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

1.2 Base de données

1.2.1 Intoduction

1.3 API

1.3.1 Intoduction

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 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.
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 un proposition en particulier en partant de la page SupplierContractPage.
8. **SupplierModifyContract** :
   Cette page donne la possibilité au fournisseur de modifier ses propositions et 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 le refuser.
2. **ContractFull** :
   Cette page permet aux utilisateurs de voir un contrat en cours en partant de la page ClientFull ou WalletFull.
3. **ConsumptionPage** :
   Cette page permet aux utilisateurs d'observer la consommation liée à un contrat.

1.4.6 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.

1.4.7 Les langues

1.4.8 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.

1.5 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 [1].

1. Le readme contient la documentation manipulée.

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 modifiés 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 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.