans le developpement nécéssitant de consacrer son temps à énormément de documentation. De plus, l'arborescence d'un projet android est très disparate.

Enfin, même si l'extension compte pour 40% de la note. Il faut avoir une base solide prête à toute épreuve. C'est pourquoi il a été choisi de se concentrer plus sur la base que l'extension.

4.4 Ressentis

Quand j'ai commencé cette extension, je me suis dis : pourquoi ?. Pourquoi avoir pris cette extension ? Honnêtement, je ne sais pas. Je voulais certainement tester quelque chose de nouveau et c'est chose faite. En effet, J'ai appris plein de choses et il me reste tellement à découvrir ! En plus de cela, J'ai adoré. Ca faisait longtemps que je n'avais jamais autant aimé programmer(depuis le cours de projet d'info en BAC1) et de voir le résultat ! Ce fut une expérience très courte (une semaine et demi) mais très enrichissante et passionante.

5 Extension 1.6.7 : Facturation et paiement d'acomptes

5.1 Introduction

Lors de l'implémentation d'un fonctionnement de facturation, quelques modifications ont eu lieu par rapport à la phase d'analyse fait au premier quadrimestre. En effet, je ne m'étais pas rendu compte de la difficulté à implémenter cette fonctionnalité

5.2 API

Nous retrouvons ici toutes les fonctionnalités implémentées par rapport à l'API. Bien entendu, nous ne reviendrons pas sur la partie commune qui est la même qu'expliqué précédemment. Ceci n'est qu'un explication des ajouts fait à cette API.

Tout d'abord, dans la class *ClientAPI* nous retrouvons ces différentes méthodes. Toutes ces méthodes répondent au même fonctionnement : Recevoir une requête via le paramètre *routingContext* et en retirer l'id, le body et parfois un élément présent dans la requête comme un id client. Ensuite, ces méthodes appeleront les Managers *InvoiceManager* pour les factures et *BankManager* pour toute la gestion du compte bancaire.

1. getAllInvoices :
   Permet de recevoir la liste des *InvoiceBasic*[3] lié à un client.
2. getInvoice :
   Permet de recevoir une *InvoiceFull* avec plus d'informations
3. changePaymentMethod :
   Permet de changer la méthode de paiement (Manuel ou Automatique)
4. changeAccountInformation :
   Permet de changer les informations bancaires.
5. changeProposal :
   Permet de changer la proposition d'accompte mensuel faite par le client.
6. addBank :
   Permet de créer une entrée bancaire en base de données lors de la création d'un client.
7. getBank :
   Permet de récupérer les informations bancaires d'un client.
8. changePaid :
   Permet d'effectuer le paiement d'une facture.

3. Voir section Base de Données

5.3 Base de données

Dans cette partie, nous verrons les différentes *class* implémentées pour l'extension ainsi que les rajouts de tables dans la base de données

Dans la base de données, 2 tables ont été ajoutées :

1. Bank :
   Contient les informations bancaires d'un client. Une nouvelle ligne de cette table est crée dès lors qu'un client s'inscrit sur le site. Son id client est le même que dans la table *client* et les reste des colonnes est fixé à 0 ou null.
2. Invoice :
   Contient toutes les factures. Chaque facture est liée à un id client ainsi qu'un contrat. Bien entendu, un contrat correspond à une seule facture et vice-versa.

Pour les class Java, il y a 3 nouveaux Objets et 2 Managers. Pour les Objets il y a :

1. Bank :
   Qui va créer un objet *Bank* pour stocker les informations bancaires d'un client telles que le numéro de compte, le nom du compte et la date d'expiration.
2. InvoiceBasic :
   Qui va créer un objet *InvoiceBasic* avec peu d'informations pour éviter le surplus d'informations lors de la requête. Ces informations sont l'id du client, le prix, la proposition (qui correspond initialement à un douzième du prix annuel) et le status (payé ou non).
3. InvoiceFull :
   Qui va créer un objet *InvoiceBasic* avec les mêmes informations mais les informations supplémentaires telles que l'id du contrat, le restnt à payer, la méthode de paiement ou la date de paiement seront ajoutées via une méthode *setMoreInformation*

Pour les Managers, il y a 2 nouveaux Managers et quelques modifications aux Managers de la base pour répondre au fonctionnalités de l'extension.

1. InvoiceManager :
   Ce Manager va permettre de gérer la création et la modification d'informations d'une facture. On ya retrouve les méthodes suivantes :
   (a) doesInvoiceExist :
   Permet de savoir si une facture existe déjà ou non
   (b) getProposalName :
   Permet de récupérer le nom de la proposition associée à un contrat
   (c) createInvoice :
   Créer une facture en base de données

(d) changePrice :
Mets à jour les informations liées au prix

(e) changeAlreadyPaid :
Mets à jours les informations d'une facture lors d'un paiement

Le reste des méthodes font la même chose que leur homologue dans la partie API

2. BankManager :
Ce Manager va permettre de gérer la création et la moficiations d'informatons d'un compte bancaire. On y retrouve les méthodes suivantes :

(a) doesBankExist :
Permet de savoir si une compte bancaire existe

Le reste des méthodes font la même chose que leur homologue dans la partie API

3. ConsumptionManager :
Dès qu'une consommation est ajoutée/modifiée, la prix de la facture correspondante est mis à jour et une notification est envoyée au client.

Pour des raisons pratiques, une facture est créer dès qu'un contrat est établi entre un fournisseur et un client. Cette facture est mis à jour à chaque nouvelles consommation et le client peut déjà payer la facture à ce moment là. Aucun mail n'est envoyé pour dire qu'une facture est prête mais j'ai utilisé le fonctionnement de notifications pour envoyer, à mise à jour de facture, une notification correspondante.

5.4 Font-end

Quelques modifications par rapport à la maquette de départ on été apportées :

1. ChangeInvoiceProposal :
   Permet de changer la proposition en respectant du marge de 20 pourcent autour de la proposition initiale.
2. ChangePaiementInformations :
   Permet de changer les informations du compte bancaire
3. InvoiceFull :
   Permet d'afficher les informations complètes et détaillées d'une facture. On peut aussi, via une pop-up, modifier la méthode paiement.
4. InvoicePage :
   Permet d'afficher toutes les factures d'un client et leur statut de paiement.
5. PaiementQR :
   Permet d'effectuer le paiement avec un QR Code. Le fonctionnement bancaire n'ayant pas pu être implémenté, un bouton a été mis pour simuler le paiement effectué.
6. NotificationPage :
   Une nouvelle notification a été ajoutée pour prévenir lorsqu'une facture a été mise à jour.

5.5 Conclusion

Durant toute la réalisation de cette extension, j'étais face à beaucoup de problèmes étant donné que je découvrais la partie back-end en même temps. Le manque de temps ne m'a pas permis de rendre une extension comme j'aurai voulu et la difficulté d'implémentation liée à cette extension ne m'a pas aidé. Les fonctionnalités demandées étaient assez précises et n'étant pas du monde de la facturation, je n'ai pas pu respecter certaines normes classiques liées au paiement.

Finalement, ce fut un projet enrichissant, qui a tester mes limites physiques et mentales.