Mini projet Android

Application GreenLiving

Ludovic Lehmann et Jonathan Cornaz

Contenu

Introduction/Objectif

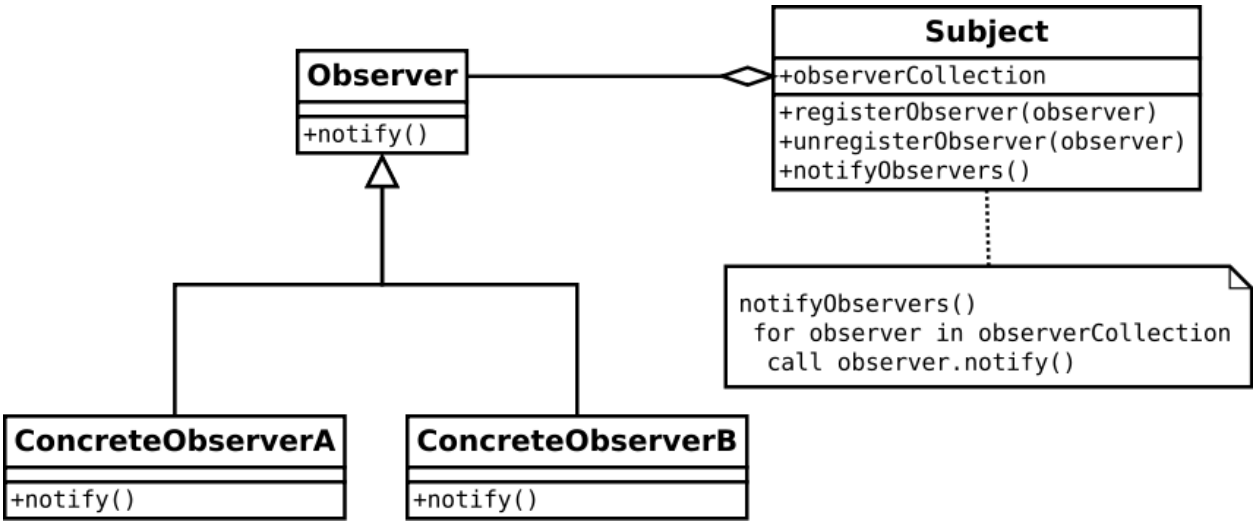
Dans le cadre du cours de systèmes d'exploitation mobiles et applications, nous avons l'occasion de nous familiariser avec l'environnement de développement et les concepts d'Android. L'objectif de ce mini-projet est de développer une petite application Android. Dix mini-projets différents étaient disponibles. Nous avons choisi le numéro 6 : GreenLiving.

Le but de l'application est de faciliter la gestion d'un budget. L'utilisateur peut y saisir son salaire, les prévisions de ses dépenses ainsi que saisir les dépenses effectivement effectuées. L'application doit alors lui permettre de facilement savoir s'il est en train de dépasser son budget ou non.

Généralité

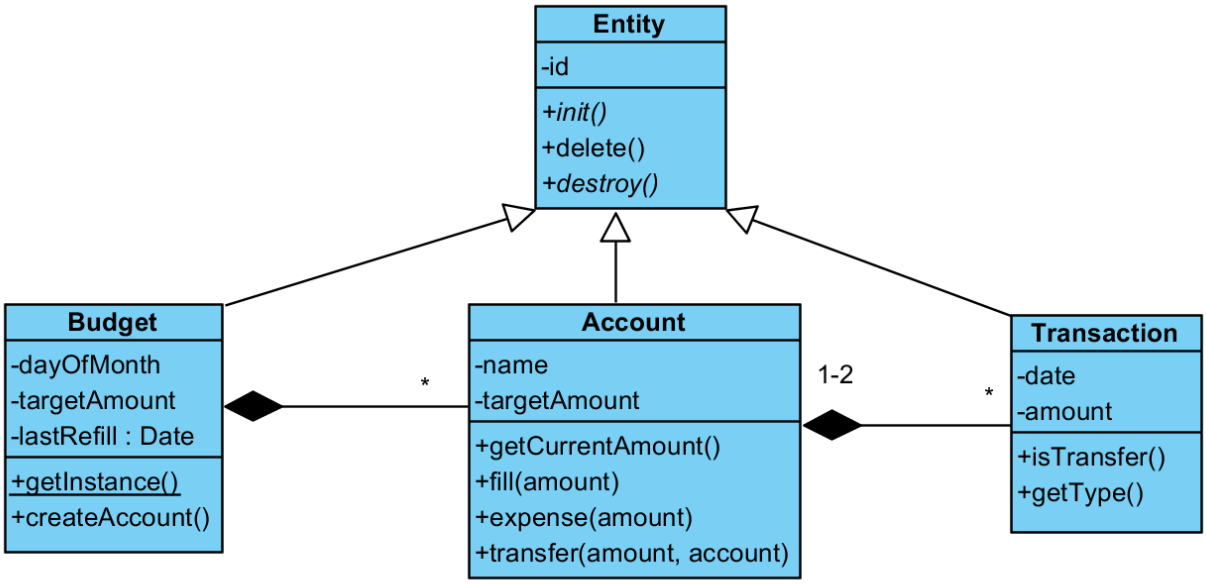
Le patron de conception (design pattern) observateur/observé est l'un de ceux qui sont le plus utilisés en Java. Grâce à ce patron de conception, nous pouvons déclencher des actions dans plusieurs classes (les observateurs) lorsqu'il se passe un évènement dans la classe observée.

Nous avons choisi de l'utiliser à chaque fois que c'était possible, afin de faciliter le développement de l'application et surtout de gérer les évènements de manière efficace. Ainsi dès que le modèle est modifié l'interface graphique et la couche de persistance vont être averties.



(source Wikipedia)

Modèle de données



Le modèle de données se base sur la classe “Entity” qui permet de généraliser les comportements de toutes les entités.

La classe “Budget” est un *singleton* qui représente le budget en lui-même. C’est là que sont définis le salaire mensuel de l’utilisateur (*targetAmount*) et le jour du mois auquel il touche son salaire (*dayOfMonth*). C’est depuis cette classe que l’on peut créer des comptes.

La classe “Account” représente un “compte” pour lequel on prévoit des dépenses. On lui définit un nom, ainsi que le montant que l’on prévoit de dépenser mensuellement (*targetAmount*). Depuis cette classe, on peut créer des transactions.

La classe “Transaction” représente les enregistrements des dépenses effectivement faites. Elle permet également de représenter l’argent gagné ainsi que des transferts d’argent entre les comptes. Ces transferts permettent de changer l’allocation que l’on a fait de l’argent en cours de mois. Les transactions vont influencer le montant courant (*currentAmount*) du compte.

Le budget possède également un Compte spécial appelé “Off-budget”. L’argent prévu pour ce compte représente l’argent qui n’est pas encore budgeté. Ainsi si le “targetAmount” de ce compte est positif, alors l’utilisateur a budgeté moins que ce qu’il gagne. Et un montant négatif l’avertit qu’il a, au contraire, trop budgeté.

Automatiquement à son ouverture, l’application va vérifier s’il est temps re-remplir les comptes. Pour ça, elle va vérifier quelle est la dernière date de remplissage et la comparer avec la date courante ainsi qu’avec le jour du mois du salaire définit dans le budget. Le cas échéant elle va

créditer sur chaque compte l'argent qui lui est alloué mensuellement. Il est à noter que le compte "Hors budget" est également touché par ce mécanisme. Ainsi, il est possible d'accumuler de l'argent non-budgété et de le transférer sur un autre compte à postériori.

Persistance

Pour la persistance, nous avions trois objectifs :

- Les mécaniques de persistance ne doivent pas être mêlées avec le modèle.
- La couche de persistance doit être autonome et se charger automatiquement d'effectuer les opérations nécessaires lorsque le modèle évolue.
- Le code de la persistance doit s'abstraire le plus possible de la technologie de persistance un maximum (SQLite)

Afin d'atteindre ces objectifs nous avons décidé d'utiliser ORM (Object - Relationnal Mapping). Ceci nous a dispensé de l'écriture des instructions SQL, et donc fourni les abstractions souhaitées quant à la technologie tout en nous permettant d'être complétement détaché du modèle. En effet, seules des annotations ont dû être ajoutées dans la description des classes du modèle. Et si nous avons utilisé *ORMLite* avec *SQLite*, la dépendance à ces technologies reste faible et elles pourraient théoriquement être changées sans grande modification ni même impacter le modèle.

La couche de persistance est générée par une classe principale "PersistenceManager" qui a la charge d'instancier les autres classes de persistance. Lorsqu'elle est démarée par l'appel de la méthode statique *PersistenceManager.start(Context),* la persistance charge les données stockées et s'abonne comme observatrice du modèle. Dès qu'elle sera notifiée d'un changement (par la méthode *Observable.notifyObservers()*) elle analyse la nature du changement (création, modification, suppression) et appelle les méthodes de persistance nécessaires.

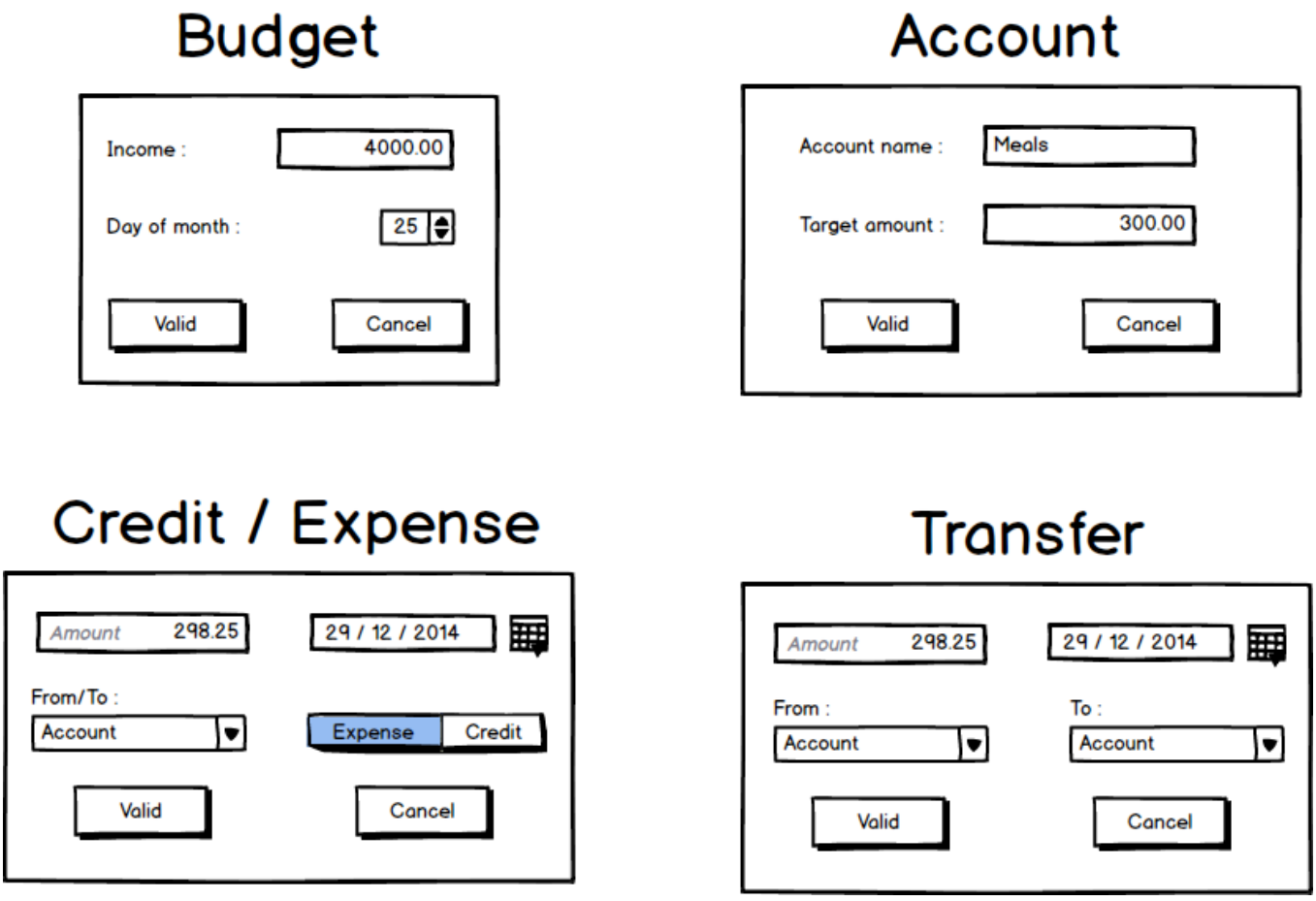
Interface graphique

Lors de la conception de l'application nous avons choisi de faire une vue avec deux onglets :
-un onglet «Comptes»;
-un onglet «Transactions».

Dans chacun de ces onglets doit s'afficher une liste contenant les objets correspondants.

Nous avons choisi aussi de faire un menu qui permet d'accéder aux différentes fonctionnalités de l'application (Créer un nouveau compte, définir un budget, dépenser de l'argent etc…)

Ce menu doit changer en fonction de l'onglet actif. Les contrôles doivent s'afficher dans des petites fenêtres

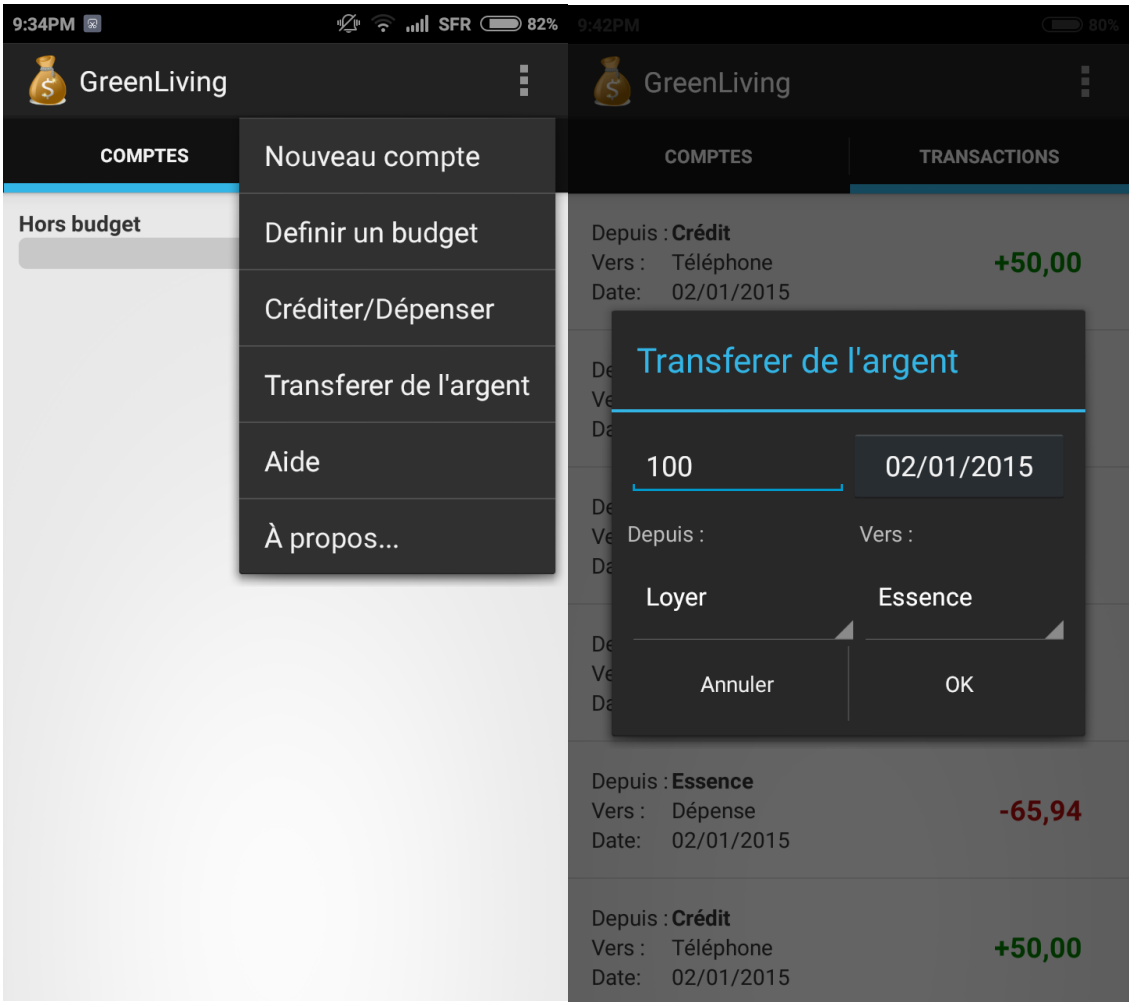


Mockup des contrôles de l'application

Pour obtenir les résultats espérés lors de l'implantation nous avons choisi d'utiliser une Activité qui contient deux Fragments. Il est possible de swiper d'un Fragment à l'autre, et le nom de ces derniers est affiché dans l'ActionBar.

Activité principale avec les deux Fragments et les onglets dans l'ActionBar

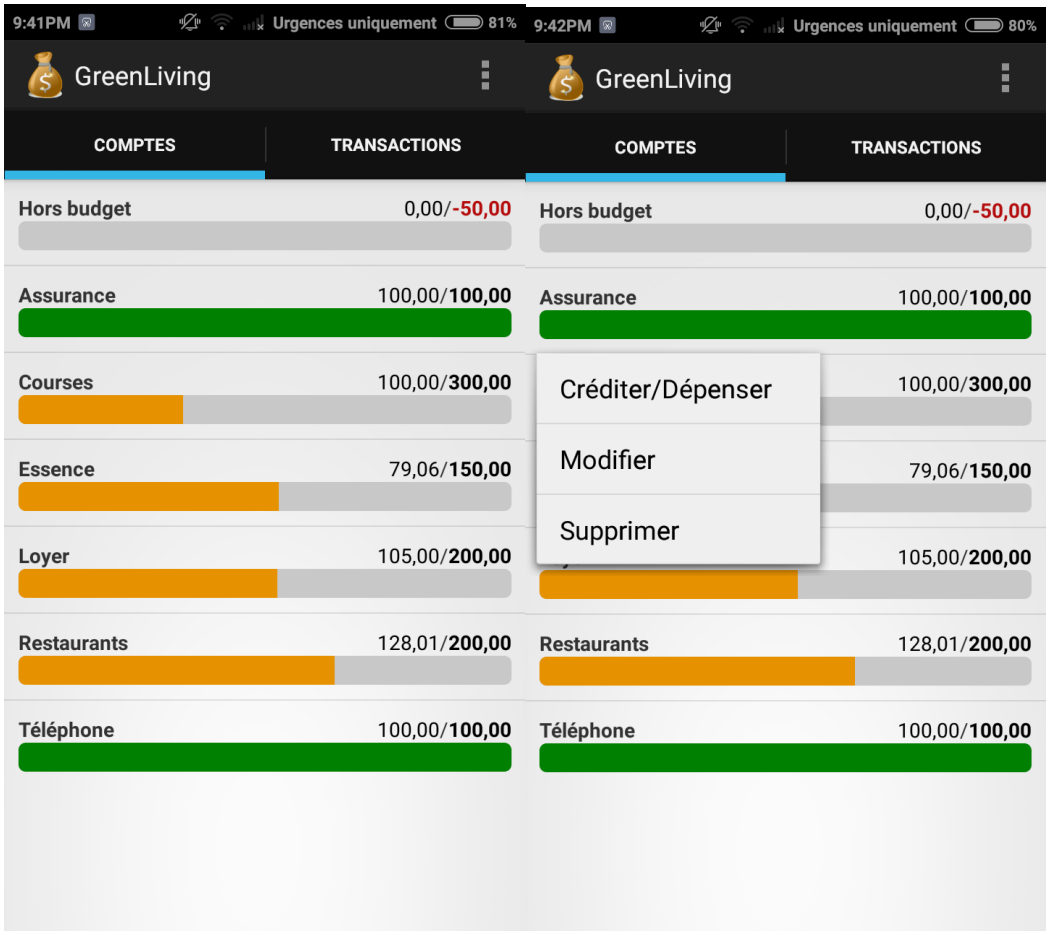
Chaque Fragment a un menu qui lui est propre, accessible depuis le bouton à droite de l'ActionBar et depuis le bouton Menu du téléphone. Il s'agit d'un simple Menu Android avec des Items rassemblés dans des Groupes. Les contrôles accessibles depuis ces Menus sont des Activités avec le thème "Theme.Holo.Dialog". Ceci permet de les afficher comme des Dialogs.



Menu du Fragment Comptes et Activité de transfert d'argent affichée avec le thème Holo.Dialog

Dans la mesure du possible nous avons utilisé des LinearLayout pour le design des interfaces graphiques. Malgré tous les nouveaux Layouts qui sont apparus avec les versions récentes du SDK, nous considérons que le LinearLayout reste toujours celui qui s'adapte le mieux aux différentes tailles d'écran des smartphones (problème bien typique d'Android).

Dans chaque Fragment nous utilisons une ListView pour afficher les objets que nous voulons. Pour faire cela, nous utilisons un Adapter et avons créé un Layout qui représente un élément de la liste. Sur le fragment des comptes, le clic court sur un élément de la liste permet d'afficher uniquement les transactions en rapport avec ce compte. Le clic long permet de modifier l'élément sélectionné. Ceci est fait en implantant les interfaces OnClickListener et OnItemLongClickListener. Sur le fragment des transactions seul le clic long est utilisé : il permet de modifier ou de supprimer une transaction. Le clic long affiche un PopupMenu qui fonctionne d'une manière similaire au Menu classique.



Fragment Comptes : ListView et PopupMenu après un clic long sur un élément

Toutes les chaines de caractères ont été définies dans deux fichiers *strings.xml* afin de pouvoir facilement génér les langues. L'un d'eux a été rédigé en anglais et placé dans le dossier *res/values* ce qui en fait la langue par défaut. Le second a été rédigé en français et se situe dans le dossier *res/values-fr*. Ainsi si l'application est exécutée sur un terminal paramétré en français l'application sera automatiquement intégralement traduite.

Afin de permettre à l'utilisateur de facilement identifier notre application, nous avons utilisé un logo trouvé sur internet (distribué avec la licence creative commons CC-0) qui permet de bien illustrer l'utilité de l'application.

Problèmes rencontrés

Nous avons eu l'occasion de nous confronter à plusieurs problèmes aussi divers que variés.

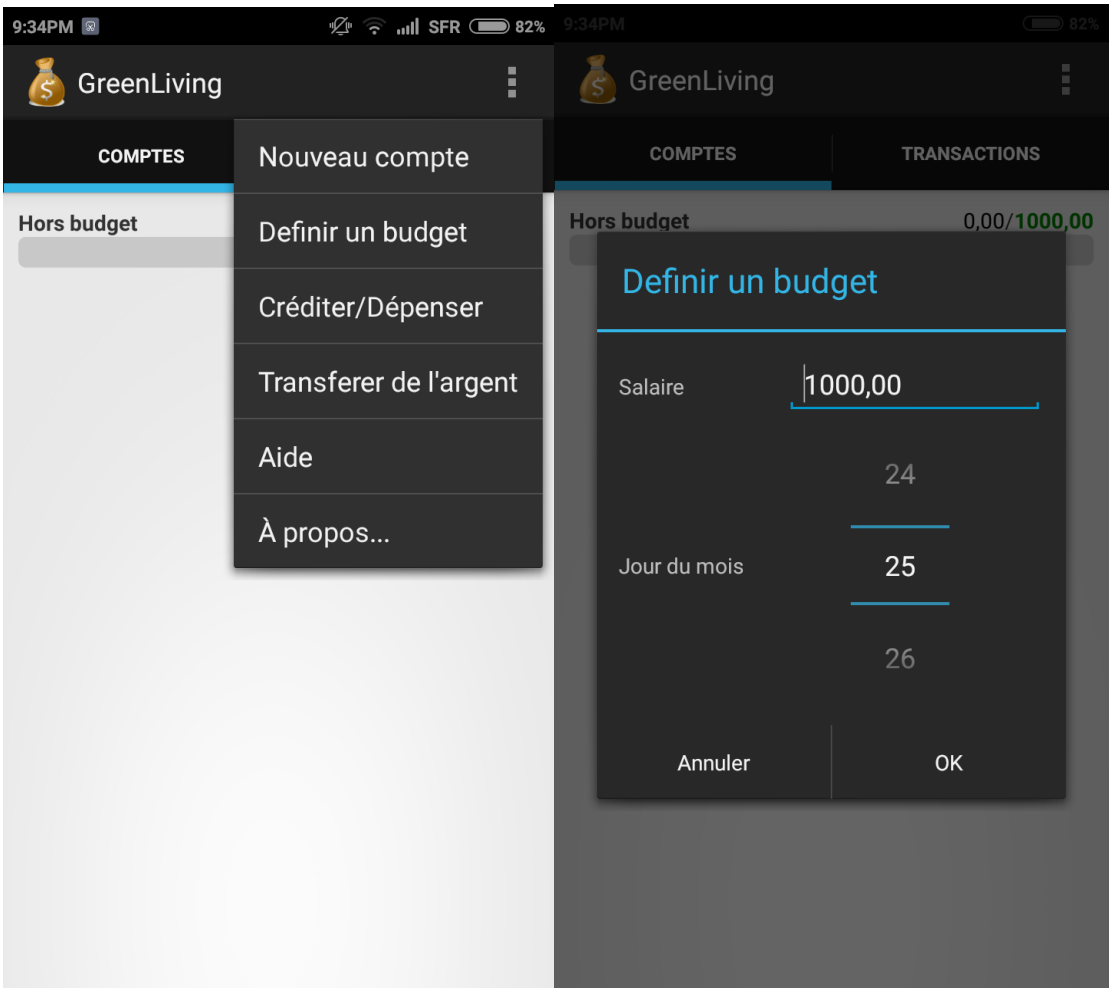
Notre plus grosse erreur fut certainement de sous-estimer le temps de développement. En effet, si ce projet est appelé "mini-projet", il n'en est rien. Entre le développement du modèle, de la couche de persistance, de l'interface graphique et la résolution de bugs, le nombre d'heures de développement a été beaucoup plus conséquent que nous l'avions estimé au début.

Les fragments ont notamment été particulièrement délicats à implanter et à stabiliser à cause de leur cycle de vie. Nous avons été confrontés de nombreuses fois à des *NullPointerException* lorsque nous avons implanté les fragments. De plus la couche de persistance est particulièrement complexe. Lorsque la sauvegarde du modèle a fonctionné, nous avons eu affaire à plusieurs bugs qui ne se produisaient qu'après le chargement d'un modèle existant.

Mais ces problèmes ont tous pu être surmontés, ont été motivant à résoudre et nous ont permis d'en apprendre plus sur le développement Android.

Mode d'emploi

Pour définir son budget mensuel, il faut se placer dans l'onglet comptes et choisir : **menu -> définir un budget**. Il sera alors possible de définir quel est le revenu mensuel et quel jour du mois il est gagné.

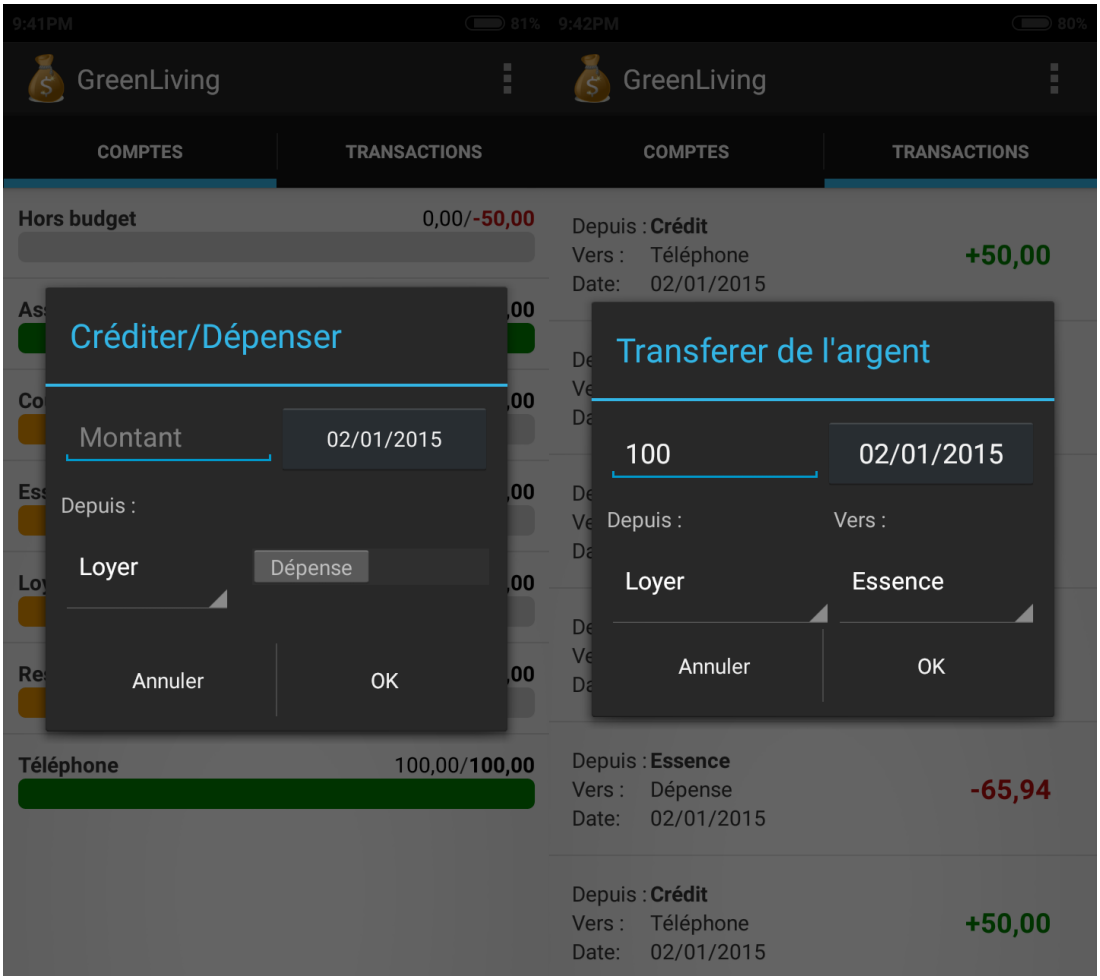


Menu -> Définir un budget

Pour ajouter un compte, il faut se placer dans l'onglet "Comptes" et choisir : **menu -> nouveau compte**. Un dialogue s'ouvrira alors permettant de définir le nom du compte ainsi que la somme d'argent mensuelle à lui allouer. Dès que le compte sera créé, il sera possible de constater que l'argent alloué au compte "Hors budget" aura diminué.

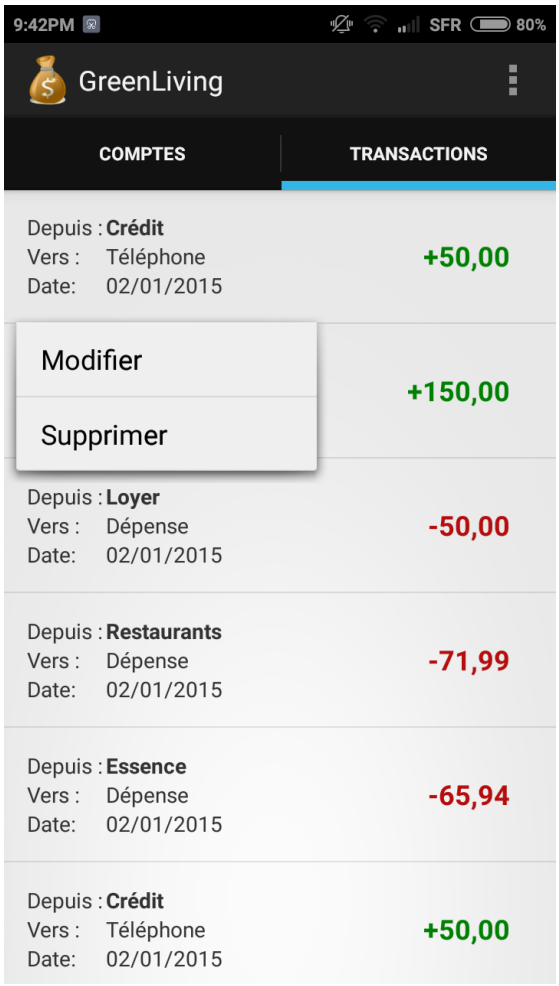
Menu -> Nouveau compte

Depuis les deux onglets (Comptes et Transaction) il est possible d'enregistrer ses transactions en choisissant : **menu -> Créditer/Dépenser**, **menu -> Transférer** ou en effectuant un long clic sur un compte et en choisissant **Créditer/Dépenser.**



menu -> Créditer/Dépenser, **menu -> Transférer**

Il est possible de modifier et de supprimer les comptes comme les transactions en effectuant un long clic dessus et en choisissant : **Modifier** ou **Supprimer.**



Clic long sur une transaction pour la modifier ou la supprimer

Chaque fragment contient un menu **Aide**.

Les comptes se remplissent et se vident tout seuls en fonction des transactions.

Conclusion

Afin de bien travailler ensemble, et ce malgrès la distance qui nous sépare, nous avons beaucoup utilisé *Git* pour le suivi de version*, Skype* pour la communication et *Google Drive* pour l'écriture concurrente du rapport.

Après la conception de l'application, les tâches se sont réparties assez naturellement :

- Jonathan a plutôt travaillé sur la couche de persistance (packages Model et DAO) et sur les ListAdapter de l'application.
- Ludovic a plutôt travaillé sur l'interface graphique de l'application (package ui).

Cependant nous avons beaucoup communiqué et travaillé ensemble sur une grande partie du projet, surtout au moment du débogage.

Ce projet était passionnant du début à la fin, et ce malgré les problèmes que nous avons rencontrés . Nous avons pris du plaisir à développer cette application. Ce fut pour nous une très bonne expérience de travail en équipe.