NIARKORP BROS

UHJvamV0IEphdmEgQXZhbmPp

4ndr01d 63n3r1c 70d0

miaouPlop

akiavara

Application Androïd permettant d’utiliser plusieurs systèmes de gestion de tâches

Table des matières

I. Application développée

a. Choix du sujet

Pour le choix du sujet, nous avions deux possibilités : l'interface homme-machine, gérant la partie visuelle de l'application et la partie « backends », gérant le « moteur » de l'application. Le premier sujet étant surtout réservé aux étudiants en image, notre choix s'est naturellement porté sur la partie « backends ».

b. Choix du système de gestion de tâches

En ce qui concerne le choix du système, nous avions cinq possibilités : GoogleTasks, Todo.ly, Toodledo, Remember the Milk et Orgmode. Les quatre premiers sont composés d'une API[1] basée sur le modèle REST[2], alors que, le dernier, quant à lui, s'utilise par l'intermédiaire d'un système de fichiers.

Utilisant énormément et très souvent Google, nous avons décidé de développer notre application sur ce système. De plus, la facilité d'utilisation de leurs API ainsi que la documentation extrêmement bien faite ont confirmé ce choix.

L'API REST de Google s'utilise comme suit : une URI[3], l'utilisation d'une méthode HTTP (GET, POST, PUT, DELETE) et un format de données d'échanges, ici, le JSON[4]. Ce format est utilisé à la fois par nos requêtes vers le serveur ainsi que par les réponses du serveur à notre application.

[1] « Interface de Programmation », interface permettant l'interaction de programmes les uns avec les autres. Elle est souvent composée par une bibliothèque de fonctions, procédure, classes, etc.
[2] « Representational State Transfer », style d'architecture dans lequel un composant lit ou modifie une ressource en utilisant une représentation de celle-ci.
[3] « Uniform Resource Identifier », courte chaîne de caractère identifiant une ressource sur un réseau dont la syntaxe respecte une norme d'internet.
[4] « JavaScript Object Notation », format générique textuel permettant de représenter une information de façon structurée.

II. Solutions techniques choisies

Afin de respecter le sujet, énormément de solutions techniques étaient envisageables. Nous pouvions soit tout faire nous-même, créer le système de synchronisation, de gestion de compte, etc. ou, utiliser les outils intégrés au système Androïd.

En ce qui nous concerne, nous avons choisi la deuxième solution nous disant qu'il ne sert à rien de réinventer la roue car, la plupart du temps, nos algorithmes et nos solutions seront moins efficaces que les solutions déjà mises en place. Ainsi, toutes les solutions présentées utiliseront des mécanismes présents dans Androïd.

a. La gestion du compte

En ce qui concerne l'utilisation de GoogleTasks, un compte Google est nécessaire. Il est donc impératif de mettre en place une solution technique permettant de lier l'application au compte Google de l'utilisateur.

Le choix le plus simple et le plus évident était d'utiliser une « webview » dans laquelle l'utilisateur saisissait ses informations puis autorisait l'application. Ensuite, il nous aurait fallu demander un nouveau token à Google lorsque le précédent arriverait à expiration. C'est cette solution que nous avions choisie au début.

Après avoir étudié le cas de la synchronisation, nous nous sommes fait la réflexion qu'il serait plus judicieux d'utiliser le système de gestion de compte intégré à Androïd. Ainsi, par l'intermédiaire de la classe *GoogleAccountHandler*, nous vérifions que l'utilisateur possède un compte de type « com.google » (type défini par Androïd pour les comptes Google). Si tel est le cas, alors nous utilisons ce compte et Androïd lance une *Activity* permettant d'autoriser ou refuser de lier l'application avec le compte Google de l'utilisateur une fois pour toutes. Si aucun compte n'existait, Androïd lance une *Activity* qui permet soit de créer un compte Google soit d'en utiliser un existant. Ensuite, le sytème d'exploitation s'occupe lui-même de la gestion de la récupération des *tokens* ainsi que de l'authentification au protocole « oauth ».

b. La synchronisation

En ce qui concerne la synchronisation, ici aussi, nous aurions pu faire notre propre système. Cependant, après beaucoup de recherches et après avoir visionné la vidéo[5] de Virgil Dobjanschi [6]sur le développement d'applications avec architecture REST sur Androïd, nous avons décidé de mettre en place la solution proposée par le système d'exploitation.

Pour cela, nous avions besoin de différents composants : un compte (voir partie précédente), un *ContentProvider* (voir partie suivante) et un *SyncAdapter*.

[5] http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html
[6] Virgil Dobjanschi, développeur Androïd chez Google

1) Syncadapter.xml

Ce fichier XML se trouve dans « res/xml » et comporte les paramètres nécessaires à indiquer au système sur la façon dont fonctionnera la synchronisation. Deux éléments sont obligatoires : l'adresse du *ContentProvider* ainsi que le type de compte à utiliser pour la synchronisation. D'autres paramètres, optionnels cette fois-ci, peuvent être indiqués comme, par exemple, le fait que la synchronisation soit visible à l'utilisateur ou non.

2) SyncService.java

Cette classe doit étendre la classe *Service* du système afin de renvoyer le singleton contenant le *SyncAdapter*. Pour cela, il est nécessaire de réécrire deux fonctions : *onCreate* qui permet de créer la classe de synchronisation et la fonction *onBind* qui permet de renvoyer le service en soi. Ce service doit ensuite être déclaré dans le « AndroidManifest.xml » afin que le système fasse la liaison entre cette classe et le « SyncAdapter.xml ».

3) SyncAdapter.java

Cette classe doit étendre la classe *AbstractThreadedSyncAdapter* et réécrire la méthode *onPerformSync* afin d'être appelée automatiquement par le système dès qu'une synchronisation est en cours. Elle contient aussi toutes les fonctions nécessaires à la mise à jour du système distant ainsi que de la base de données locale. Lors d'une synchronisation, les opérations se déroulent de la façon suivante :

- Appel à la fonction *updateRemote* de la classe *RemoteManager* qui s'occupe de mettre à jour le serveur distant ;
- Une fois cette mise à jour effectuée, appel du *callback*[7] *onFoldersUpdated* qui, lui-même, fait appel à la fonction *getAll* du *RemoteManager* qui permet de récupérer toutes les listes de tâches ainsi que toutes les tâches du serveur distant ;
- Dès que tout a été récupéré, appel du *callback onFoldersReceived* qui fait appel à la fonction *performDiff* qui permet de mettre à jour la base de données locale.

Afin d'être sûr que **toutes** les opérations soient bien réalisées, celles-ci sont faites de manière atomique, c'est-à-dire que si une opération rate, elles sont toutes reprises. D'autre part, pour le système de mise à jour, la méthode des « TAG » est utilisée. Autrement dit, lors d'une insertion locale, un tag « insert » est appliqué à la tâche, lors d'une mise à jour, un tag « update » et lors d'une suppression un tag « delete ». Ceci permet de ne récupérer que les tâches tagguées pour la mise à jour du serveur et de toujours savoir ce qui a été changé. Enfin, le cas de l'insertion d'un tâche est spécial car les identifiants donnés par Androïd ne sont pas les mêmes que ceux du système distant. Ainsi, lorsqu'une tâche insérée dans la base de données locale est insérée sur le serveur distant, elle est tout de suite supprimée en local. Etant donné que l'opération suivante est la récupération de **toutes** les tâches du serveur, la nouvelle tâche sera donc forcément réinsérée sur la base locale avec le bon identifiant.

Nous tenons à ajouter quelques précisions sur la synchronisation : celle-ci est, soit déclenchée par le système, soit déclenchée manuellement à l'aide de la fonction *notifyChange* dont le troisième paramètre précise qu'une synchronisation doit être effectuée. Cependant, ceci ne **fonctionne pas** pour des raisons que nous ignorons.

[7] Fonction ou classe de rappel passée en argument à une fonction

c. La mise en « cache »

Pour le système de mise en cache, nous devions utiliser une base de données SQLite. Afin de faciliter cette utilisation et de respecter le système de synchronisation d'Androïd, nous avons décidé d'utiliser un *ContentProvider*.

Un *ContentProvider* est une classe chargée de « simplifier » l'utilisation de la base de données en fournissant des fonctions basées sur le système « d'URIs ». Lors d'une requête, l'URI donnée en paramètre est analysée. Pour cela, lors de la création de la classe, nous avons utilisé un *UriMatcher* qui permet d'attribuer un *int* à l'URI analysée si celle-ci était enregistrée. Grâce à celui-ci, l'utilisation d'un *switch* permet ensuite de choisir une requête pré-écrite et de lui fournir les paramètres nécessaires à sa bonne réalisation. Ce *ContentProvider* doit ensuite être déclaré dans le « AndroidManifest.xml » afin que le système permette son utilisation avec le système de synchronisation et d'autres applications qui pourraient en avoir besoin.

III. Difficultés rencontrées

Au cours de ce projet, nous avons rencontré énormément de difficultés liées à l'utilisation des systèmes intégrés à Androïd. Dans cette partie, nous allons brièvement décrire les principaux problèmes rencontrés.

a. La gestion du compte

En ce qui concerne la gestion de compte, nous avons pendant longtemps cherché le moyen de créer un compte de type « com.google ». Au départ, nous utilisions la fonction *addAccountExplicitly* qui permettait de créer un compte de son propre type. Cependant, dès que nous essayions de créer un compte de type Google, nous obtenions une erreur. Par la suite, après beaucoup de recherches, nous avons finalement testé de créer un compte par la fonction *addAccount* en mettant le type de compte « com.google » et, contre toutes attentes, la création de compte a fonctionné.

Un deuxième problème s'est alors présenté : la création du compte est effectuée dans un *thread* géré par Androïd. Nous avons donc créé une classe privée dont le seul but est de servir de *callback* à la fonction *addAccount*. Ainsi, tant que le compte n'est pas créé, aucune opération n'est lancée. Nous avons appliqué le même système à la récupération des *tokens*.

b. La synchronisation et la mise en « cache »

Comme pour la partie précédente, l'une des difficultés importante que nous avons rencontrée a été la gestion des *threads* créés par Androïd. Nous avons donc utilisé la même méthode que précédemment en créant des fonctions de *callback*.

De plus, nous avions besoin d'être sûrs que **toutes** les opérations étaient bien effectuées. Nous avons donc utilisé une variable qui nous permet de reprendre les opérations où elles se sont arrêtées afin d'obtenir un système de mises à jour atomique.

Enfin, en ce qui concerne le *ContentProvider*, la difficulté principale a été la compréhension en soi du système. En effet, au début, nous avons eu du mal à saisir le concept des *URIMatcher* qui permet d'analyser les URIs données en paramètres aux fonctions permettant de faire les requêtes.

Conclusion

Pour conclure, nous avons beaucoup apprécié de travailler sur ce projet car il nous a permis de bien mieux comprendre les fondements du système ainsi que de nous améliorer dans la programmation d'applications Androïd.