RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

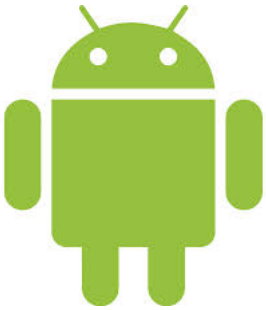
Ministère de l'enseignement supérieur et de la recherche scientifique

Université Amar Telidgi - Laghouat

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUE ET D'INFORMATIQUE

Mémoire en vu de l'obtention du diplôme de licence en informatique



RÉALISATION D'UNE INTERFACE D'INTERROGATION DE BASES DE DONNÉES DEPUIS ANDROID

Présenté et soutenue par

Rouane Oussama

Mémoire dirigé par

Guellouma Younes

Juin 2013


بسم الله الرحمن الرحيم

Remerciements

Je remercie Allah de m'avoir donnée volonté et m'aider afin d'achever ce travaille ...

Je tiens à remercier toutes les personnes qui m'ont aidé pour la réalisation de ce travail ...

Je tiens à remercier aussi :

Monsieur : Bouzidi Mohamed Reda
pour avoir accepté de présider le jury de ma soutenance ...

Mon encadreur : Guellouma Younes
pour ses qualités humaines et professionnelles, pour son encadrement, ses directives, ses remarques constructives, et sa disponibilité ...

Un grand merci à tous mes enseignants, sans eux ce travail n'aura jamais vu le jour ...

Rouane Oussama.

Dédicace ...

Je dédie ce travaille à ...

Mes parents

Mon frère et Mes sœurs

Ma grande famille...

Et tous mes amis ...

Table des matières

Liste des figures

Introduction général

Les progrès conjoints de la micro-électronique, des technologies de transmission sans fil et des applications embarquées ont permis de produire à coût raisonnable des terminaux mobiles de haute technologie comme les Smartphones et les tablettes PC ...

Actuellement la société Apple à travers son Smartphone « iPhone», sa tablette PC « iPad » et son système d'exploitation « iPhone OS » est en forte concurrence avec la communauté Open Handset Alliance (OHA) qui englobe Google, Motorola, HTC, Samsung, etc. Cette dernière équipe ses terminaux mobiles par le système d'exploitation mobile « Android OS ». Cette concurrence a stagné l'évolution des téléphones.

L'intérêt de ce mémoire est d'étudier de plus près un des systèmes d'exploitation embarqué les plus populaire, Android. Le but de cette étude est double, elle permet d'une part de mieux comprendre l'architecture logicielle de ce type de systèmes et d'autre part de développer une petite application Android qui permet de requêter une base de données externe, un programme client émet des requêtes d'interrogations SQL à un serveur d'applications qui fait lui-même l'accès à un SGBD qui fournit la base de données.

Ce présent mémoire sera structuré en 3 chapitres :

Le *premier chapitre* intitulé **« Notions des bases de données »** : on va commencer par un petit rappel concernant les bases de données, ces critères, on va voir les systèmes de gestion de bases de données, ces objectifs et ces fonctionnalités,puis, on va voir quelques notions nécessaire concernant les bases de données relationnelle. ...

Le *deuxième chapitre* intitulé **« La plateforme Android »** : est consacré à l'introduction du système d'exploitation Android, dans cette partie on va donner une brève définition des différents versions de ce système, ces fonctionnalités et enfin son architecture logicielle ...

Le *troisième chapitre* intitulé **« La programmation sous Android »** : on va voir les étapes d'installation de l'environnement de développement Android, ensuite, on s'intéressera à la conception et à la réalisation d'une petite application Android qui permet de requêter une base de données à distant ...

Enfin, on clôtura ce mémoire par une conclusion générale ...

Notions des bases de données

1

1.1 Introduction

Ce chapitre définit et donne un commentaire sur certains concepts nécessaires utiles dans les notions des bases de données. Les données permanentes constituent certainement le matériau de base à partir du quel l'utilisateur va élaborer la plupart de ses applications. Certaines applications ses réduisent à gérer et consulter les données ; ces données sont rangées dans des fichiers et sont structurées en enregistrement.

Mais si la masse et la complexité de données à traiter deviennent importantes, il est nécessaire que ces données aient aussi une structure complexe. Les données sont classées dans plusieurs fichiers en fonction d'objets qu'elles décrivent. Il existe entre les fichiers des liens qui sont à l'image des relations entre objets décrits il apparait aussi rapidement que les données nécessaires à une application pourrait être utiles pour d'autres applications voire à d'autres utilisateurs. Ces données constituent alors ce qu'on appelle une BD.

1.2 Base de données

1.2.1 Définition

Une Base de données est un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique.

Une Base de donnée peut être aussi pour certains, une collection de fichiers relies par des pointeurs multiples, aussi cohérents entre eux que possible, organisés de manière à répondre efficacement à une grande variété de questions.

Une Base de données est donc un ensemble structuré des données accessibles par l'ordinateur pour satisfaire plusieurs utilisateurs simultanément au temps opportun.

1.2.2 Les critères d'une base de données

Une Base de données doit répondre aux critères suivants :

- **Exhaustivité :** implique que l'on dispose de toutes les informations relatives au sujet donné.
- **La non redondance :** implique l'unicité des informations dans la base de données .En général on essaie d'éviter la duplication des données car cela pose des problèmes de cohérence lors des mises à jour de ces données.
- **La structure :** implique l'adaptation du mode de stockage des renseignements aux traitements qui les exploiterons et les mettrons a jour ; ainsi qu'au cout de stockage de ces renseignements dans l'ordinateur.

Le stockage physique d'une base de données consiste en un ensemble d'enregistrements physiques. Organisés à l'aide des listes, des pointeurs et différentes méthodes d'indexation.

1.3 Système de gestion de base de données

1.3.1 Définition

Les données stockées dans des bases de données modélisent des objets du monde réel, ou des associations entre objets. Les objets sont en général représentés par des articles de fichiers, alors que les associations correspondent naturellement à des liens entre articles. Les données peuvent donc être vues comme un ensemble de fichiers par des pointeurs ; elles sont interrogées et mise à jour par des programmes d'application écrits par les utilisateurs ou par des programmes utilitaires fournis avec le SGBD.

Un SGBD peut être défini comme un langage (logiciel) qui sert à interagir avec une base de données et qui permet à l'utilisateur de définir les données de la base, de les consulter et de les mettre à jour.

Georges Gardarin, définit un SGBD comme étant un ensemble de logiciels systèmes permettant de stocker et d'informer un ensemble de fichiers, interdépendants, mais aussi comme un outil permettant de modéliser et gérer les données d'une entreprise[GAR] .

1.3.2 Les objectifs et fonctions d'un SGBD

Fonction d'un SGBD

Un SGBD permet de décrier les données de bases, de les interroger, de les mettre à jour, de transformer des représentations de données, d'assurer le contrôle d'intégrité, d'occurrence et de sécurité.

Les objectifs d'un SGBD

1. Indépendance physique : un SGBD offre la facilite de changer le schéma interne sans changer le programme d'application. Cela signifie que le niveau physique peut être modifie indépendamment du niveau conceptuel.

2. Indépendance logique : ici le SGBD permet de modifier schéma conceptuel sans changer le programme d'application. Donc le niveau conceptuel peut être modifie sans remettre en cause le niveau physique.
3. La manipulation de données par des langages non procéduraux : le SGBD doit permettre interrogation et la mise à jour de données par des langages de haut niveau spécifiant les données que l'on veut traiter (de quoi) et non pas comment y accéder.
4. La facilité d'administration : les langages de haut niveau référencent des descriptions logiques des données (schéma externes) stockées dans le dictionnaire de données pour permettre la création et modification de la description.
5. Fiabilité des données : le SGBD permet de vérifier les contraintes des données (intégrité référentielle, reflexes, etc.) ; gérer des transactions (atomicité des transactions) et sécurité (mot de passe, etc.) ; récupérer des données en cas de crash logiciel, OS (Operating System) ou disque.

1.4 Base de données relationnelle

Les BDDR sont conçues à partir du modèle relationnel, elles sont d'une grande importance du fait de leur popularité au sein de la recherche et de l'information de gestion. Le succès des BDDR tient essentiellement à leur simplicité. Elles ne contiennent qu'une seule structure de données : tables, avec des lignes et des colonnes et les relations reliant ces tables.

Hainant, Jean – Luc, lui définit une BDDR comme une collection de tables des données ou fichiers plats, une structure extrêmement simple et intuitive qui , pour l'utilisateur du moins ne s'encombre d'aucun détails techniques concernant les mécanismes de stockages sur disque et d'accès aux données[LUC].

Table : est une entité qui contient (une suite de lignes stockées sur un support externe. Elle peut être définie comme étant un groupe de propriétés, reflet d'un objet présentant un intérêt pour le système étudie dotée d'une existence propre, et identifiable.

Ligne (tuple ou r – replets) de la table : est une suite de (une ou) plusieurs valeurs, chacune étant d'un type détermine. D'une manière générale, une ligne regroupe des informations concernant un objet (entité), un individu, un évènement, etc. dans une BDDR, tous les signes présents dans une table ont le même format ou la même structure.

Colonne (attribut) de la table : est l'ensemble des valeurs de même type correspondant à une même propriété des entités décrites ; ces colonnes jouent des rôles différents vis-à-vis des entités représentées par les lignes d'une table. Une colonne donnée joue le rôle d'un identifiant, d'une clé étrangère, d'une information complémentaire etc.

Identifiant : c'est une colonne choisie pour identifier une entité et aussi la ligne qui la représente dans la table c'est à dire qu'elle distingue sans ambigüité l'occurrence d'un objet.

Clé étrangère : c'est une colonne constitue de l'identifiant d'une autre table et joue un rôle de référence a une ligne de cette table ; on l'appelle ' colonne de référence ' ou 'clé complémentaire sur l'entité'.

La relation : est encore appelée table relationnelle ou table.

1.5 Conclusion

On a vu dans ce chapitre quelques notions théoriques sur les bases de données et les bases de données relationnels ,ces critères et surtout le système de gestion des bases de données ,ses fonctions et ses objectifs, à savoir le système Android dans le deuxième chapitre.

La plateforme Android

2

2.1 Introduction

On présentera dans ce chapitre une description du système d'exploitation Android, son historique, ses fonctionnalités et finalement son architecture[MUR].

2.2 Description

Android est un système d'exploitation open-source pour smartphones, PDA et autres terminaux mobiles, conçu par Android, une start-up rachetée par Google en juillet 2005. Il existe d'autres types d'appareils possédant ce système d'exploitation tels que les téléviseurs et les tablettes.

Afin de promouvoir ce nouveau système d'exploitation ouvert, Google a su fédérer autour de lui un consortium d'une trentaine d'entreprises : l'Open Handset Alliance (OHA) créée officiellement le 5 novembre 2007. Toutes ces entreprises interviennent, plus ou moins directement, dans le marché de la téléphonie mobile.

Le but de cette alliance est de mettre en place des normes ouvertes dans le domaine de la téléphonie mobile. Ce qui veut dire que les développeurs d'application Android pourront accéder aux fonctionnalités du cœur de téléphone via une API très fournie.

Figure 2.1 – *Open Handset Allience*

Android aura comme principaux concurrents Apple avec l'iPhone, Microsoft et son Windows Mobile et Nokia avec Symbian mais également des solutions libres telles que LIMO ou OpenMoko.

2.3 Historique

En juillet 2005, Google a acquis Android Inc, une petite startup qui développait des applications pour téléphones mobiles .C'est à ce moment-là que des rumeurs sur l'entrée de Google dans le secteur du mobile ont commencé. Mais personne n'avait des données sûres à propos des marchés dans lesquels ils allaient se positionner.

Après ce rachat fait par Google, une équipe dirigée par Andy Rubin, un ancien d'Android Inc, a commencé à travailler sur un système d'exploitation pour appareil mobile basé sur Linux. Durant 2 ans, avant que l'OHA soit créé officiellement, un certain nombre de rumeurs ont circulé au sujet de Google. Il a été dit que Google développait des applications mobiles de son moteur de recherche, qu'elle développait un nouveau téléphone mobile, etc.

En 2007, le 5 novembre, l'OHA a été officiellement annoncée, ainsi que son but : développer des standards open sources pour appareil mobile. Le premier standard annoncé a été Android, une plateforme pour appareils mobiles basée sur un kernel Linux 2.6.

En octobre 2008, apparait la première version commerciale d'Android 1.0 sur le HTC Dream qui n'avait pas reçu de nom. Cette version s'est avérée être la B du système.

FIGURE 2.2 – *Les versions de systeme Android*

La version 1.5 Cupcake corrigea le manque d'API [1] et rendit le système plus utilisable. Depuis, Android 1.6, 2.0 et 2.1 ont apporté d'importantes améliorations respectivement sur les fonctionnalités et sur l'interface graphique du système.

Android 2.2 Froyo a fortement mis l'accent sur la synergie avec Internet. L'envoi d'applications et de liens instantanés depuis un ordinateur est désormais possible. Aussi, Google annonce-t-elle que le navigateur chrome

1. Application programming interface

intégré à Android 2.2 est le navigateur mobile le plus rapide au monde grâce à l'intégration du moteur JavaScript V8.

Android 3.0 Honeycomb est spécialement étudié pour les tablettes tactiles. Les premiers modèles devraient être annoncés au 2011.

On y apprend quelques nouveautés comme la prise en charge de la vidéo-conférence via Gtalk, la nouvelle interface Gmail ou encore le lecteur de livre électronique Google. La refonte graphique de l'interface utilisateur est assez réussie, plus d'informations devraient suivre dont surement des éclaircissements sur l'intégration ou non de l'interface de cette version d'Android sur les futurs smartphones.

Android 4.0 Ice Cream Sandwich devrait arriver très vite (mi 2011) pour rajouter encore plus de fonctionnalités aux terminaux. Pour le développement, ces nouvelles versions d'Android devraient proposer de nouveaux composants permettant de réaliser des applications avec une ergonomie plus adaptée aux tablettes tactiles.

Android 3.0 et Android 4.0 devraient apporter plus d'outils aux constructeurs leur permettant de proposer des tablettes tactiles, qui seront capables de rivaliser (surtout au niveau de l'ergonomie) avec Ipad.

Android 4.1 et 4.2 Jelly Bean dont la principale nouveauté est une amélioration des fonctionnalités et des performances de l'interface utilisateur. Les améliorations sur le taux de rafraîchissement de l'écran porté à 60 FPS afin de créer l'impression que l'interface est plus fluide que sous Ice Cream Sandwich.

2.4 Bugdroid

Le personnage nommé Bugdroid est le petit robot vert utilisé par Google pour présenter Android. Ce personnage est sous licence « creative commons by (3.0) » et peut donc être utilisé librement.

Le site Engadget annonce que Bugdroid, le logo d'Android, serait en fait un personnage d'un jeu des années 1990 sur Atari : Gauntlet : The Third Encounter.

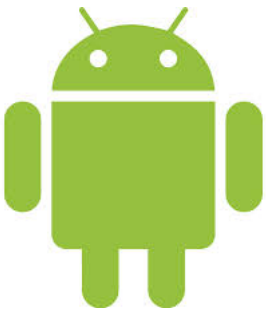


Figure 2.3 – *Le logo d'Android*

2.5 Fonctionnalités d'Android

Android a été conçu pour intégrer au mieux les applications existantes de Google comme le service de courrier Gmail, l'agenda Google Calendar ou encore la cartographie Google Maps.

Voici quelques fonctionnalités proposées par Android classées par version :

2.5.1 Android version 1.5 (Cupcake)

FIGURE 2.4 – *Android version 1.5*

1. Enregistrement et lecture des vidéos.
2. Mise en ligne directe des vidéos sur YouTube.
3. Mise en ligne directe des photos Picasa.
4. Prise en charge du Bluetooth A2DP.
5. Dossiers dynamiques et widgets pour le home.
6. Copier/coller étendu aux pages web.
7. Nouvelle version du clavier virtuel.

2.5.2 Android version 1.6 (Donut)

FIGURE 2.5 – *Android version 1.6*

1. L'application Galerie permet d'effacer plusieurs photos à la fois.
2. Amélioration de l'Android Market.
3. Amélioration de la vitesse de la recherche vocale et intégration étendue à plus d'applications natives.
4. Prise en charge sur une seule application de la prise de photo et de l'enregistrement vidéo.
5. Possibilité de rechercher simultanément dans les favoris, les historiques, les contacts et sur Google depuis le home via le widget recherche.
6. Moteur Text-to-speech.
7. Prise en charge de plusieurs résolutions d'écran.

2.5.3 Android version 2.0/2.1 (Éclair)

FIGURE 2.6 – *Android version 2.0/2.1*

1. Interface utilisateur revue (lock screen) et lanceur d'application.
2. Fonds d'écran animés.
3. New browser interface avec prise en charge du HTML5.
4. Prise en charge du protocole Microsoft Exchange.
5. New contact lists.
6. Prise en charge en natif du flash et du zoom numérique pour des appareils photos.
7. Gestion multi-comptes Gmail et ajout de la synchronisation avec Facebook.

2.5.4 Android version 2.2 (Froyo)

FIGURE 2.7 – *Android version 2.2*

1. Augmentation de la performance et de la vitesse.
2. Fonctionnalité de Hot spot Wifi.
3. Partage de contact sur bluetooth.
4. Mise à jour automatique des applications.

2.5.5 Android version 2.3 (Gingerbread)

FIGURE 2.8 – *Android version 2.3*

1. Support des grands écrans à résolutions extra-larges (WXVGA et plus).
2. Nouveaux effets audio tels que la réverbération, l'égalisation, la virtualisation du casque audio et accentuation des graves.
3. Support de nouveaux capteurs (comme le gyroscope et le baromètre).
4. Ajout d'un gestionnaire de téléchargement.

2.5.6 Android version 3.0 (Honeycomb)

FIGURE 2.9 – *Android version 3.0*

1. Interface entièrement optimisé pour les tablettes tactile.
2. Support des processeurs multi-corps.
3. Possibilité de crypter les données de l'utilisateur.
4. Des nouvelles Api pour prendre en compte des périphériques USB par les terminaux Android (Pour permettre de développer par exemple une application permettant de transformer une tablette en télécommande universelle grâce à un émetteur infrarouge USB. Support des manettes, support de clavier et souris USB).

2.5.7 Android version 4.0 (Ice Cream Sandwich)

FIGURE 2.10 – *Android version 4.0*

1. Boutons virtuels intégrées à l'interface, en remplacement des boutons physique, ces derniers peuvent être maintenus selon les choix du constructeur.

2. Un nouveau launcher personnalisable.
3. Amélioration de la reconnaissance vocale.
4. Déverrouillage par reconnaissance faciale, si appareil équipé d'une camera frontale.
5. Ajout d'Android Beam, une application permettant l'échange d'informations (favoris, contacts, YouTube, vidéos ...) par NFC [2] (si appareil équipé de puce NFC).
6. Enregistrement vidéo en 1080p.

2.5.8 Android version 4.1/4.2 (Jelly Bean)

FIGURE 2.11 – *Android version 4.1/4.2*

1. Interface utilisateur plus fluide.
2. Recherche vocale hors ligne.
3. Nouvelles fonctionnalités pour les notifications.
4. Application appareil photo amélioré.
5. application Google Search est remplacé par google Now, un assistant intelligent comparable à Siri.
6. Clavier amélioré intégrant la saisie gestuelle ou Swype qui permet de créer des mots en reliant les lettres et non plus uniquement en appuyant sur les touches.

2. Near field communication

2.6 Architecture d'Android

Pour bien comprendre la plateforme Android, on va détailler par la suite l'architecture du système Android. Le portail des développeurs Android nous présente l'architecture du système avec le schéma ci-contre :

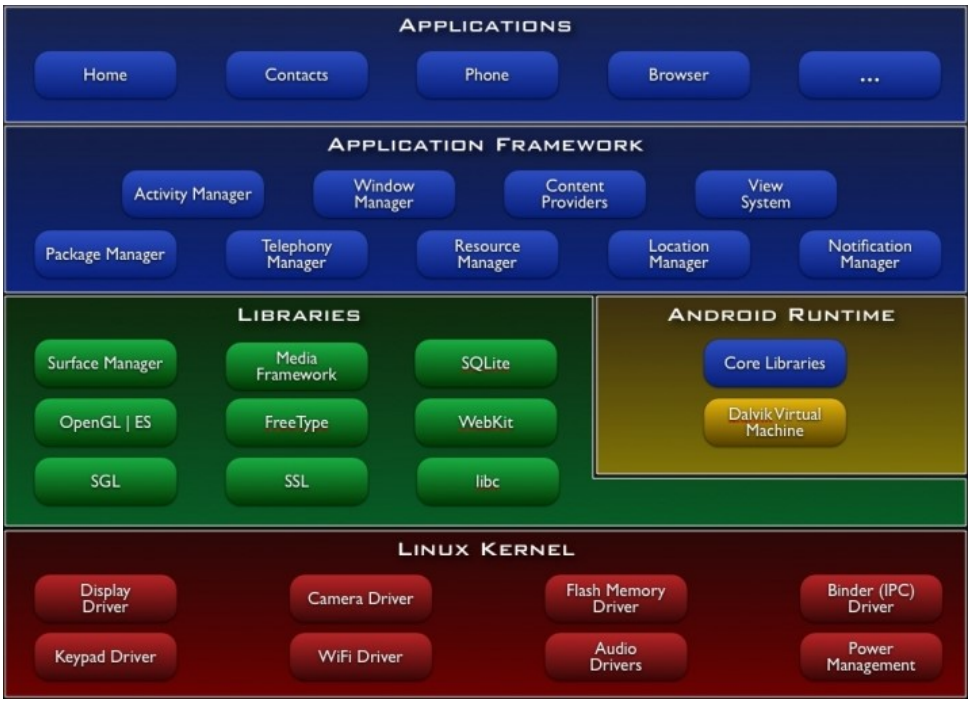


Figure 2.12 – *L'anatomie du systeme Android*

2.6.1 Linux Kernel

Android s'appuie sur le noyau Linux 2.6 pour les services système ,la sécurité, la gestion de la mémoire et des processus, le réseau et la gestion des drivers. Le noyau sert de couche d'abstraction entre le matériel et le reste de la pile logicielle. Android n'est pas linux mais il est basé sur un kernel linux. Pourquoi sur un kernel linux ?

Le kernel linux a un système de gestion mémoire et de processus reconnu pour sa stabilité et ses performances. Le modèle de sécurité utilisé par linux, basé sur un système de permission, est connu pour être robuste et performant. Il n'a pas changé depuis les années 70.

Le kernel linux fournit un système de driver permettant une abstraction avec le matériel. Il permet également le partage de librairies entre différents processus, le chargement et le déchargement de modules à chaud.

le kernel linux est entièrement open source et il y a une communauté de développeurs qui l'améliorèrent et rajoutent des drivers. C'est pour les points cités ci-dessus que l'équipe en charge du noyau a décidé d'utiliser un kernel linux[LUG].

2.6.2 Android Runtime

Android inclut un ensemble de librairies fournissant la plupart des fonctionnalités des librairies standard de Java. Chaque application Android s'exécute dans un processus, avec sa propre instance de la machine virtuelle Java, appelée Dalvik. Elle a été écrit pour optimiser l'exécution d'une multitude d'instances de la machine virtuelle, avec une empreinte mémoire réduite. Dalvik s'appuie sur le noyau Linux pour les fonctionnalités bas-niveau tels que les threads ou la gestion de la mémoire.

2.6.3 Librairies

Android fournit un ensemble de librairies C/C++ utilisées par différents composants du système. Ces fonctionnalités sont rendues disponibles aux développeurs au travers du framework d'application d'Android. On trouve parmi ces librairies : librairie C standard, moteurs d'affichage 2D et 3D comme le Open GL, SQLite pour les bases de données, rendu des polices de caractères etc...

2.6.4 Application Framework

Le Framework d'application est la couche qui nous intéresse tout particulièrement. C'est elle qui fait le lien, grâce à un ensemble d'APIs Java, entre le système et l'application. Étant un système ouvert, Android permet aux développeurs de concevoir des applications très riches et de tirer parti d'un maximum de fonctionnalités. Les développeurs ont donc accès aux mêmes fonctionnalités que celles utilisées par les applications fournies avec Android. Toute application Android repose sur un ensemble de services et systèmes parmi lesquels :

1. Un ensemble de «Views» permettant de construire l'interface graphique de l'application : listes, grilles, champs textes, images, et même intégration d'un navigateur web ou d'une vue Google Maps.
2. Des «Content Providers» qui permettent aux applications d'accéder à des données d'autres applications ou de partager ses propres données.
3. Un «Ressource Manager» pour accéder à des éléments autres que du code : données textuelles traduites, images, descriptions XML d'interfaces graphiques etc.
4. Un «Activity Manager» pour gérer le cycle de vie de l'application.

Ce rapide survol de l'architecture du système m'a permis de mieux comprendre comment fonctionne une application Android. Confinée dans la couche la plus haute, elle accède au système uniquement via les API's Java exposées par la couche Application Framework. Ainsi, si une fonctionnalité est présente dans le noyau Linux (couche rouge sur le schéma) ou dans les librairies système (couche verte), mais qu'elle n'est pas reliée au Framework d'application, elle ne sera pas utilisable directement dans une application Android.

2.7 Conclusion

Dans ce chapitre, on a fait une étude de l'art d'Android tout en présentant un bref historique, les fonctionnalités que nous pouvons trouver sur ce système d'exploitation et l'architecture d'Android, à savoir les principaux composants du système. Le but de tout ça est comment on peut développer une application Android, c'est ce qu'on va voir dans le dernier chapitre.

La programmation sous Android

3

3.1 Introduction

Afin de répondre aux besoins croissants des exigences des smartphones, les développeurs d'Android ont créé une plateforme spéciale très bien structuré. Ce dernier chapitre sera découpé en 2 parties :

La première partie consacré à décrire cette plateforme simplement et de montrer comment on peut développer aisément des applications pour Android.

La deuxième partie consacrée à décrire les étapes de développement d'une petite application qui permet de requêter une base de données externe[MTI].

3.2 Les applications Android

3.2.1 Définition

Une application Android se compose de différents fichiers, relies entre eux par un fichier de configuration. Les applications Android sont très variables et visent plusieurs domaines. On peut trouver des applications de commerce, utilitaire, service d'information, des jeux, etc... [DG]

3.3 Les composants d'une application Android

Les applications Android se composent donc de plusieurs éléments, on va voir les composants les plus importants :

3.3.1 Activity

Une activité représente la couche de présentation d'une application Android. Une description simplifiée est qu'une Activity représente un écran d'une application Android. Ceci est un peu incorrect puis les Activités peuvent être affichées comme des boites de dialogue ou en transparence.

Une application Android peut disposer de plusieurs Activités.

3.3.2 Service

Un service, à la différence d'une activité, ne possède pas de vue mais permet l'exécution d'un algorithme sur un temps indéfini. Il ne s'arrêtera que lorsque la tâche est finie ou que son exécution est arrêtée. Il peut être lancé à différents moments :

- Au démarrage du téléphone.
- Au moment d'un événement (arrivée d'un appel, SMS, mail, etc. . .).
- Lancement d'une application.
- Action particulière dans une application.

3.3.3 ContentProvider

Le ContentProvider fournit une interface aux données d'application. Via un ContentProvider notre application peut partager des données avec d'autres applications. Android contient un système de base de données SQLite fréquemment utilisé conjointement à un ContentProvider. SQLite enregistre les données, qui pourront être accédées par un ContentProvider.

3.3.4 Intents

Les Intents sont des messages asynchrones qui permettent à une application de requêter une fonctionnalité d'un autre composant du système Android, par exemple des Services ou des Activités.

Les Intents permettront de combiner des composants faiblement couplés pour réaliser certaines tâches.

3.3.5 BroadcastReceiver

Un BroadcastReceiver peut être enregistré pour recevoir des messages systèmes et des Intents. Un BroadcastReceiver sera notifié par le système Android.

3.3.6 Views et ViewGroups

Les Views sont des composants (widgets) de l'interface utilisateur, par exemple des boutons ou des zones de saisie.

La classe de base de toutes les Views est **android.view.View**. Les Views ont souvent des attributs qui peuvent être utiliser pour changer leur apparence ou leur comportement.

Un ViewGroup est en charge de positionner d'autres Views. Un ViewGroup est aussi appeler gestionnaire de composants.

3.3.7 Autre composants

Android fournit d'autres composants mais la liste ci-dessus décrit la plupart d'entres eux. Les autres composants sont les “Live Folders”, “Live Wallpapers”, “Notifications” etc.

3.3.8 Les permissions

Dans le système Android, pour qu'une application puisse utiliser une ressource du téléphone (Gps, bluetooth, Internet), le développeur défini lors de la programmation de celle-ci, la liste de permissions que le programme a besoin pour bien fonctionner.

Exemple : pour demander un accès à internet, on indiquera la ligne :

```
<uses-permission android :name="android.permission.INTERNET" />
```

3.4 Les activités sous Android

3.4.1 Définition

Une activité est le point d'entrée principal pour une application Android, elle peut être définie comme un écran de l'application. Donc pour chaque écran de l'application, il lui faut une ou plusieurs activités. Elle est composée de deux volets :

La logique de l'activité : est la gestion de cycle de vie de l'activité qui est implémentés en java dans une classe hérité de l'abstract class Activity.

L'interface utilisateur : qui pourra être définie soit dans le code de l'activité, soit de façon plus générale dans un fichier XML avec des balises très adapté.

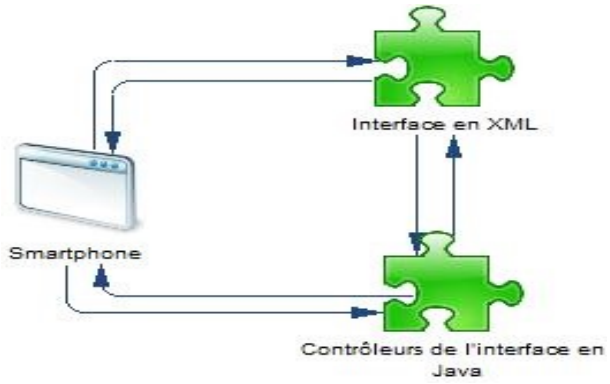


Figure 3.1 – *Interaction entre l'interface et le controleur*

3.4.2 Cycle de vie d'une activité

Les méthodes ci-dessous sont appelées au cours de cycle de vie. si elles sont surchargées,elle doivent faire appel à leurs homologue de la superclasse super.oncreate(param).

onCreate()

Cette méthode est appelée à la création d'une activité. Elle sert à l'initialiser ainsi que toutes les données nécessaires à cette dernière. Quand la méthode OnCreate est appelée, on lui passe un Bundle en argument. Ce Bundle contient l'état de sauvegarde enregistré lors de la dernière exécution de cet activité.

onStart()

Cette méthode est pour signifié le début d'éxecution d'une activité (début du passage au premier plan).

Si l'activité ne peut pas aller en avant plan quelque soit la raison, l'activité sera transférée à OnStop.

onResume()

Cette méthode est appelée après OnStart (au moment où l'application repasse en foreground). à la fin de l'appel à la méthode onResume l'application se trouve au premier plan et reçoit les interactions utilisateurs.

onPause()

Si une autre activité passe au premier plan, la méthode onPause est appelée sur l'activité. Afin qu'on sauvegarde l'état de l'activité et les différents traitements effectués par l'utilisateur.

onStop()

Appelée quand l'activité n'est plus du tout visible quel que soit la raison.

onDestroy()

Appelée quand l'application est totalement fermée (Processus terminé). Toutes les données non sauvegardées sont perdues.

Le schéma suivant définit le cycle de vie d'une activité Android :

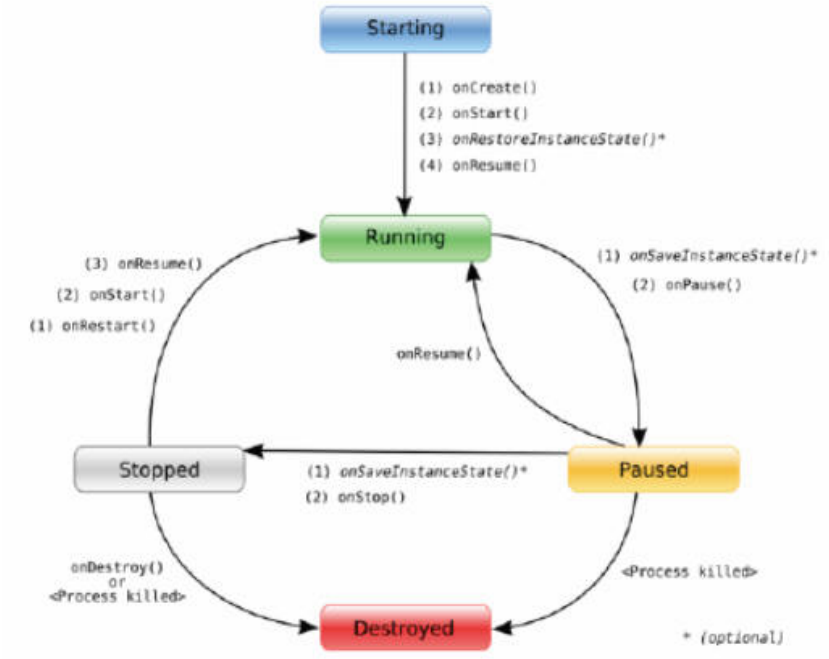


FIGURE 3.2 – *Cycle de vie d'une activité*

3.5 Installation de l'environnement de développement

L'installation de l'environnement de développement a quelques nouveautés, avec notamment l'arrivée de Windows 8 et la mise à disposition par Google de l'Android Developer Tools qui est une version d'Eclipse intégrée et déjà configurée dans le nouveau pack SDK Android / ADT :

Figure 3.3 – *Android Developer Tools*

3.5.1 Téléchargement de JDK

Le premier prérequis est le JDK (Java Développent Kit) qui est disponible gratuitement chez Oracle. La différence entre le JDK et la JRE plus commune, est que le JDK permet de compiler du code Java, alors que la JRE (Java Runtime Environnent) ne permet que d'exécuter du Java. Depuis le site d'oracle : http ://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html

3.5.2 Téléchargement de SDk

Un SDK, c'est-à-dire un kit de développement, est un ensemble d'outils que met à disposition un éditeur afin de nous permettre de développer des applications pour un environnement précis. Depuis le site des développeurs Android : http ://developer.android.com/index.html

3.5.3 Installation du JDK et de ADT

Après la décompression du fichier adt-bundle-windows-x86_64.zip téléchargé, on va obtenir deux sous dossiers éclipse et SDK L'installation du JDK est simple en suivant la procédure d'installation par défaut.

3.5.4 lancement et configuration de ADT

Après le lancement de l'Android Developer Tools nous demande, comme le fait Éclipse, l'emplacement de stockage de nos projets.

Il existe plusieurs versions de la plateforme Android qui ont été développées depuis ses débuts et il existe donc plusieurs versions différentes en circulation.

Le premier nombre correspond à la version d'Android et le second à la version de l'API Android associée. Quand on développe une application, il faut prendre en compte ces numéros, puisqu'une application développée pour une version précise d'Android ne fonctionnera pas pour les versions précédentes.

On doit télécharger et installer les SDK autres que celui inclus pour le package fourni par Google (Android 4.2), comme par exemple celui de Android comme la version 1.5 et 2.3 Gingerbread qui reste la version d'Android la plus utilisée.

Pour cela on utilise le SDK Manager qui est dans le dossier Tools du SDK : il est sous la forme du fichier "android.bat" sur lequel il faudra double-cliquer pour l'exécuter.

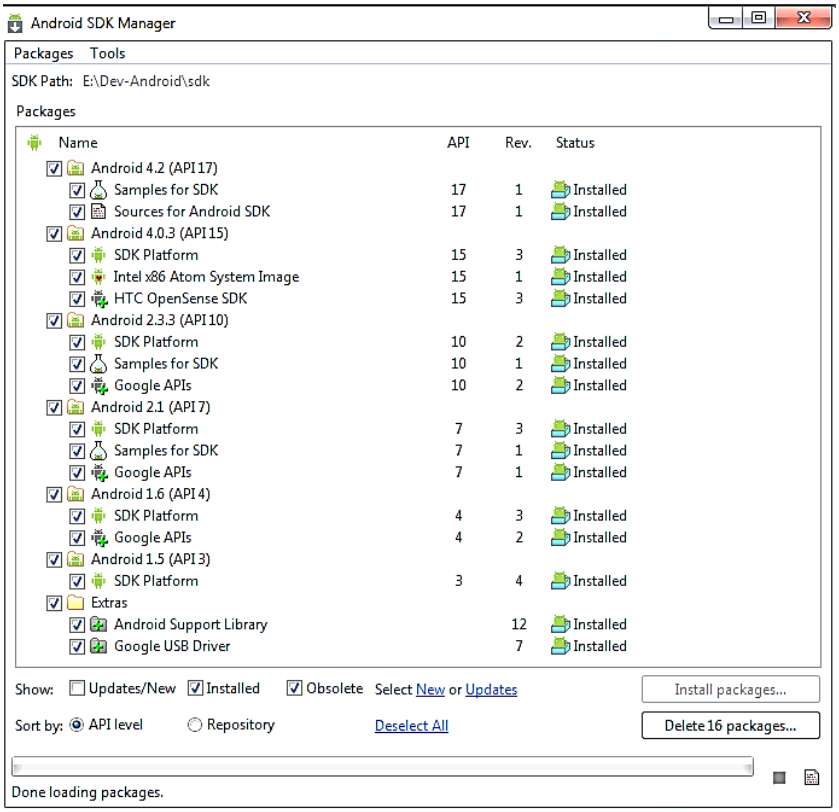


FIGURE 3.4 – *Android SDK Manager*

3.5.5 Création D'un projet Android

Pour créer un nouveau projet Android, on n'a qu'à suivre ces étapes : Cliquer sur file puis New Android Application Project après ça une fenêtre s'affiche :

Le premier écran nous permet de spécifier

- Le nom de l'application.
- Le nom du projet.

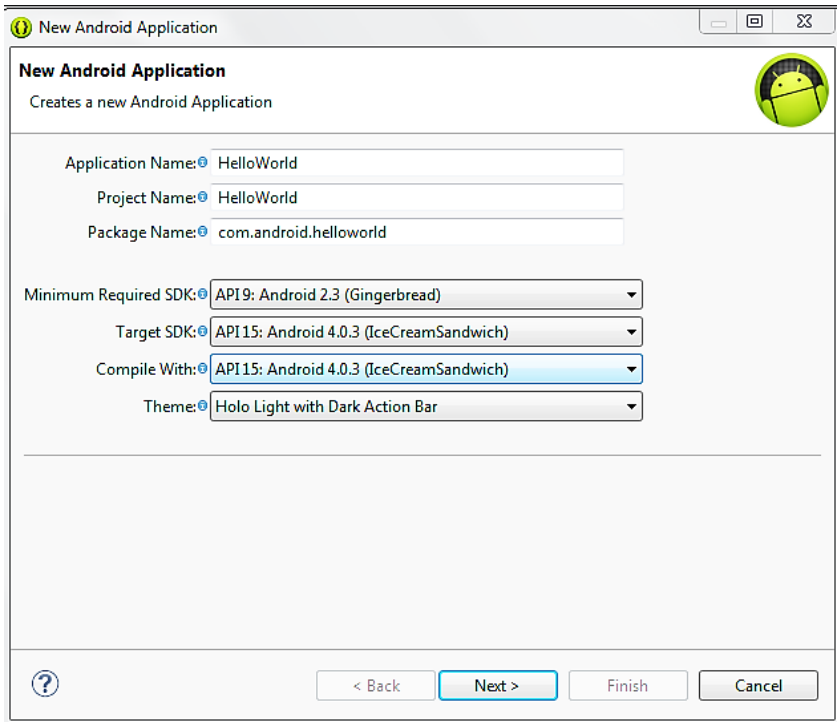


FIGURE 3.5 – *Création d'un nouveau projet Android*

• L'identifiant de l'application (nom du package) : comme en java, un projet Android se compose par un ou plusieurs packages afin de faciliter l'accès aux cœurs d'une application. Le nom de chaque package se compose de trois partie séparé par des points.

• La version d'Android utilisée pour développer et compiler notre application (Target SDK et Compile with).

• La version minimum d'Android compatible avec l'application (Minimum Required SDK).

• Le thème global pour l'application.

3.5.6 Structure d'un projet Android

On va voir maintenant les composants les plus importants d'un projet Android, on peut aussi créer d'autres composants selon nos besoins :

MainActivity.java : correspond à l'activité principale créée avec le projet (car liée au fichieractivity_main.xml).

R.java : Fichier contenant les "paramètres" de l'application. Il se génère automatiquement,

Bin : Le dossier "Bin" contient les binaires de l'application.

Res : Ce dossier contient les images utilisées dans le projet, les XML du projet (les fichiers XML servent à définir l'interface de l'application) et le dossier "values".

Drawable-(xh/h/m/l)dpi : Ces dossiers contiennent les images de l'application. Tous ces dossiers devraient, dans l'idéal, contenir les mêmes images, mais avec des résolutions différentes. En effet, selon la résolution de l'écran utilisé, les images seront tirées du dossier correspondant.

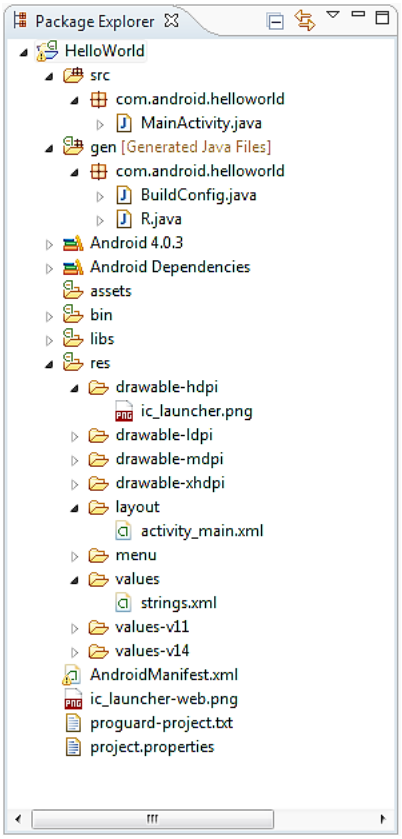


FIGURE 3.6 – *L'arborescence d'un projet Android*

Layout : Ce dossier contient les XML de l'application pour les interfaces. Qui correspond à l'interface de l'activité principale de l'application.

Values : ce dossier contient aussi des XML, mais qui servent à stocker des chaînes de caractères.

AndroidManifest.xml : fichier XML dans lequel on déclare les différentes activités de l'application, et certains paramètres de l'application (par exemple si on autorise la rotation de l'application, le sens de l'application (vertical, horizontal), l'icône de l'application, etc...).

3.5.7 Création d'un émulateur

Pour créer un AVD [1] (Emulateur), cliquer sur le bouton "créer un AVD" , une fenêtre s'affiche : Chiquer sur new, une autre fenêtre s'affiche :

1. Android virtual device

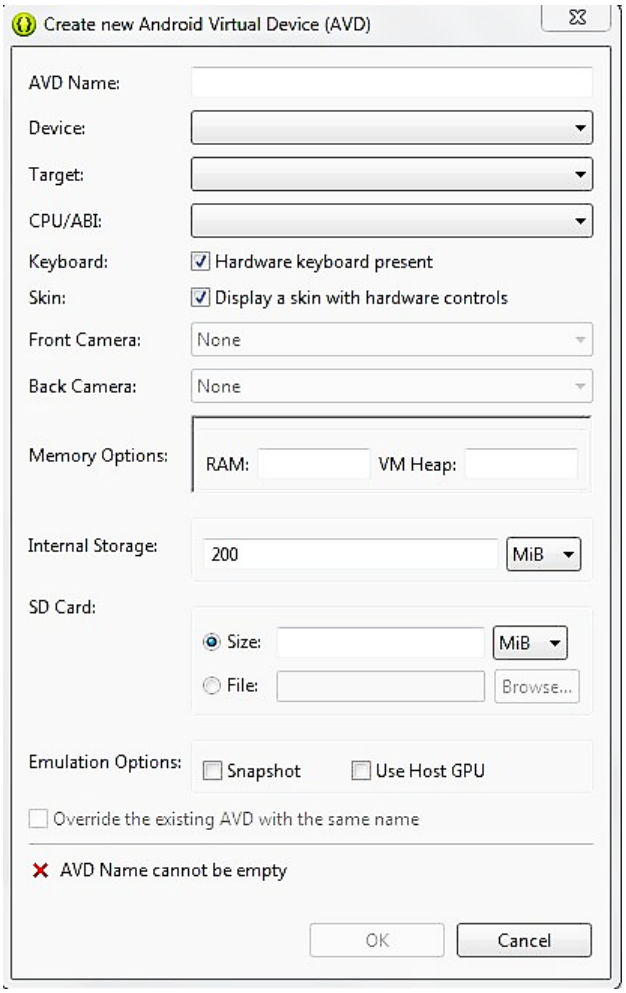


FIGURE 3.7 – *Création d'un nouveau AVD*

- AVD Name : le nom de l'émulateur.
- Devise : le modèle de l'émulateur.
- Target : la version du SDK.
- Memory Options : l'espace de mémoire vive réservé pour l'émulateur.
- SD Card : pour créer un dossier qui joue le rôle d'un externe mémoire...
- La case "Use Host GPU" pour que l'émulateur utilise le GPU (le processeur graphique) de notre PC.

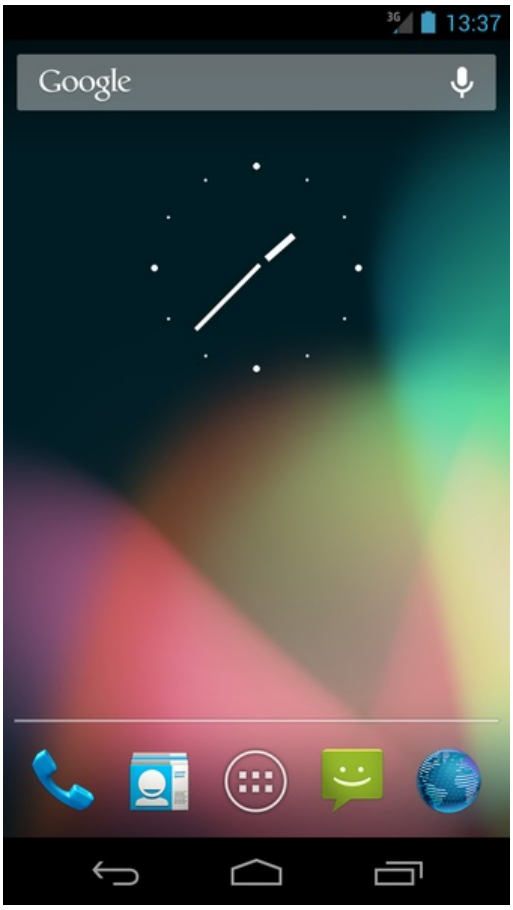


FIGURE 3.8 – *L'AVD version 4.2 de Jelly Bean*

3.6 PRÉSENTATION DE L'APPLICATION

3.6.1 Base de données externe

Concernant le stockage des Données, il existe plusieurs solutions :

1. Mettre en place une BDD externe (MySQL, PostgreSQL, Oracle etc.)
2. Mettre en place une BDD interne : en ce qui concerne Android, il s'agit d'une BDD SQLite ; Stocker les données dans un fichier.

Le choix du type de base dépend évidemment des utilisations que l'on va faire des données. L'avantage d'une BDD externe est l'espace de stockage des données qui peut être immense, contrairement à une BDD stockée sur le smartphone - espace de stockage limité. Cependant le temps de réponse de la BDD interne peut être largement inférieur à celui d'une BDD externe.

3.6.2 Architecture avec une Base de Données Externe

D'un côté, nous avons notre application Android, d'un autre, notre base de données :

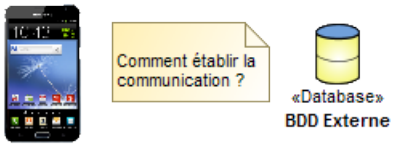


FIGURE 3.9 – *Communication entre la BDD et le client Android*

L'idée, pour faire communiquer ces deux entités, est d'utiliser un Middleware. Ce Middleware, par définition, va organiser, adapter et traiter les échanges entre l'application et la BDD.

FIGURE 3.10 – *Architecture 3 niveaux*

En fait, notre application va ennoyer une demande au Middleware (ce middleware tourne sur un SGBD), Le Middleware va recevoir la demande, et la traduire en requête compréhensible par la BDD en langage SQL. Il va ensuite envoyer cette requête à la BDD, qui va renvoyer la liste des champs. Le middleware va récupérer et adapter ces données au format souhaité par l'application Android, puis renvoyer le résultat à l'application, qui va l'afficher.

Revenons au Middleware. C'est une entité qui tourne sur un serveur SGBD et qui va permettre de récupérer des informations de la BDD, et de répondre aux demandes de diverses sources (par exemple notre application Android). Celui-ci est développé avec la technologie JavaEE et JDBC [2].

Voici le schéma de fonctionnement de l'application :

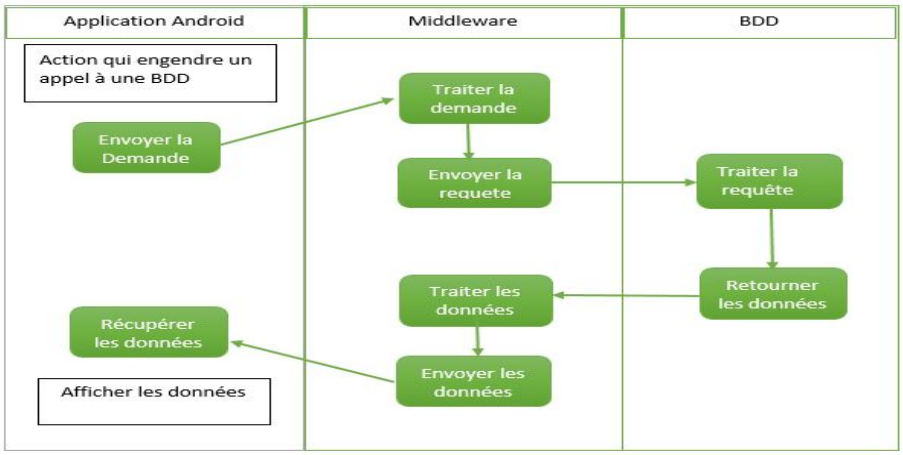


FIGURE 3.11 – *Schema de fonctionnement de l'application*

2. Java database connectivity

3.7 L'art de développement

Dans cette partie, on va voir les étapes de l'implémentation du projet :

3.7.1 L'application Android

l'utilisateur Android ne fait pas grand-chose, simplement, pour la première activité il saisi l'adresse IP de serveur et cliquer sur un bouton pour passer à la deuxième activité, dans cette dernière, il remplit les champs : le nom de la base de données interrogé, le nom de la table, et le numéro des champs, et cliquer sur un bouton pour voir les résultats.

On doit d'abord adapter l'interface de l'application pour répondre aux besoins de l'utilisateur, on doit créer deux interface l'un correspond à l'activité principal **activity_main.xml** , et l'autre lorsque l'utilisateur clique sur un bouton pour passer au deuxième activité **client.xml**. La création faite dans le répertoire **res/layout**.

FIGURE 3.12 – *L'interface de l'application DbDroid*

on doit aussi ajouter le droit d'utilisation de l'Internet au fichier manifest.xml pour que notre application puisse communiquer avec l'extérieur (utilisation des sockets de protocole TCP/IP).

```
<uses-permission android:name="android.permission.INTERNET"/>
```

FIGURE 3.13 – *Le droit d'accés à l'internet*

Maintenant,on peut passer au fichier **MainActivity.java**,qui se trouve dans le package **com.os.dbdroid** l'interface XML correspond à cet activité est **activity_main.xml**. cette classe contient une seul méthode OnCreate en lui passant un bundle comme paramètre pour sauvegarder son état lors de la dernière exécution, et ajouter l'interface correspond à cet activité :

```
public class MainActivity extends Activity{


protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

L'interface crée précedement contient plusieurs Vus, ce qui nous intéresse c'est le champ de saisie et le bouton, on doit déclarer les objets de cet interface pour les contrôler grâce à la méthode findViewById, Android gère un fichier **R.java** qui contient l'arborescence du projet, ce qui nous permet de rechercher n'importe quel composant à partir de son Id.

```
final EditText  IpServ = (EditText) findViewById(R.id.Ipserv);
        Button loginButton = (Button) findViewById(R.id.conneserv);
```

On doit ajouter un listener lorsque l'utilisateur clique sur le bouton, l'interface OnClickListener contient la méthode onClick(View v) qui permet de récupérer l'état du bouton et effectuer plusieurs taches.

On doit instancier un objet TEXTE de la classe String qui permet de récupérer l'adresse IP saisie par l'utilisateur, ce mécanisme fait grâce à la méthode getText().toString().

On doit aussi mettre quelque restrictions si l'utilisateur n'a pas saisie l'adresse IP grâce à la méthode equals(), cette méthode permet de comparer la chaine saisie par l'utilisateur avec une autre chaine, si elle est vide on doit indiquer à l'utilisateur que le format d'adresse IP est incorrecte. et on ne le passe pas à la deuxième activité.

Pour cela, Android fournit une classe spéciale s'appelle Toast, les toast sont des messages qui apparaissent au-dessus de toute application lancée grâce à la méthode makeText(Context, "notification",durée) , pour le lancer on doit utiliser la méthode show().

Si le format de l'adresse IP est valide, le passage à la deuxième activité est simple en utilisant un objet de type intent, ce dernier comporte deux paramètres, le premier correspond à l'activité actuelle et le deuxième à l'activité visé.

Pour passer l'adresse IP comme paramètre à la deuxième activité, afin d'instantier l'objet Socket et faire le connexion, on doit affecter la méthode putExtra à l'objet intent, en lui passant une chaine de caractère indique le nom du message et le contenu du message comme paramètres.

L'activite se lance grâce à la méthode startActivity(intent).

```
@Override
                public void onClick(View v) {

                        // + Recuperer Ip saisie par l'utilisateur
                        String texte = IpServ.getText().toString();
                        // + Gestion des erreurs
                        if (texte.equals("")) {
                                Toast.makeText(MainActivity.this,
                                                "Erreur format d'adresse
IP",Toast.LENGTH_SHORT).show();
                                return;
                        }
                        // + Passage de l'activité Main vers l'acivité Client
                        Intent intent = new
Intent(MainActivity.this,client.class);
                        // + envoyer le msg au deuxieme activité "label comme
un paramtere indique + ip
                        intent.putExtra("ip",texte);
                        // + Start l'intent
                        startActivity(intent);
                }
```

Maintenant, on peut passer au fichier **client.java** qui contient le cœur de l'application et qui se trouve dans le package **com.os.dbdroid**, cette activité hérite de la classe abstrait **Activity** qui contient plusieurs méthodes (onCreate, OnPause etc. . . .), et aussi, elle doit implémenter l'interface **onclicklistener** qui contient la méthode OnClick.

```
public class MainActivity extends Activity implements OnClickListener
```

On doit aussi définir les objets de l'interface de l'application pour les contrôler.

```
        // + Les Views
    private Button mButton;
    private EditText mDataBase;
    private EditText mTable;
    private EditText mChamp;
```

La méthode OnCreate permet de créer une activité et sauvegarder son état lors de la dernière exécution en lui passant un bundle.

```
protected void onCreate(Bundle savedInstanceState)
```

On doit aussi instancier des objets qui nous permettre de contrôler les différents vus de l'interface client.xml grâce à la méthode findViewById.

```
// + Rechercher les Views pour les controler
mButton = (Button) findViewById(R.id.connexion);
mDataBase = (EditText) findViewById(R.id.EdBase);
mTable = (EditText) findViewById(R.id.EdTable);
mChamp = (EditText) findViewById(R.id.EdChamp);
```

Pour récupérer l'État du bouton, et s'il est activé, on doit implémenter l'interface onClickListener à la classe activité, ce qui implémente une méthode onClick(). puis on va placer l'écouteur sur le contexte de l'activité (this).

```
// + Un Listener
mButton.setOnClickListener(this);
```

L'allocation des objets de type chaines de caractères qui vont récupérer les saisies de l'utilisateur grâce à la méthode getText().toString().

```
public void onClick(View v) {
            // TODO Auto-generated method stub
            try {

                  // + Recuperer les saisies de l'utilisateur
                  String DbName = mDataBase.getText().toString();
                  String TabName = mTable.getText().toString();
                  String champ = mChamp.getText().toString();
```

L'utilisation des sockets permet de communiquer le client avec le serveur avec des protocoles assez proche aux réseaux en mode fiable TCP/IP (communication entre les deux entités envoie – accusé). Le socket client hérite de la classe java.net.Socket, et elle prend comme paramètres l'adresse IP et le port de serveur.

On doit instancier un nouveau objet de la classe socket qui prend deux paramètres sont l'adresse IP et le port 7777.

On doit récupérer l'adresse IP avec les intents ,l'objet EXTRA de la classe bundle permet de récupérer cette adresse grâce à la méthode getIntent().getExtras().

```
// + Recuperer l'adresse ip saisie par l'utilisateur avec intent
                  Bundle extra = getIntent().getExtras();
                  String IpServer = extra.getString("ip").toString();
                  // + Instancier le Socket service
                  Socket Service = new  Socket(IpServer,7777);
```

L'instanciation des objets de la classe ObjectOutputStream représente un flux objet qui permet de sérialiser (envoyer) les objets au coté serveur dans un objet service de la classe socket grâce à la méthode writeObject(), et il va désérialiser les résultats reçus à partir de côté serveur avec un objet de la classe ObjectInputStream.

```
// + Les flux D'entrée Sortie des Objets in, out
      ObjectInputStream in = new ObjectInputStream(Service.getInputStream());
      ObjectOutputStream out = new
ObjectOutputStream(Service.getOutputStream());

                  // + Sauvegarder Les objets sérialisé
                  out.writeObject(DbName);
                  out.writeObject(TabName);
                  out.writeObject(champ);
```

La méthode la plus simple pour indiquer à l'utilisateur que les objets saisies sont envoyé avec succès est d'utiliser une petite notification(Toast).

```
// + Un Toast dit que la requète est envoyé
                    Toast.makeText(this, "requette envoyée avec
succée",Toast.LENGTH_SHORT).show();
```

la méthode readObject() permet de lire les objets reçus en utilisant un appel à la méthode qui retourne l'objet avec un type Object. Un cast au type tableau est nécessaire pour obtenir ce type.

les deux méthode suivantes get(i).toString() permet d'extraire et de caster les lignes qui portant sur un objet réponse de la classe arraylist :

```
// + Table des Champs Recu
                    ArrayList reponce = (ArrayList) in.readObject();

                    String donne = "";
                    int i = 0;

                    // + les resultats sous forme des chaines de caracteres
                    while (i<reponce.size()) {

                          donne  = donne + reponce.get(i).toString();
                    }
```

Finalement pour afficher les champs résultants il suffit d'utiliser un toast qui permet d'afficher le résultat de la requête au client :

```
      // + Les resultats sous forme D'un Toast
                    Toast.makeText(this, "Resultat de la requette
:"+donne,Toast.LENGTH_LONG).show();
```

3.7.2 Le middleware

Dans la deuxième partie du code, on doit définir une classe Serveur implementé en java, et qui effectue plusieurs opérations :

- réception des champs saisies par l'utilisateur.
- traduire les données reçu sous forme d'une requête d'interrogation non paramétré (Sélect).
- charger le driver de la base de données.
- établir la connexion entre le JDBC et le SGBD.
- envoyer la requête au SGBD.
- recevoir les champs resultants.
- envoyer les champs résultants au client sous un forme adapté(table des champs).
- gérer les demandes de plusieurs clients(utilisation des threads).

Premièrement, la méthode main permet de combiner l'exécution de plusieurs classes et méthodes. Le but de l'instanciation d'un objet inconnu de la classe serveur qui est définit au dessous est d'effectuer plusieurs taches , la méthode main peut lever des exceptions de la classe IOException.

```
public static void main (String[] args) throws IOException {

            new Serveur().go();
      }
```

la méthode go() regroupe des opérations de l'établissent d'une connexion entre le client et le serveur, elle peut lever des exceptions de type IOException.

L'objet service est une instanciation de la classe socketServer qui a un comportement d'un serveur via une interface par socket,cette classe utilise seulement le numéro de port pour l'écoute, contrairement à la classe SocketClient . La méthode essentielle est l'acceptation d'une connexion d'un client grâce à la méthode accept().

L'utilisation des threads est très importante en cas où il y a plusieurs clients peuvent requêter la base de données au même temps, les threads ou fil d'exécution sont des processus léger permet d'exécuter des taches en parallèle.

L'instanciation d'un objet de la classe Traitement hérite de Thread est très important, la méthode start() permet de lancer le fil d'exécution sur l'objet instanciée pour exécuter des taches en une seul fois.

```
void go() throws IOException{
            ServerSocket service = new ServerSocket(7777);
            System.out.println("Service démarré ...");
            boolean ServerOn = true;

            while (ServerOn) {

                  // Accepter la demande de la connexion
                  Socket client = service.accept();
                  System.out.println("Un client veut executer une requete
...");

                        new Traitement(client).start();


            }
```

La différence entre la méthode start() et run() est que la méthode run() peut exécuter des taches de l'utilisateur plusieurs fois contrairement au start() et cela est un avantage, dans cette méthodes on doit utiliser des flux d'objet pour faire des échanges d'objets.

D'abord, l'instanciation d'un objet de type flux objet qui récupère l'adresse IP de client qui veut connecter au middleware, portant sur un objet de type Socket grâce au deux méthodes getInetAddress().getHostAddress().

```
      @Override
            public void run(){

                  // les flux d'entrée sortie
                  ObjectInputStream in ;
                  ObjectOutputStream out ;

                  try {

                        System.out.println("IP de Client
:"+client.getInetAddress().getHostAddress());
```

Pour faire des échanges entre ces deux entités, il suffit d'instancier des objets de type OBjectInputStream et OBjectOutputStream, qui vont récupèrent et transmettent les données portant sur un objet client de la classe Socket en effectuant respectivement les deux méthodes getInputStream() et getOutputStream(), Ces méthodes peuvent lever des exceptions de type ClassNotFoundException.

```
                  in = new ObjectInputStream(client.getInputStream());
                  out = new ObjectOutputStream(client.getOutputStream());
```

La transformation des champs en une forme d'une requête SQL, puis la transmission de résultat au client se fait grâce à la méthode **writeObject()** en lui passant le résultat de la requête comme paramètre.

```
            // + Ecrire les données sous forme d'une requete MySQL
                  try {

                        String Requete = "SELECT *"+  "FROM " + Table;
                        System.out.println("La requete : "+Requete);
                        System.out.println("");

                        ResultQuery(Requete);

                              // + Envoyer le resultat de la requete au
client
                        out.writeObject(this.ResultQuery(Requete));
```

Pour se connecter à une base de données en utilisant un driver, la documentation du driver fournit le nom de la classe à utiliser. Dans ce cas, on doit charger le pilote "com.mysql.jdbc.Driver" pour une base de données MySQL, le chargement du driver se fera avec le code suivant :

```
private void getconnection() throws java.sql.SQLException {

                  // + Charger le driver JDBC
                  String pilote = "com.mysql.jdbc.Driver";
                  try {

                        Class.forName(pilote);

                        System.out.println("driver trouvé...!");

                  } catch (ClassNotFoundException e) {
                        // TODO: handle exception
                        System.out.println("driver non
trouvé...!"+e.getMessage());
                  }
                  // + l'@dresse de la base de données dans le Host
                  String URL_jdbc =
"jdbc:mysql://localhost:3306/"+this.Dbase+"?";

                  // + Etablir une connection avec la base de données
                  connect = (Connection)
DriverManager.getConnection(URL_jdbc,"root","");
                  System.out.println("Connexion réussie ! ..");

            }
```

Cette méthode de la classe **Class** peut lever l'exception **ClassNotFoundException**.

Pour se connecter à la base de données, on doit instancier un objet de la classe prévue par l'éditeur du pilote et dont on ignore tout, mais qui réalise l'interface **Connection** en lui précisant sous forme d'URL la base à accéder.

La syntaxe URL peut varier d'un type de base de données à l'autre mais elle est toujours de la forme : **protocole :sous_protocole :nom**.

La méthode **getConnection()** peut lever une exception de type SQLException.

La méthode **ResultQuery** permet de récupérer les champs résultants sous forme d'un tableau, elle prend comme paramètre la requête crée, et retourne un tableau de chaine de caractères. Cette méthode peut lever des exceptions de type SQLException.

```
      public ArrayList ResultQuery (String requete) throws
java.sql.SQLException
```

L'objet **stm** de la classe **Statement** permet d'envoyer des requêtes SQL à la base. Sa création s'effectue à partir d'une instance de la classe **Connection**. Pour nos requêtes SQL de type interrogation n'est pas paramétré (SELECT), la méthode à utiliser de l'interface **Statement** est **executeQuery()**.il est nécessaire de lui fournir en paramètre la requête SQL sous forme d'une instance de la classe String. Le résultat d'une requête d'interrogation est renvoyé dans un objet d'une classe inconnue qui réalise l'interface **ResultSet** par la méthode **executeQuery()**.

```
      Statement stm = (Statement) connect.createStatement();

                        ResultSet res= (ResultSet) stm.executeQuery(requete);
                        DatabaseMetaData md = connect.getMetaData();
```

Une fois que la connexion établit, on peut instancier un objet de la classe DatabaseMetaData qui permet d'obtenir des informations sur la base de données.

```
System.out.println("");
                        // +Récupère le nom de produit de base de données
                        System.out.println(" Le nom de la base de données
:"+md.getDatabaseProductName());

                        // +Récupère la version de produit de base de données
                        System.out.println(" La version de la base de données
:"+md.getDatabaseProductVersion());

                        // +Récupère le nom de pilote de base de données
                        System.out.println(" Le nom du pilote
:"+md.getDriverName());

                        // +Récupère la version de pilote de base de données
                        System.out.println(" La version du pilote
:"+md.getDriverVersion());
```

Des appels successifs à **next()** permettent de parcourir l'ensemble des lignes de résultat. et aussi de remplir le tableau avec les champs résultants grâce la méthode **add()** qui récupère l'objet **res** de l'interface **ResultSet** et le traduire au type chaine de caractère grâce à la méthode **getString(Integer.parseInt(Champ))**. Elle retourne false lorsqu'il n'y a plus d'enregistrement. Il faut toujours protéger le parcours d'une table dans un bloc de capture d'exception car cette méthode peut lever des exceptions de la classe **SQLException**.

La fermeture de l'interface de la connexion et du socket d'écoute est obligatoire, pour finir la communication entre le client et le serveur, elles s'exécutent par l'appel à la méthode close.

```
while(res.next()){
                        // + recuperer les champs resultants

System.out.println(res.getString(Integer.parseInt(Champ)));

liste.add(res.getString(Integer.parseInt(Champ)));
                }

                // + Fermeture la lecture de resultat
                res.close();
                // Fermeture de la connexion
                connect.close();
```

3.8 Conclusion

On a décrit dans ce chapitre la démarche suivie pour développer une petite application Android qui permet de requêter une base de données externe.

Conclusion général

On a étudié dans ce mémoire un des système embarquées les plus populaire au monde Android, on a tous d'abord commencés à donner quelques notions théorique nécessaire sur les bases de données, ensuite on a étudié de plus près le système Android, et son architecture logiciel, on a passer à l'installation de l'Android développer tools, un environnement de développement Android intégrés, et finalement, on a montré comment développer une application Android qui permet de requêter une base de données externe

Ce projet de fin d'étude m'a permet de mieux comprendre la philosophie de system Android qui est basé sur le langage Java, j'ai aussi approfondi mes compétences dans la programmation orienté objet en java.

Pour le futur, j'envisagerais de continuer dans ce domaine très prometteur, j'espère pouvoir être de très fameux un bon programmeur de système Android.

Bibliographie

[GAR] Georges GARDARIN,Base de donnees objet et relationnelle ,Eyrolles, 1990
[LUC] Jean-Luc Hainaut,Base de donnees et modèles de calcul ,2000
[DG] damien GUINARD julian chable,emmanuel roles,programmation android de la conception au deploiement,eyrolles, 2010
[MUR] mark MURPHY,l'art du developpement android ,peason,2010
[MTI] MTI.EPITA.FR Introduction à la programmation sous Android http ://www.mti.epita.fr/blogs/2010/08/03/introduction-a-la-programmation-sous-android/
[LUG] henri LAUGIE,java et eclipse developpez une application java ,2009