GRAND CERCLE MOBILE - GCM

Document d'implantation

Luiza CICONE - Jérémy KREIN - Jérémy LUQUET - Paul MAYER
ISI - IF
Tutrice : Gaëlle CALVARY

Mai 2012

Table des matières

1 Conception

1.1 Conception ergonomique

Ici on va mettre le DSE.

Le diagramme suivant représente la modélisation objet du domaine d'application qui guide nos choix de conception.

1.2 Conception logicielle

1.2.1 Architecture logicielle

L'architecture utilisée pour les deux applications est l'architecture Modèle-Vue-Contrôleur qui à été facilement mise en œuvre avec les frameworks Android pour Java et Cocoa Touch pour Objective-C.

L'application réalisée ne peut exister sans que le site du Grand Cercle ne soit tenu à jour. Nous récupérons ainsi toutes les données présentes sur ce site à l'aide de cinq parsers qui agissent sur cinq fichiers xml différents en même temps que l'affichage d'accueil à l'aide de threads.

Plusieurs choses sont ainsi réalisées lors de cette phase :

- récupération des données sur le site
- création des listes de clubs, de cercles et de types d'événements, nécessaire pour la gestion des préférences.
- initialisation de toutes les autres structures de données (listes des données relatives aux événements, aux news, aux bons plans).
- initialisation des préférences (dictionnaire pour iOS et base de donnée pour Android)

Un des points importants de la récupération de données est la présence ou non de connexion internet :

- soit une connexion internet est disponible. Dans ce cas on sauvegarde les fichiers xml en provenance du site dans la mémoire et on parse ces fichiers qui sont alors contenus dans la mémoire interne du téléphone.
- soit aucune connexion n'est possible et dans ce cas, on parse les données qui sont dans la mémoire, sauvegardées lors de connexions antérieures.

Lors de la première utilisation de l'application, un message est affiché à l'utilisateur s'il n'est pas connecté à internet pour lui indiquer qu'aucune donnée n'a pu être chargée (mais l'application se lance correctement). Lors de l'initialisation de la base de données, aucun filtre n'est appliqué et les données sont récupérées dans leur totalité, l'utilisateur ayant ensuite le choix d'appliquer ou non ces filtres.

1.2.2 iOS

Pour Objective-C, les classes sont séparées dans les fichiers header (extension `.h`) et les fichiers d'implémentation (extension `.m`) Les classes qui correspondent au modèle contiennent les attributs et les mutateurs comme dans le modèle objet présenté dans la figure **??**. Les classes de la vue sont les fichiers avec l'extension `.xib` qui sont des fichiers de type `xml`. Ces fichiers peuvent être créés et édités facilement grâce à l'interface graphique fournie par Xcode. Toutes les vues sont contrôlées par des classes `ViewController` qui font la connexion entre les objets et les données des contrôleurs. Les classes du contrôleur sont les classes pour récupérer, parser et filtrer les données.

Patrons de conception

Pour toutes les classes contrôleur nous avons utilisé le `Singleton` pour s'assurer d'avoir une seule instance qui effectue les traitements des données.

Nous avons utilisé aussi quelques autres patrons de conception fournis par le framework et Xcode.

– **Composite** est le patron utilisé pour la hiérarchie des éléments de la vue. Chaque élément est une sous-classe de `UIView` donc il hérite de toutes ses méthodes. En utilisant l'interface graphique, nous avons ajouté des objets à la vue en suivant une certaine hiérarchie

– **Commande** est le patron utilisé pour rediriger l'exécution quand un événement apparaît. Cela a toujours été fait à l'aide de l'interface graphique de Xcode, permettant de relier un objet à une méthode.

– **Observateur** est le patron qui fournit la mise à jour facile entre les vues et les modèles. Nous avons utilisé `Notification Center` pour informer la vue des changements du modèle et vice-versa.

1.2.3 Android

La technologie Android repose sur la dualité entre la langage `Java` et le langage `xml`. A chaque affichage correspond un fichier `xml` qui met en place les différentes fenêtres qui constituent l'écran, appelées des « layouts », ainsi qu'un fichier Java qui représente ce qu'on appelle une activité, c'est à dire la fenêtre visible à l'écran. La vue offerte à l'utilisateur est mise en place par la méthode setContentView([fichier xml]) dans le code Java. Néanmoins, le fichier xml en paramètre de cette méthode correspond à une configuration statique de l'affichage. Ainsi, dès que nous avons besoin de modifier dynamiquement une vue, nous le faisons dans le code Java, grâce à l'identifiant de la vue récupéré dans le code xml : c'est ce qui fait la dualité Java/xml. Étant donné que le positionnement des différents objets dans les layouts est moins facile en Java que dans un fichier xml, il est avantageux d'utiliser le xml dès que cela est possible.

Patrons de conception

– **Singleton** a été mis en œuvre pour notre base de donnée, afin de s'assurer qu'elle n'est instanciée qu'une seule fois et ainsi d'assurer un traitement des données cohérent pour toute l'application.

– **Commande** est le patron utilisé pour rediriger l'exécution quand un événement apparaît (exemple : appui sur une zone de l'écran, sur un bouton, ...)

– **Observateur** est utilisé lorsque les préférences sont modifiées par l'utilisateur. Il se résume a un seul appel à la méthode `ParseEvent()` afin d'actualiser les données affichées et stockées, étant donné que les données qui ne correspondent pas aux préférences n'ont pas de raison d'être stockées dans les structures de données appropriées.
– **Fabrique** est nécéssaire pour construire une instance du parser SAX qui permet de récupérer les données.

Dans l'application Android, nous avons préféré utilisé un parser SAX plutôt qu'un DOM, puisque le SAX est plus rapide que le DOM (cf http ://www.developer.com/ws/android/development-tools/Android-XML-Parser-Performance-3824221-2.htm).

Notre application témoigne d'une volonté de simplicité et de rapidité auprès des utilisateurs. Ainsi, la classe `DragableSpace.java` permet de « slider » afin de limiter le nombre d'onglets. Le changement de vue n'est réalisé que si l'utilisateur effectue le slide assez rapidement.

2 Mise en œuvre

2.1 iOS

Parser Nous avons utilisé un thread pour parser les informations de l'`XML` pour ne pas bloquer l'interface utilisateur. Quand l'information est disponible une notification est envoyé et la mise à jour de l'affichage est faite.

Préférences

Pour sauvegarder les préférences nous avons utilisé le dictionnaire par défaut de l'application `[NSUserDefaults standardUserDefaults]`. A la clé choisie il y un pointer vers : un dictionnaire pour les filtres ou un vecteur pour le choix de thème. Pour le filtre de cercle, club et type nous avons un dictionnaire des booléens où la clé est le nom du cercle, du club ou du type. La couleur du thème est sauvegardé en RGB dans un vecteur de taille 3.

Cache des images

Chaque fois qu'on trouve une nouvelle image à télécharger, on la met dans une queue et on lasauvegarde dans la mémoire lorsqu'elle est disponible. On garde un dictionnaire ayant comme clé le hash(signature des données) de l'URL de l'image. Le téléchargement est fait dans un thread et une notification est envoyée pour mettre à jour l'affichage.

Librairies externes

Pour parser l'information provenant du fichier `XML` nous avons utilisé la librairie `TBXML` qui d'après les critiques est le plus rapide et le plus adapté a nos besoins. De plus, une classe avec des extensions pour `NSString` est utilisé pour décoder les entités `HTML`

On se sert de la librairie `tapuku` pour la gestion du cache des images et pour le calendrier. Des changements ont étés fait au calendrier pour factoriser le code et pouvoir contrôler la taille du calendrier facilement. Une simple classe appellé `Reachability` offre les fonctionnalités pour tester la connexion internet de l'utilisateur et d'agir en conséquence.

2.2 Android

Parser Nous avons utilisé un thread pour parser les informations de l'`XML` pour ne pas bloquer l'interface utilisateur. Quand les cinq parser ont terminé leur execution, la méthode `onPostExecute(Void result)` est appelée, et crée la classe `GCM`. La différence avec les autres activités est que `GCM` étend la classe `TabActivity` et non `Activity`. En effet, cette activité permet de construire les onglets de notre application à l'aide d'un objet appelé `TabHost`. Toutes les vues des différents onglets sont crées dans GCM.java, ce qui permet à l'utilisateur de naviguer rapidement entre les différents onglets.

Préférences Pour sauvegarder les préférences, une base de donnée `SQLite` a été employée. Chaque préférence (par cercle, par club, par type et le thème) est stockée dans une table différente.

Cache des images

A chaque image trouvée par le parser, on associe une clé de type `String` qui est en fait l'url de l'image. On stocke ensuite dans une `HashMap` le fichier bitmap correspondant à cette clé.
Deux conventions ont été adoptée pour le stockage des images :

- soit l'image est une affiche d'événement et elle est stockée dans le cache pendant une semaine
- soit l'image est un logo d'association ou d'enseigne (pour les bons plans) et elle est stockée de manière infinie dans le cache

Librairies externes

Afin de pourvoir réaliser la fonctionnalité d'import d'un événement au calendrier du téléphone, nous avons été obligé d'ajouter à nos options de compilation la librairie `android-8`, issue de l'API 8 d'Android. Notre application doit fonctionner sur les téléphones équipés des API 7 et plus, donc cette librairie est inclue dans les API 8 et plus, mais posait un problème pour l'API 7. Nous l'avons donc rajouté lors de la compilation, et elle fonctionne sur l'API 7 (téléphones sous Android 2.1).

3 Validation

Les deux applications ont été validées suivant une démarche commune afin d'assurer un niveau de qualité équivalent d'une application à l'autre.

Certains tests ont été réalisés au fur et à mesure du développement des applications, d'autres ont été

effectués sur des prototypes fonctionnels et enfin la version finale des deux application a été évaluée par un panel d'utilisateurs potentiels.

3.1 Tests unitaires

Chaque module a été testé grâce à des tests unitaires. Majoritairement, ces tests consistent à envoyer en entrée du module un cas de test spécifique, de tracer le module afin de vérifier qu'au cours de l'exécution rien d'anormal ne se produit, et de récupérer en sortie un résultat que l'on compare au résultat attendu. Les tests spécifiques sont pour la plupart des tests dits "boite blanche".

Ces tests sont avant tout des tests techniques qui permettent de s'assurer qu'il n'y a pas d'erreur d'analyse et/ou de programmation.

3.2 Tests d'intégration et de validation informatique

Ces tests permettent de tester la cohérence et l'articulation des modules entre eux, de vérifier que les modules communiquent bien entre eux.

Ces tests ont été réalisés de manière similaire aux tests unitaires, à savoir un traçage des opérations effectuées et une comparaison du résultat obtenu avec le résultat attendu.

3.3 Validation ergonomique

Afin de vérifier la conformité de nos différents prototypes avec les besoins formulés par les utilisateurs, regroupés dans le cahier des charges, nous avons fait appel à un panel d'utilisateurs potentiels disposant de leur propre smartphone sous iOS ou Android, et ce dans le but de confronter ces applications à des utilisateurs habitués aux pratiques sur ces deux systèmes d'exploitation.

Nous avons mis en place des une procédure qui nous a permis de toujours demander à ces utilisateurs potentiels d'effectuer les mêmes manipulations dans l'application et d'observer leur réactions. Pour prendre en compte leurs remarques et ne pas en oublier, nous avons procédé à des enregistrements vidéo de ces tests.
Pour enrichir l'application, nous avons créé une URL de test sur le site du `Grand Cercle` afin de faire afficher un grand nombre d'événements dans l'application et de rendre le test plus réaliste.

Le protocole est fourni en annexe, il reprend la totalité des fonctionnalités implémentée dans notre application, et comprend des manipulations très simple pour commencer ainsi que des tâches jugées plus difficile à réaliser pour finir.

A Protocole de tests utilisés dans les tests utilisateurs

1. Lance l'application Grand Cercle Mobile.

2. Dis moi quels sont les 4 prochains événements qui se dérouleront à Grenoble INP.

3. Donne moi les informations suivantes sur les 2 prochains événements : date, association qui organise, lieu, heure, type d'événement.

4. Y a-t-il un événement le 15 Novembre ?

5. Trouves un jour en Novembre où il y a un événement ?

6. Trouve l'événement le plus éloigné en date ?

7. Ajoute le au calendrier de ton téléphone

8. Imaginons que tu sois intéressé(e) par les événements de ton cercle et les Kfet de toutes les autres associations, fais en sorte d'afficher seulement les événements qui t'intéresse.

9. Quelles sont les dernières nouvelles concernant les élus étudiants ?

10. Quand a été publié la dernière nouveauté concernant le Grand Cercle ?

11. Cite moi trois enseignes dans lesquelles tu peux bénéficier de réductions en tant qu' étudiant de Grenoble INP.

12. Quelles sont les offres concernant la Fnac ?

13. Trouve moi le lien de la page internet contenant les contacts du Grand Cercle.

14. Configure l'application aux couleurs de ton école ?

15. As tu des remarques, des choses à dire concernant l'application ?