IFC2

AM-PM

Another Manager for Pills Reminder

Auteurs :
dos Santos Pedro
Fuino Francesco
Lala Alain

Responsables :
Jaton Markus
Vincent Mark

Date :
11/01/2010

Table des matières

1. Introduction

De nos jours, l'évolution du domaine médical nous a permis de guérir de nombreuses maladies. Cependant tout bénéfice n'étant pas sans conséquences, nous voyons une complexification des traitements proposés aux patients. Cette complexité risque à long terme d'engendrer d'autres pathologies comme des crises d'angoisse pouvant aggraver l'état actuel de la personne.

Fort de ce constat, il est possible d'imaginer une application indiquant, de manière fiable et simple d'utilisation, les médicaments à prendre tout au long de la journée. De plus les avancées technologiques en matière de téléphonie ainsi que leur accessibilité au grand public, en font un support idéal pour ce type d'applications.

Dans l'absolu cette application serait utilisée par des personnes responsables ayant toutes leurs capacités mentales. En effet, une personne incapable de se souvenir qu'elle doit prendre des médicaments ou n'étant pas consciente de son état, doit par mesure de sécurité rester sous surveillance du personnel médical.

2. Cahier des charges

2.1.Définition du projet

Il s'agit de concevoir une application qui permette à un médecin de doser des médicaments pour le patient, et de lui rappeler chaque jour de prendre la dose prescrite au moment utile. Le service médical peut, sur un service web, entrer les dosages de médicaments, et l'utilisateur recevra périodiquement en temps utile des indications sur son traitement médicamenteux. Il devra quittancer la prise des médicaments de manière systématique. On n'inclura pas la gestion du stock de médicaments chez le patient. Cette application est surtout prévue pour les médicamentations lourdes et complexes, impliquant plusieurs médicaments à prendre en des suites parfois complexes. (HIV, chimiothérapies lourdes,...).

2.2.Actions

Actions du service médical :

- Mettre à jour la liste de médicaments du patient
- Vérification si nécessaire du suivi du traitement

Actions du patient :

- Voir la liste des médicaments à prendre
- Quittancer la prise des médicaments

Disponibilité :

- Service web
- Téléphone

Non pris en charge :

- Pas de gestion de stock pour le client

2.2.1. Côté service médical

Le service médical doit être capable de mettre à jour la liste des médicaments du patient à prendre à travers une interface web.

- Identification du patient
  - Le patient doit être identifiable dans la base de données. Voir les différentes manières.

- Mise à jour de la liste
  - Mise à jour de la liste de médicaments associés au patient

2.2.2. Côté patient

L'utilisateur reçoit de manière périodique et au moment utile la liste ainsi que la quantité des médicaments qu'il doit prendre.

L'application doit donc être capable de :

- Etablir la connexion avec le serveur
- S'identifier au niveau de la base de données
- Télécharger la liste des médicaments à prendre pour la journée courante
- Mettre à jour la liste des alertes dans l'agenda du téléphone
- Mise à jour de la liste toutes les 12h

Au moment de l'alarme, la procédure suivante doit être mise en place :

- Arrêter l'alarme
- Affichage de la liste des médicaments à prendre dans l'ordre avec la quantité
- L'utilisateur doit valider un à un les médicaments une fois pris
- Si l'application est fermée alors qu'il y a encore des médicaments à prendre : attente de 5min avant de remettre une autre alarme
- Pour chaque médicament valider, envoyer un message au serveur

2.3.Sécurité

Dans la mesure où nous traitons des informations couvertes en grande partie par le secret médical, nous devrons prendre en compte les points suivants par assurer la sécurité des informations échangées :

- Si l'utilisateur se fait voler son portable, il faut mettre en place un système permettant de bloquer les connexions de ce téléphone en particulier et changer dans la base de données les informations du téléphone ayant accès au compte du patient. Une technique serait d'associer l'identification au niveau de la base de données à un numéro interne au téléphone (IMEI).

- Il faut garantir que les applications clientes soient correctement identifiées et authentifiées avant de lui donner accès à la base de données. Une solution serait d'introduire un intermédiaire entre la base de données et l'application cliente qui masquerait ainsi la base de données.

- Tout au long du transfert, il faudrait également garantir que même si les informations étaient interceptées, elles ne pourraient pas être lues. Il faudrait donc mettre en place un mécanisme de chiffrement et de déchiffrement des messages échangés.

3. Etat de l'art

Avant de nous lancer dans notre projet, nous avons étudié les différentes solutions proposées à ce jour dans le commerce.

Il existe des systèmes de boîtes à compartiment contenant les médicaments. Ainsi, le patient sait à quelle heure il doit prendre quels médicaments. Cependant ce système reste mécanique et aucun rappel n'est fourni au patient.

Figure 1 - Pilulier hebdomadaire MediMemo

Dans les solutions électroniques trouvées, le pilulier prévenait le patient soit en vibrant dans le cas portatif soit en émettant un signal sonore.

Figure 2 - Carrousel

Figure 3 - Médivib

Ces solutions ne sont pas adaptées à tous les types de traitement, soit la taille des compartiments est trop petite ou alors le système de rappels peu flexible. De plus il n'est pas possible au médecin de contrôler en direct la prise des médicaments.

4. Choix de l'infrastructure

Afin de rendre notre approche des différents problèmes plus facile, nous avons choisi de séparer ce projet en trois parties distinctes.

A l'une des extrémités de notre système, nous avons la base de données, qui peut être éditée par le médecin grâce à l'application web. A l'autre extrémité, nous avons l'application cliente, qui se trouve sur le mobile du patient, et qui doit pouvoir accéder à la base de données pour récupérer les informations dont elle a besoin pour fonctionner correctement.

La solution la plus évidente qui nous est venu à l'esprit dans un premier temps a été de permettre à l'application de directement pouvoir se connecter à la base de données. Cependant, rendre notre base de données accessible de cette manière n'est tout simplement pas possible et ce principalement pour des raisons de sécurité. En effet, une fois que le mobile est entre les mains du patient, nous avons un contrôle plus restreint sur les requêtes qu'il peut effectuer à la base de données, et lui donner la possibilité d'accéder directement aux informations qui s'y trouvent pose de nombreux problèmes ne serait-ce qu'au niveau de la confidentialité vis-à-vis des autres patients.

Il nous a donc fallu créer une entité de contrôle des accès à la base de données afin de pouvoir garantir le secret médical, c'est-à-dire que les informations d'un patient ne soient pas divulguées à une tierce personne.

Le serveur que nous avons implémenté répond alors à ce besoin. Il est à noter, qu'au niveau de cette infrastructure de communication entre la base de données et l'application cliente du patient, nous avons un modèle que nous pouvons assimiler au modèle MVC. Ainsi, notre base de données joue le rôle de modèle, l'application cliente celui de vue, et au milieu nous avons le serveur qui prend le rôle du contrôleur en fournissant à la vue les informations dont elle a besoin du modèle.

Si nous regardons le système dans l'ensemble, nous avons donc une représentation de notre infrastructure qui ressemble au schéma suivant

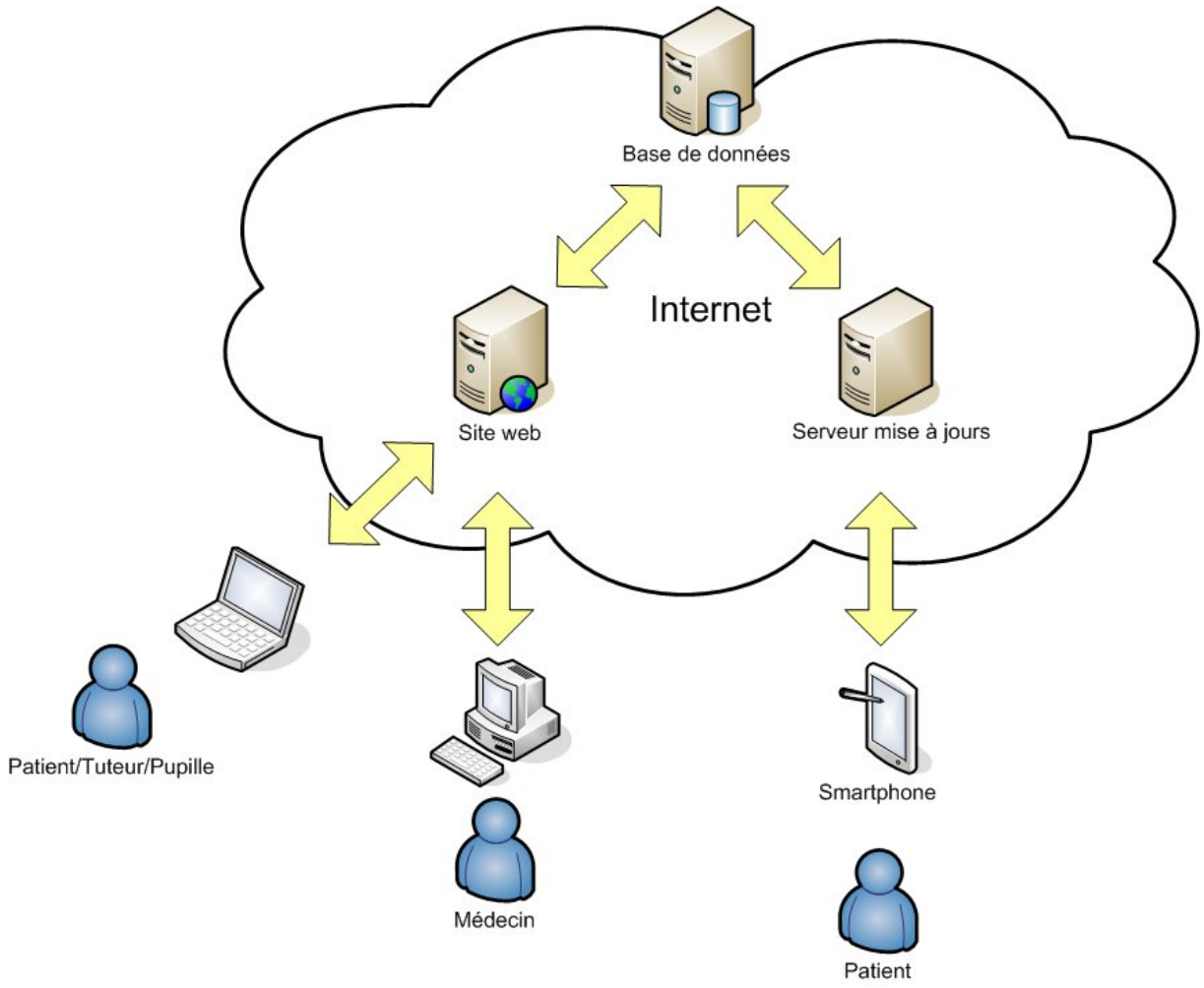

Figure 4 - Choix de l'infrastructure

5. Définition d'un médicament

Avant de définir des composantes plus précises de notre solution nous devons dans un premier temps nous mettre d'accord sur la définition d'un médicament, ainsi même si un médicament est géré de manière différente dans les différentes parties de notre solution, nous avons une base commune, c'est-à-dire la forme canonique de référence d'un médicament.

5.1. Forme canonique d'un médicament

Avant de nous concentrer sur la notion de médicament de manière précipitée nous avons pris le temps de réfléchir de manière globale ainsi nous sommes partis d'une maladie, d'un traitement, pour arriver à un médicament, une posologie et une fréquence. Cette réflexion nous amené sur un concept de médicament très pointu. Cependant, après discussion avec des personnes travaillant dans le domaine de la santé, nous avons modifié notre conception « pointue » et « exacte » d'un médicament. Nous avons aussi pu profiter de cette expérience pour définir une solution qui serait au plus proche d'une solution réelle et surtout utilisable.

Sur cette base nous avons défini qu'un médicament se devait d'avoir les composantes suivantes :

- Un nom qui indiquerait au patient quel médicament prendre,
  à titre d'exemple cela pourrait être : aspirine.

- Un horaire c'est-à-dire un moment de la journée ou le médicament doit être pris, par exemple midi.
  Nous avons préféré diviser la journée en moments au lieu d'inclure une notion de fréquence, ce choix est expliqué plus bas dans la définition d'un moment de la journée.

- Des informations complémentaires indiquant au patient si le médicament possède des contre-indications, des remarques complémentaires comme par exemple : « à prendre avant le repas ». Ces remarques sont bien évidement individuelles ainsi elles pourront être beaucoup plus utiles que si elles étaient générales.

A ces composantes nous avons rajouté une notion de date, ceci facilite la gestion de la durée d'un traitement, ainsi pour chaque jour du mois nous pouvons facilement dire quels médicaments prendre. Ceci couplé avec la notion de moment de la journée nous permet de définir de manière précise quel médicament le patient doit prendre un jour donné à une heure donnée avec une remarque sur comment il doit le faire. Nous avons aussi ajouté la notion de dosage, c'est-à-dire la quantité de médicament qu'il faut prendre pour un médicament donné à un instant donné.

Pour modéliser le nombre de médicaments que le patient doit prendre par jour nous avons préféré modéliser ceci avec des moments de la journée, à savoir :

Nuit	00h00	à	03h59
Aube	05h00	à	07h59
Matin	08h00	à	11h59
Midi	12h00	à	15h59
Après-midi	16h00	à	19h59
Soir	20h59	à	23h59

Nous avons divisé une journée en 6 moments avec une correspondance en français, ainsi cette solution est plus humaine qu'une simple heure ou intervalle. Nous avons retenu cette solution par rapport à la notion de fréquence pour plusieurs raisons :

1. Pour une personne qui doit prendre beaucoup de médicaments avec des fréquences diverses, il est plus facile de s'y retrouver avec des moments dans la journée.

2. Il est plus facile pour le médecin de placer des médicaments aux heures des repas avec ce système plutôt qu'avec une fréquence. Il est aussi possible de mettre des médicaments en dehors des heures des repas.

3. Ce système permet d'établir une routine de prise de médicaments en accord avec l'ordonnance du médecin. Cela amène une structure. Toutes les 4 heures on change de moment de la journée, ceci permet aux patients d'adapter le système à leurs modes de vies.

4. Les moments de la journée spécifiant les heures de prises des médicaments, il est plus simple pour un médecin de planifier un médicament à prendre que le soir, ou

que le matin par exemple. Alors que la notion de fréquence aurait besoin d'un renseignement supplémentaire.

Voici ce à quoi ressemble notre forme canonique de médicament

Nom *Nom du médicament*
Horaire *Moment de la journée où le médicament doit être pris*
Date *Date de prise du médicament.*
Dose *Quantité de médicament à prendre.*
Infos *Informations complémentaires sur un médicament, remarques, ou contre-indications.*

Remarque : Il est à noter que ceci est une définition de base d'un médicament, étant donné que chaque partie de l'application gère des informations différentes dans des conditions différentes, il est logique que la définition de médicament évolue.

6. Site Web

6.1. Technologies et leurs utilisations

Dans la partie web du projet AM-PM différentes technologies ont été utilisées. Nous allons ici en faire une description et préciser le contexte dans le quel ces technologies sont utilisées. Nous avons essayé d'exploiter au maximum chaque technologie et de bien séparer leur utilisation.

6.1.1. HTML

Dans la mesure où il s'agit d'une application web, il aurait été difficile de contourner l'HTML. Il s'agit d'un langage de balisage qui permet d'écrire de l'hypertexte, d'où son nom « Hypertext Markup Language ».

Il nous est utile ici pour formater le contenu statique de nos pages web. Nous construisons ainsi le squelette de notre site sur lequel nous ajouterons les feuilles de styles CSS et le contenu, aussi bien statique que dynamique.

6.1.2. CSS

Le CSS, Cascading Style Sheet, traduit par feuille de style en cascade en français, est un langage informatique permettant de décrire la présentation des documents HTML et XML.

C'est le langage que nous allons utiliser pour appliquer un style à nos différentes pages web. Il nous permet ainsi de compenser les lacunes que nous trouvons dans le langage HTML quant au rendu esthétique des pages.

6.1.3. PHP

Figure 5 - Logo PHP

« PHP (sigle de PHP: Hypertext Preprocessor), est un langage de scripts libre principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale » [1].

Nous allons ici nous en servir pour communiquer avec la base de du site et la génération de contenu dynamique.

Les différentes classes utilisées pour le site sont étroitement liées à la description de notre base de données. Ceci nous permet ainsi de garder les mêmes concepts aussi bien dans la base de données que dans les classes PHP.

6.1.4. JavaScript

Nous pouvons dire que le JavaScript représentera le cœur de notre application web dans la mesure où il nous permettra de rajouter un côté dynamique à notre page HTML mais surtout parce qu'il établira le lien entre l'utilisateur, qui voit notre site depuis son navigateur, et la base de données où sont contenus l'ensemble des informations.

De plus, c'est par le biais de ce langage que nous allons contrôler les objets Ajax.

6.1.5. Ajax

Grâce à l'Ajax, Asynchronous JavaScript And XML, qui est une technique d'accès aux informations de JavaScript, nous pouvons, selon les désirs de l'utilisateur, accéder et afficher certaines informations directement depuis la base de données sans avoir à recharger la totalité du contenu de la page.

6.1.6. MySQL

Figure 6 - Logo MySQL

MySQL est un serveur de bases de données SQL (Structured Query Language) Open Source, rapide, robuste et multiutilisateurs. Selon le type d'application, sa licence est libre ou propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels.

6.2.Organisation des répertoires

Afin de garder une certaines organisation dans la structure de nos répertoires nous avons choisis de séparer ces derniers selon l'utilisation que nous en faisons. La structure est présentée dans la [Figure 7].

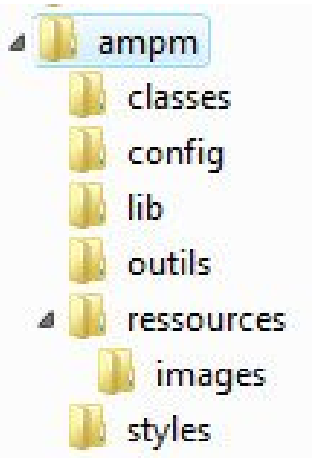

Figure 7 - Structure des répertoires

- classes
  - dossier contenant toutes nos classes PHP
- config
  - dossier contenant les différents fichiers de configurations de l'application
- lib
  - dossier contenant les librairies JavaScript
- outils
  - dossier contenant les différents fichiers appelés par le JavaScript
- ressources
  - dossier contenant les ressources de l'application
- ressources/images
  - dossier contenant les différentes images utilisées
- styles
  - dossier contenant les styles de l'application

Afin de protéger l'accès à nos scripts nous avons placés à la racine de notre application un fichier *.htaccess* empêchant l'accès direct aux différents fichiers.

6.3.Conventions utilisées

Afin de garder une certaine concision dans notre programmation nous avons choisis des conventions de nommage pour les fichiers mais aussi dans notre code.

C'est ainsi que tous nos fichiers de classes commencent par le nom du fichier suivi de .class puis de l'extension.

Toutes les actions sont symbolisées par des verbes, par exemple l'action d'ajouter un patient *ajouterPatient()*.

En ce qui concerne les noms de variables nous avons utilisé la convention dromadaire *maVariable* et la mise en majuscule des constantes *MACONSTANTE*.

6.4.Fonctionnement de l'application

Notre application nécessite que le JavaScript soit actif sinon un message d'erreur invite l'utilisateur à l'activé.

Une fois le JavaScript actif nous avons plusieurs cas :

- Patient,
- Médecin,
- Super utilisateur,
- Utilisateur inconnu

Nous allons voir les déroulements possibles pour chaque type d'utilisateur.

6.4.1. Procédure de connexion

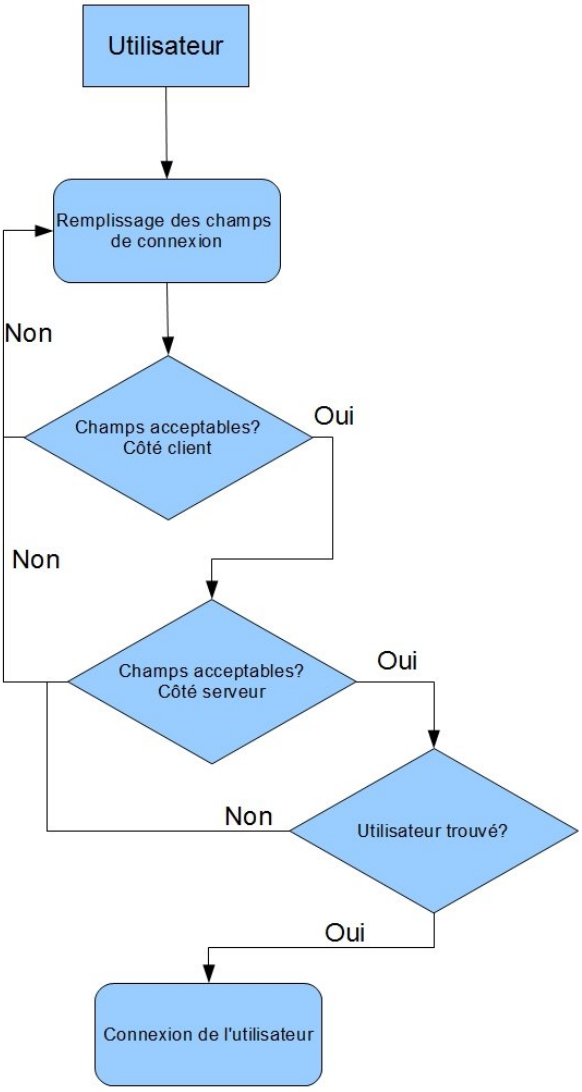

Figure 8 - Procédure de connexion

Lors de la procédure de connexion nous avons prêté attention à ne pas donner d'informations à l'utilisateur sur l'erreur de connexion en lui disant que le nom d'utilisateur est inconnu ou que le mot de passe est inexacte. Il y a un contrôle des informations entrées dans le formulaire tout d'abord du côté client en JavaScript puis ensuite du côté serveur en PHP. Si ces informations sont acceptables, nous continuons avec la procédure de connexion et nous allons vérifier la présence du couple login/mot de passe dans la base de données. Si ce couple est trouvé, nous initialisons les variables de sessions avec l'identifiant de l'utilisateur, son identifiant de rôle, son nom et son prénom. Le rôle, le nom et le prénom sont stocké en session ceci afin d'éviter de faire des requêtes supplémentaires pour aller les récupérer.

Le mot de passe de l'utilisateur est quand à lui haché avec la méthode SHA-512. Cette dernière est une fonction de hachage cryptographique fournissant une empreinte de 512 bits. Nous avons préférés cette méthode plutôt que l'ancienne SHA-1 afin de garantir un plus haut niveau de sécurité étant donné que l'empreinte est plus importante.

Le mot de passe entré par l'utilisateur lors de sa connexion est dans un premier temps haché en JavaScript avec la méthode SHA-512 ceci afin d'empêcher l'envoi en clair au serveur du mot de passe. Cet envoi en clair pourrait être problématique si par exemple une personne snifferait les paquets envoyés lors de la connexion, il pourrait récupérer le mot de passe de l'utilisateur en clair. Pour le moment notre méthode de connexion n'est pas optimisée au point de vue de la sécurité du fait qu'il est sensible aux replay attacks. Un utilisateur en mesure de récupérer le paquet contenant la demande de connexion pourrait reproduire l'envoi du paquet ce qui lui permettrai de se connecter.

Une fois l'utitilisateur connecté il pourra voir la réussite de sa connexion par le message de notification et l'affichage de son nom et prénom sur le site [Figure 9].

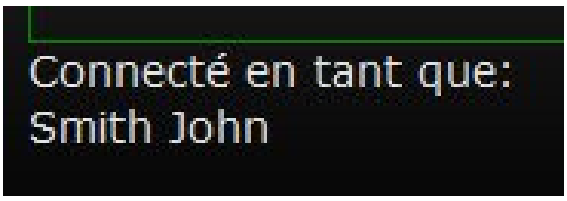

Figure 9 - Informations de connexion

6.4.1.1. *Patient*

Pour le moment le patient peut voir les différentes maladies qui sont en cours de traitements ainsi que les médicaments s'y rapportant.

6.4.1.2. *Super Utilisateur*

Le super utilisateur est l'utilisateur pouvant ajouter, modifier et supprimer un médecin, voir la liste des médecins ou encore recherche un médecin en entrant son nom.

6.4.1.3. *Médecin*

Le médecin est la partie principale de l'application.

Ce dernier peut :

- ajouter une nouvelle maladie,
- ajouter un nouveau médicament,
- ajouter un patient,
- ajouter un tuteur
- ajouter une maladie à un patient
- ajouter un médicament à une maladie d'un patient
- modifier un patient
- rechercher un patient
- voir la liste de tous ses patients

L'utilisation de l'Ajax pose encore ici un problème. L'utilisateur ayant accès au fichier source contenant le code JavaScript, il lui sera facile de trouver une requête d'affichage de patient de tester un identifiant et de récupérer ainsi les informations de ce patient. Il a donc été important de sécuriser les requêtes SQL effectuées afin de ne pas autoriser à un utilisateur l'accès à des patients ne lui appartenant pas.

6.4.1.4. Utilisateur inconnu

Un utilisateur inconnu est soit un utilisateur enregistré mais ayant entré des informations de connexion erronées, soit un visiteur de passage.

6.4.2. Procédures d'ajout

Toutes les procédures d'ajout de notre application fonctionnent sur le même principe. Un formulaire est présenté à l'utilisateur. Ce formulaire est tout d'abord vérifié en JavaScript puis finalement passé au serveur pour une deuxième validation. Si les informations sont correctes l'ajout s'effectue.

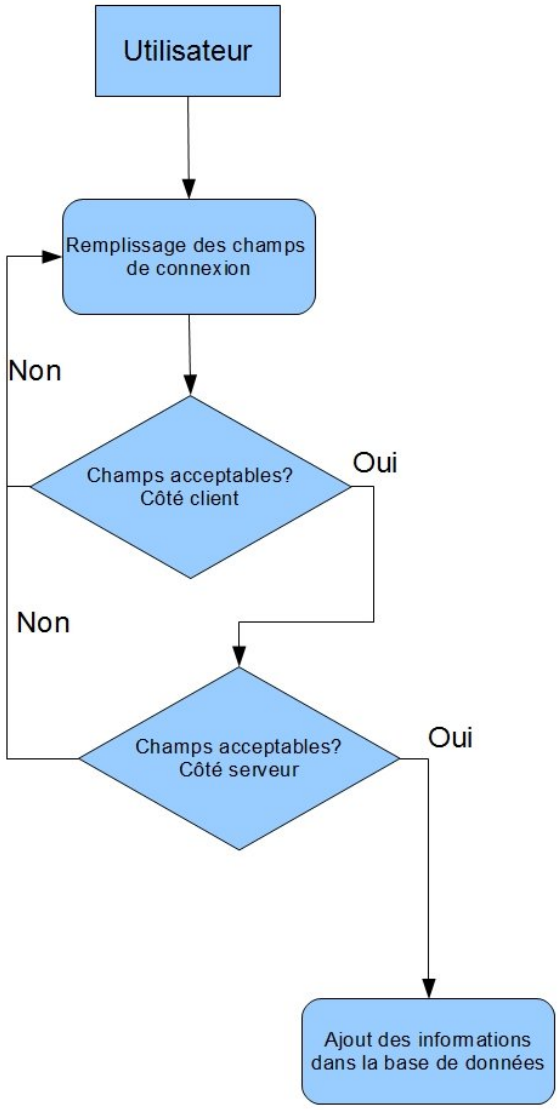

Figure 10 - Procédure de connexion

6.4.3. Procédure de modification

La modification s'effectue de la même façon que l'ajout à la différence prêt qu'il nous est nécessaire de passer l'identifiant de ce qui va être modifié. Ici aussi il est critique de contrôler le droit de l'utilisateur à effectuer l'action.

6.4.4. Procédure de suppression

Il est important ici de bien contrôler l'identifiant qui est envoyé par le client ceci afin de ne pas supprimer des données non permises. Cette vérification est faite du côté serveur avant l'exécution de la requête.

6.4.5. Procédure de recherche

Afin de faciliter la recherche d'un utilisateur, nous avons créé des fonctions de recherche en direct. Ces fonctions recherchent dans la base de données se basant sur le nom de l'utilisateur. Elles sont particulièrement appréciées pour leur rapidité d'exécution et leur filtrage rapide de l'information.

6.4.6. Affichage des informations

Lors d'une recherche de patient, une fois ce dernier trouvé, le médecin a directement accès aux informations primordiales concernant le patient. Cet affichage se fait par tooltip comme lors d'un passage de la souris sur une image faisant apparaitre le contenu du title [Figure 11].

Figure 11 - Information sur le patient

Nous avons ensuite accès aux différentes maladies du patient s'il en a et là encore il est possible de voir directement les informations sur la maladie [Figure 12].

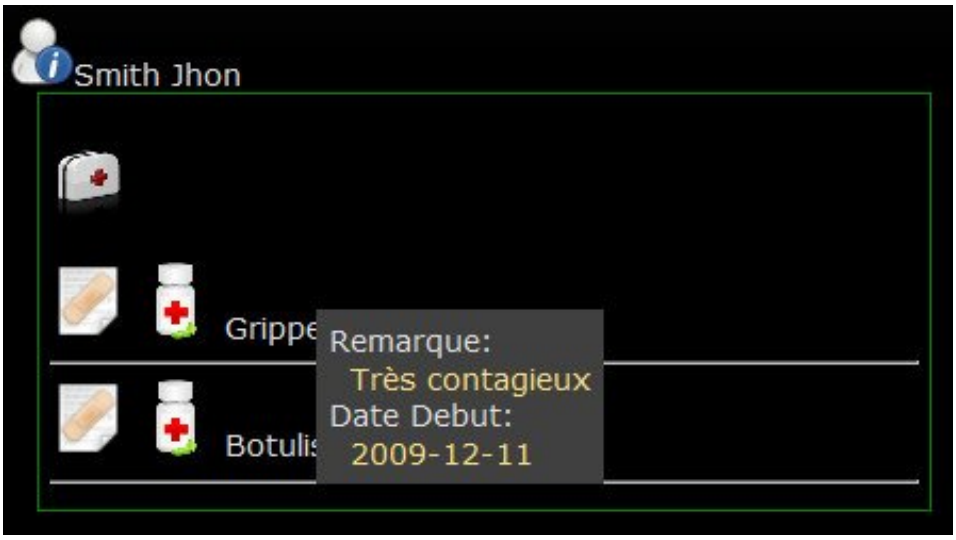

Figure 12 - Informations sur la maladie du patient

Viennent finalement les médicaments où nous aurons sur le même principe les informations utiles [Figure 13].

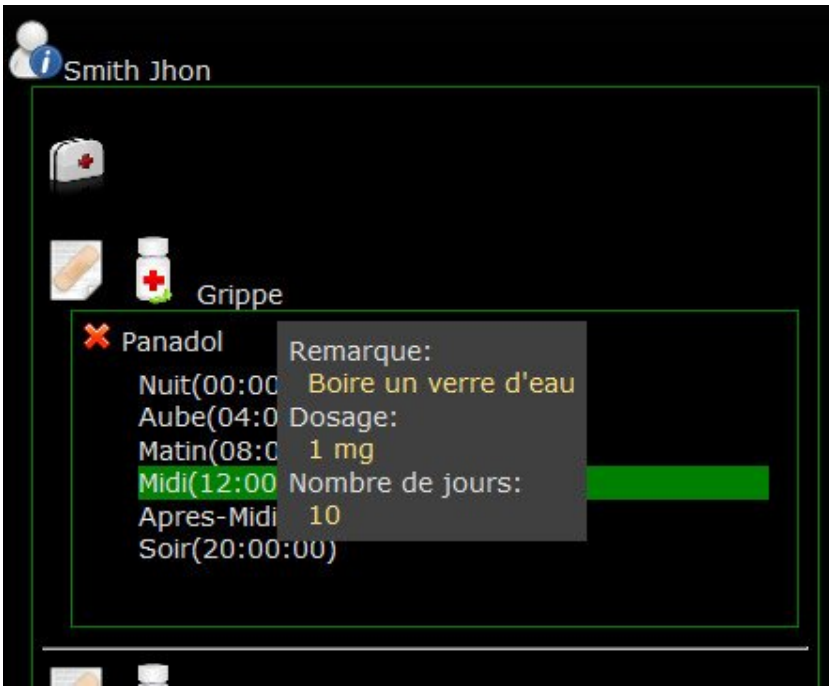

Figure 13 - Informations sur le médicament

6.4.7. Procédure de déconnexion

Nous avons choisi de fixer un timeout de session ceci apportant une sécurité supplémentaire si l'utilisateur oublie de fermer sa session. Ce timeout peut être modifié par le responsable du site. Il peut être aussi efficace de pouvoir gérer le timeout en imaginant qu'un poste doive garder une session ouverte pendant un intervalle de temps important, ou au contraire un intervalle minimal.

6.4.8. Optimisation du chargement des scripts

Nous avons pris garde a séparé au maximum chaque groupe d'actions propre à un groupe d'utilisateur dans des fichiers séparés. Ceci nous permet donc de ne charger que les fichiers de script nécessaire pour le groupe d'utilisateur connecté. Ainsi un patient ne verra jamais les fonctions propres au médecin. Une sécurité est aussi placée dans chaque fichier afin de s'assurer qu'à tout moment l'utilisateur est bien connecté et qu'il fait partie du bon groupe.

Nous avons aussi choisi de compresser nos fichiers JavaScript ainsi que de les obfusquer rendant leur lecture plus complexe à un néophyte. Pour ce faire nous avons utilisé le site JSCompress [6] qui nous permet facilement de le faire.

6.4.9. Choix de la présentation

Tous les aspects visuels de l'application peuvent êtres modifiés au travers des fichiers de style. Ceci permet une personnalisation de l'application en fonction de la demande.

6.5.Respect des normes

Nous avons mis un point d'honneur à respecter les différentes recommandations formulées par le W3C, le World Wide Web Consortium organisme de standardisation des technologies du World Wide Web. Notre site est donc valide XHTML 1.1, ceci nous permettant de fournir un

document tourné vers le futur séparant proprement les fonctionnalités à caractère obsolète de HTML4.

Un autre point essentiel a été de fournir une application cross-browser. Ceci a impliqué un travail plus important mais une application web destinée uniquement à un seul type de navigateur n'était pas dans notre optique.

Nous avons donc testé avec succès cette application sur les navigateurs suivants :

- Internet Explorer 7 et 8
- Opéra 9 et 10
- Mozilla Firefox 3.0 à 3.5
- Safari 4
- Google Chrome 3

6.6.Framework JQuery

JQuery est un Framework simplifiant le parcours du document HTML, la manipulation du DOM, la gestion des événements, les animations et l'Ajax. Nous avons choisi ce Framework du fait de sa petite taille, sa compatibilité entre les différents navigateurs et la base de plugins important qui y sont rattachés.

6.7.Gestion des requêtes

Afin de minimiser au maximum l'utilisation de la bande passante, nous avons choisi d'utiliser de l'Ajax nous permettant de minimiser la quantité d'information à télécharger. Nous restons donc sur la même page tout au long de notre navigation et chargeons uniquement les informations qui nous sont utiles.

Seul petit problème, nous avons choisi de ne pas gérer la possibilité d'utiliser le bouton back permettant de revenir à l'action précédente. Ce choix a été pris à cause du temps limités que nous avions à disposition. Afin de le mettre en place nous avons pensé à deux solutions. La première consiste à fournir un bouton suivant et un bouton précédent dans l'application afin de gérer ceci, mais là encore nous avons un problème : la mise en favoris du lien. Nous sommes donc venus à une autre solution qui serait de changer l'url en y ajoutant un paramètre et ainsi de permettre la mise en favoris et les fonctions suivants/précédents.

En Ajax il nous est possible de faire soit des requêtes synchrone, ceci entrainant la suspension de toutes activité du navigateur jusqu'à la fin de la requête, et des requêtes asynchrone, ceci nous permettant d'envoyer plusieurs requêtes en même temps au serveur.

Le premier cas ne convenait pas à notre application car nous ne voulions pas bloquer l'utilisateur le temps de la requête. Nous avons donc opté pour la deuxième solution tout en gérant le fonctionnement de ces requêtes.

Nous avons donc définis des requêtes prioritaires et d'autres pouvant êtres interrompues avant la fin de leur exécution. Nous avons choisis de définir les requêtes prioritaires comme des requêtes entrainant une modification de la base de données, toutes les requêtes d'insertions,

suppressions, mise à jour. Les autres requêtes, sélection par exemple, quand à elles peuvent être interrompues avant la fin de leur exécution.

Ceci nous permet de régler un problème souvent rencontrés en Ajax. Prenons un exemple simple. Le médecin choisi d'afficher les informations d'un patient et choisi juste après d'afficher le formulaire d'ajout d'une nouvelle maladie. Le formulaire d'ajout de la maladie va s'afficher car la requête prend moins de temps à s'exécuter mais la première requête est toujours entrain de s'effectuer. Une fois la première requête terminée, le formulaire d'ajout de maladie va être remplacé par la fiche détaillées du patient.

Dans notre cas ceci ne peut se produire car lors de l'envoi de la deuxième requête, la première est arrêtée grâce à la méthode *abort()* de l'objet XMLHttpRequest retourné par la fonction *$.ajax()*.

Ceci est encore plus parlant lors d'une recherche en direct d'un patient en entrant son nom. Chaque relâchement de touche, relevé avec l'événement JavaScript *onKeyUp()* entraine l'envoi d'une requête. Ainsi après avoir entré deux lettre, la première requête sera stoppée au profit de la deuxième.

Nous avons aussi pensé à créer un système détectant si un problème réseau est peut être existant. Pour ce faire, nous avons incorporé à chaque requête un timeout. Lorsqu'un certain nombre de requêtes n'aboutit pas, un message d'erreur est affiché à l'utilisateur lui indiquant le problème possible [Figure 14]. Ces deux paramètres peuvent bien sûr êtres modifiés pour le moment par la personne responsable de l'installation de l'application.

Figure 14 - Message de notification de problème réseau

6.8.Installation de l'application

Afin de faire fonctionner cette application il nous faut un serveur supportant au minimum le PHP5 car nous avons utilisé la programmation orientée objet proposée par cette version et non présente dans les versions antérieures, ainsi qu'une base de données MySQL version 5 et supérieure.

Apres création et importation de la base de données il sera nécessaire de compléter le fichier de configuration de base de données avec les informations adéquates.

6.9.Ergonomie

Tous les utilisateurs n'ayant pas de connaissances accrues dans le maniement des différentes technologies, nous nous sommes donné comme objectif de concevoir une application simple et efficace facilitant son utilisation aux personnes néophytes. Chaque action entrainant une altération de la base de données ou la connexion/déconnexion de l'utilisateur est sujette à la notification de cette dernière à l'utilisateur par un message coloré l'informant directement visuellement et textuellement si une erreur est survenue ou si l'action s'est déroulée correctement [Figure 15].

Figure 15 - Notification d'une action correcte

Chaque champ de formulaire, une fois sélectionné par l'utilisateur va afficher une information indiquant ce à quoi correspond le champ, ceci facilitant la saisie [Figure 16].

Figure 16 - Informations sur le champ

Lors de la soumission, au même endroit, un conteneur de couleur différente prévient l'utilisateur si un champ est rempli de façon incorrecte [Figure 17].

Figure 17 - Erreur de remplissage du champ

Nous avons aussi intégré de petits effets visuels non dérangeant améliorant l'ergonomie de l'application.

6.10. Choix de l'encodage

Bien que pour le moment notre application ne soit pas multilingue, cette dernière utilise le jeu de caractères UTF-8 afin d'être prête à l'internationalisation aussi bien du coté des pages PHP que dans la base de données. Cette préparation est bien plus avantageuse à faire des le début que de devoir rependre tout un projet et de procéder à la conversion.

6.11. Pensée écologique

L'écologie étant au centre de toutes les discussions ces dernières années nous avons nous aussi apporté notre petite contribution en choisissant un fond noir pour notre application ceci diminuant la consommation énergétique. C'est un petit geste mais nous tenions à le faire.

6.12. Base de données AM-PM

Pour que notre application web puisse fonctionner nous avons besoin d'installer la base de données. Pour le moment cette dernière doit être importée manuellement depuis le fichier dump mis à disposition.
Il suffit de créer une base de données de votre choix et d'y importer le fichier dump. Une fois cette opération terminée, il reste à informer les paramètres de configuration de l'application avec les informations entrées précédemment.

6.12.1. Modèle conceptuel / relationnel

Cette base de données permettra la gestion des différents utilisateurs et actions selon le schéma conceptuel [Figure 18].

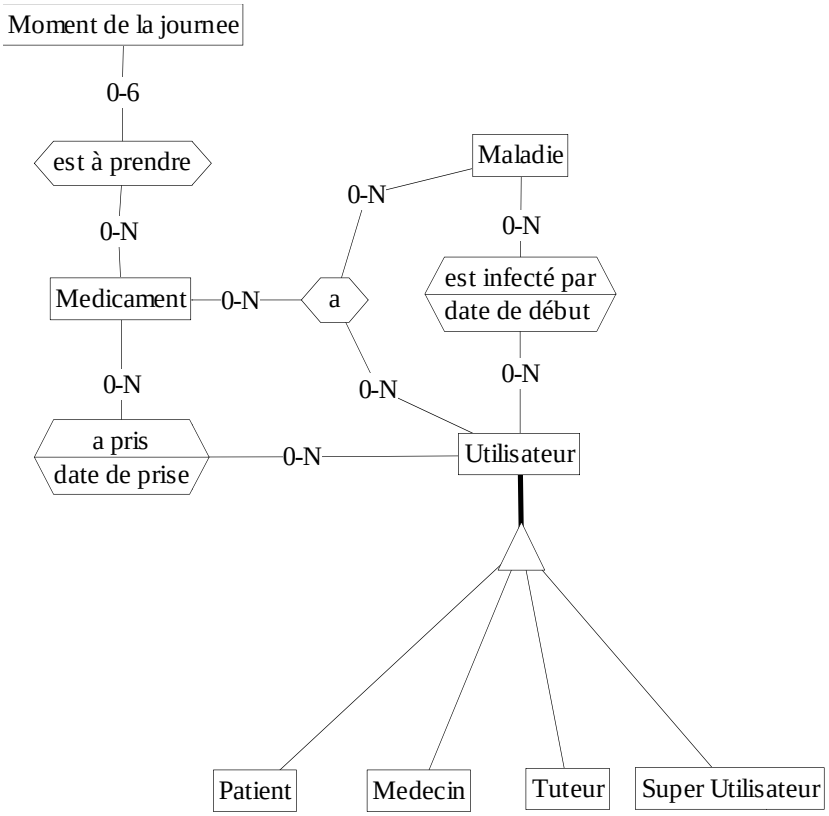

Figure 18 - Modèle conceptuel de la base de données AM-PM

Chaque entité a été séparée au maximum afin de garder son concept. Les relations nécessaires ont ensuite été créées afin de former les liens entre les entités.

Nous avons choisi de créer des identifiants virtuels sous forme de valeur entière pour chaque entité ceci nous permettant un accès plus facile aux informations. Nous ne travaillerons donc pas sur des chaines de caractères mais sur des entiers.

Dans notre application il y a plusieurs types d'utilisateurs chacun avec des privilèges différents. Un utilisateur a donc un rôle qui défini ce que ce dernier peut faire.

Nous avons pris la décision qu'un patient ne peut avoir qu'un seul et unique tuteur mais le même tuteur peut avoir plusieurs pupilles.

Nous avons aussi choisis de garder dans la base tous les patients qui ont étés enregistrés ceci par exemple afin de garder un historique et une traçabilité des informations.

Les différentes valeurs de champ ont été choisies de telle sorte à garder l'information de base. Par exemple la date de prise d'un médicament a été définie de type DATETIME ceci permettant d'éventuels calculs directement depuis la base de données.

Lors de la mise à jour d'un identifiant, cette mise à jour est propagée dans toutes les relations concernées tandis que lors d'une suppression, si un identifiant est présent dans une relation, l'action est bloquée. Ceci a été introduit afin d'introduire une sécurité supplémentaire non pas seulement pour une éventuelle envie de détruire les données mais aussi pour une manipulation erronée et non prévue du coté client et/ou serveur. Par exemple un super se trompant de médecin et voulant le supprimer. Si le médecin est en relation avec d'autres utilisateurs, l'action ne sera pas permise.

6.12.2. Etude des relations

Nous allons nous intéresser maintenant en détail à chaque relation.

6.12.2.1. Relation de tuteur

Un utilisateur a un et un seul tuteur.

6.12.2.2. Relation de médecin

Un utilisateur a un et un seul médecin. Ceci pourrait être étendu avec une notion de médecin responsable et ensuite médecin auxiliaires.

6.12.2.3. Relation de maladie

Un utilisateur a une maladie avec une date de début de la maladie éventuellement une remarque et une précision sur l'état de la maladie, si elle est soignée ou non. Ce qui va identifier un tuple vont être : l'utilisateur, la maladie et la date de début de la maladie. Ceci nous permet donc de donner plusieurs fois la même maladie à un patient à des dates différentes. Le patient peut avoir eu la grippe le 10 janvier 2009 et de nouveau le 10 janvier 2010 même si nous ne lui souhaitons pas ceci. Dans cette relation, l'utilisateur étant déjà lié à un médecin, il ne nous est pas utile d'insérer l'identifiant du médecin dans la relation. Ceci serait une information redondante.

6.12.2.4. Relation de médicament

Un utilisateur ayant une ou plusieurs maladies a un ou plusieurs médicaments une date de début de prise du médicament, un nombre de jours pendant lequel il doit le prendre, un dosage et enfin une remarque.

6.12.2.5. Relation de moment de la journée de prise du médicament

Un utilisateur ayant une maladie et un médicament pour cette maladie a un ou plusieurs moments de prise.

Nous voyons donc que nous avons essayé au maximum d'éliminer les redondances d'informations dans nos tables et relations. Nous avons aussi commencé à réfléchir au fait qu'un médecin pourrait être un patient de même que pour un tuteur c'est pourquoi nous avons laissé toutes les personnes dans la table utilisateurs.

7. Partie serveur central

Pour faire le lien entre la base de données, que nous avons vu précédemment, et le mobile du patient, que nous verrons un peu plus tard, nous avons mis en place un serveur. Dans cette partie, nous allons donc présenter l'infrastructure complète que nous avons établie pour gérer ces communications, ainsi que les implications qu'elle a du côté serveur et du côté client.

7.1.Protocole de communication

Dans la mesure où il s'agit de deux systèmes distants, il nous a fallu mettre en place un protocole permettant la communication entre le serveur et l'application cliente. Des messages sont ainsi échangés entre les deux entités.

7.1.1. Messages

Les messages sont échangés entre le serveur et l'application cliente du patient. Ces messages contiennent les informations suivantes :

- Id de session : Comme nous le verrons un peu plus tard, chaque communication avec le serveur doit faire partie d'une session.
- Commande : Il s'agit de l'action demandée. Pour le moment, et sans que cette liste se veuille exhaustive, les commandes possible sont OPEN pour ouvrir la session, CLOSE pour fermer la session, GET pour demander une information, et SEND pour envoyer une information.
- Option : Dans la mesure où il est possible d'avoir plusieurs opérations par commandes, c'est dans cette partie du message que nous précisons quelle opération nous voulons effectuer.
- Arguments : C'est ici que nous trouvons les informations nécessaires à la réalisation de la commande, ou bien, dans le cas d'un SEND, les informations demandées.

Ce qui nous donne un message qui ressemble au schéma suivant :

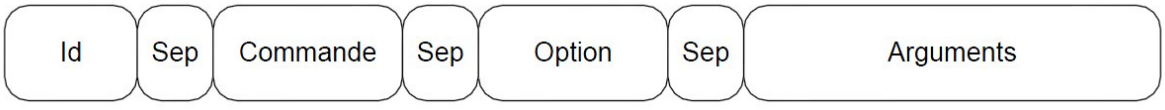

Figure 19 - Représentation schématique d'un message

Afin de faciliter la lecture et la séparation des différentes parties du message, nous avons inclus des séparateurs, représenté sur le schéma par le mot « Sep ». Ainsi, lors de la réception d'un message, l'application, serveur ou client, peut générer un objet message à partir de cette chaîne de caractères.

Dans les parties suivantes, nous allons voir les différentes séquences de message qui interviennent entre le serveur et le client, ainsi que les opérations qui sont réalisées par chacune de ces applications.

7.1.2. Session

Comme expliqué avant, la première chose que doit effectuer le client lorsqu'il désire se mettre à jour est d'obtenir du serveur une id de session. Au delà du fait qu'il soit nécessaire d'obtenir cette id pour respecter le protocole, cette étape sert également à la sécurité de l'application puisque c'est ici que nous allons identifier et authentifier le patient. Nous avons ainsi une séparation complète entre cette partie et la partie où l'utilisateur, une fois identifié et authentifié, effectue les demandes de mise à jour à la base de données.

Le client va donc dans un premier temps envoyer sa demande au serveur dans un message formaté selon le protocole. Sur réception de ce message, le serveur va chercher à identifier, et authentifier, l'utilisateur en accédant à la base de données. L'objectif de cette opération est de pouvoir avoir dans la table d'id que le serveur possède une association directe entre l'id de session que l'utilisateur va avoir et l'id que cet utilisateur dans la base de données ; ainsi il ne sera plus nécessaire par la suite d'aller chercher pour chaque message de la session les informations sur ce patient. Ceci nous permet donc de réduire le nombre de requêtes effectuées à la base de données. Une fois cette association réalisée, l'id de session est retournée au client qui pourra alors la sauvegarder pour les prochains messages de cette session.

La suite des opérations réalisées par chacune des applications est retrouvée dans le tableau suivant :

La suite des opérations réalisées par chacune des applications est retrouvée dans le tableau suivant :

Client	Serveur
1. Le client envoie un message d'ouverture de session au serveur avec les informations suivantes: • Id = 0 • Commande = OPEN • Option = CONNEXION • Arguments = IMEI (information permettant d'identifier le patient)	
	2. Le serveur crée un nouveau thread pour traiter le message reçu

	3. Le serveur crée un nouvel objet Request à partir de la chaîne de caractères reçue • L'objet n'a pas pu être correctement créé, un message d'erreur est retourné au client pour que celui puisse fermer sa connexion 4. Le serveur, dans le cas d'une demande OPEN, vérifie que celle-ci est bien effectuée avec une id = 0. • Id est différent de 0, un message d'erreur est retourné au client 5. Le serveur, en utilisant l'IMEI fourni va chercher l'id correspondant à l'utilisateur dans la base de données. • L'utilisateur n'a pas pu être trouvé dans la base de données, un message d'erreur est retourné au client. 6. Le serveur stocke l'id de l'utilisateur dans la table des id et lui associe une id de session. • L'utilisateur est déjà présent dans la table des id, l'ancienne id de session est retirer de la table et la nouvelle id de session est rajoutée dans la table à la place 7. Le serveur renvoie en réponse le message avec les paramètres suivants: • Id = id de session associé à l'id de l'utilisateur dans la base de données • Commande = OPEN • Option = CONNEXION • Arguments = NULL
8. Le client crée un nouvel objet Response à partir de la chaîne de caractères reçue. • L'objet n'a pas pu être correctement créé, fin de la procédure et attente d'une intervention humaine. 9. Le client vérifie que tous les champs sont comme attendus • Un champ n'est pas conforme, fin de la procédure et attente d'une intervention humaine. 10. Le client sauvegarde l'id de la session pour	

les requêtes suivantes.	

Tableau 1 - Diagramme de séquences pour l'obtention de l'id de session

6.2.3 Mise à jour du client

Une fois l'id de session obtenue, l'application cliente dispose de toutes les informations nécessaires pour obtenir du serveur et de la base de données sa mise à jour.

Cette partie se réalise alors en deux étapes distinctes.

Dans un premier temps, il s'agit d'obtenir la liste des médicaments que le patient doit prendre. Ici le client fait une demande avec la commande GET et l'option UPDATE. Sur réception de cette dernière, le serveur va chercher à obtenir les informations condensées sur ces médicaments, c'est-à-dire l'id du médicament, l'heure de prise et enfin la date de la première prise. Une fois cette liste récupérée, le serveur répond au client avec cette fois la commande SEND et la liste des médicaments dans la partie arguments du message.

Sur réception de cette réponse, le client va mettre à jour sa base de données local et vérifier si pour tous les médicaments qu'il vient de recevoir il a une définition valide. En effet pour chaque médicament, il faut avoir le nom du médicament afin de pouvoir l'afficher correctement, et la posologie ainsi que les indications et contre-indications pour pouvoir les indiquer au patient.

C'est donc dans cette deuxième étape que l'application cliente du patient va comparer les médicaments de la liste qu'il reçoit avec ceux qu'il a déjà en local. Pour tous les nouveaux médicaments, il va donc envoyer une demande afin de compléter les informations de ces derniers. Le message envoyé contient toujours la commande GET mais cette fois avec l'option DEFINITION et en argument l'id du médicament. Sur réception du message, le serveur va donc aller chercher dans la base de données, à partir de l'id du patient et du médicament les informations nécessaires. De même que pour la réponse précédente du serveur, le message a la valeur SEND pour le paramètre commande et la définition en arguments.

Enfin, une fois que la liste des médicaments a été entièrement traitée, le client va également retirer de sa base de données les médicaments qui ne sont plus dans la liste afin de ne plus les réafficher dans les médicaments à prendre.

Les avantages de cette séparation en deux étapes sont évidents dans la mesure où il n'est plus nécessaire à l'application cliente de demander et de recevoir toutes les informations sur tous les médicaments quand elle a déjà pris les définitions de ces derniers. Dans le pire des cas, c'est-à-dire lors de la première mise à jour, l'application devra récupérer toutes les informations mais pour la suite, s'il n'y a pas de gros changements dans les prescriptions, il lui suffira de mettre à jour la liste des médicaments uniquement.

Dans la mesure où nous avons un débit très restreint sur la connexion internet du mobile comparé à ce que nous sommes habitués à avoir quand nous développons sur des ordinateurs, cette solution à l'avantage également de réduire le flux de données échangées au strict minimum, ce qui évite tous problèmes liés à une surcharge de la connexion.

Dans le tableau suivant, nous pouvons voir la suite des opérations réalisées par les deux applications lors de cette procédure de mise à jour.

Client	Serveur
1. Le client envoie une demande de mise à jour avec les paramètres suivants : • Id = Id de session • Commande = GET • Option = UPDATE • Arguments =NULL	
	2. Le serveur crée un nouveau thread pour traiter le message reçu 3. Le serveur crée un nouvel objet Request à partir de la chaîne de caractères reçue • L'objet n'a pas pu être correctement créé, un message d'erreur est retourné au client pour que celui puisse fermer sa connexion 4. Le serveur vérifie les différents paramètres. • Si un des paramètres n'est pas conforme au protocole, un message d'erreur est retourné au client. 5. Le serveur, en utilisant l'id de l'utilisateur associée à l'id de session en cours, extrait de la base de données et génère la liste contenant les informations réduites sur les médicaments à prendre. • La liste de médicament est vide, la chaîne de caractère normale est alors remplacée par NULL. 7. Le serveur renvoie en réponse le message avec les paramètres suivants: • Id = id de session associé à l'id de l'utilisateur dans la base de données • Commande = SEND • Option = UPDATE • Arguments = Liste des informations condensées sur les médicaments à

	prendre par le patient, ou NULL
8. Le client crée un nouvel objet Response à partir de la chaîne de caractères reçue. • L'objet n'a pas pu être correctement créé, fin de la procédure et attente d'une intervention humaine. 9. Le client vérifie que tous les champs sont comme attendus • Un champ n'est pas conforme, fin de la procédure et attente d'une intervention humaine. 10. Pour chaque élément de la liste, le client va mettre à jour sa base de données locale. • Si la liste est vide (arguments = NULL), passe directement au point 13. 11. Le client vérifie s'il a la définition pour tous les médicaments qu'il a dans sa base de données locale. a. S'il y a une définition manquante, le client envoie un message de demande de définition avec les paramètres suivants : • Id = id de la session • Commande = GET • Option = DEFINITION • Arguments = id du médicament	
	b. Le serveur effectue les mêmes procédures que dans les cas précédentes concernant la création d'un nouveau thread et la vérification des informations reçues. c. Le serveur, en utilisant l'id de l'utilisateur et l'id du médicament, va aller chercher dans la base de données les informations de définition du médicament demandé. d. Le serveur envoie en réponse le message avec les paramètres suivants : • Id = id de la session • Commande = SEND • Option = DEFINITION • Arguments = définition du médicament demandé
e. Le client vérifie si la réponse est conforme au protocole (si non	

conforme, passe au prochain médicament et s'arrête à la fin en attente d'une intervention humaine) f. Le client met à jour la définition du médicament dans sa base de données locale. 12. Le client vérifie retire de sa base de données les médicaments qui n'étaient pas dans la liste reçue.	

Tableau 2 - Diagramme de séquences pour la mise à jour du client

6.2.4 Mise à jour des médicaments pris

Une autre procédure qui peut être effectué pendant une session est l'envoie par le client de la liste des médicaments que le patient a bien pris. Le but de cette opération est de pouvoir suivre au niveau de la base de données la prise des médicaments par le patient et ainsi de donner des informations supplémentaires au médecin en lui permettant de mieux suivre le traitement de son patient.

Pour se faire, le client envoie un message avec comme paramètre de commande SEND et pour la valeur du paramètre argument la liste desdits médicaments. Sur réception de ce message, le serveur va alors récupérer l'id du patient et mettre à jour les informations de la base de données à partir de la liste des médicaments pris.

Pour cette opération, nous n'avons pas tenu à faire répondre le serveur, dans un premier temps.

Client	Serveur
1. Le client envoie une demande de mise à jour avec les paramètres suivants : • Id = Id de session • Commande = SEND • Option = UPDATE • Arguments =NULL	
	2. Le serveur crée un nouveau thread pour traiter le message reçu 3. Le serveur crée un nouvel objet Request à partir de la chaîne de caractères reçue • L'objet n'a pas pu être correctement créé, un message d'erreur est retourné au client pour que celui puisse fermer sa connexion 4. Le serveur vérifie les différents paramètres. • Si un des paramètres n'est pas

	conforme au protocole, un message d'erreur est retourné au client. 5. Le serveur, en utilisant l'id de l'utilisateur associée à l'id de session en cours ainsi que la liste des médicaments pris fournis en arguments du message, va mettre à jour les informations de la base de données.

Tableau 3 - Diagramme de séquence de la mise à jour des médicaments pris

Nous pouvons remarquer que cette opération peut avoir lieu juste avant la mise à jour des informations du client ou bien lorsque le patient prend une série de médicaments.

6.2.4 Fermeture de session

Une fois toutes les requêtes effectuées et l'application cliente complètement mise à jour, il est temps pour le client de fermer sa session. Ceci est effectué en envoyant un message avec la commande CLOSE. Sur réception de ce message, le serveur va aller chercher dans la table d'id de session, l'id de la session courante et la retirer. Il répondra alors un message contenant également la commande CLOSE mais cette fois l'id de session sera 0.

Ainsi, une fois la session fermée, il n'y a plus moyen d'accéder directement aux informations de la base de données pour cet utilisateur sans avoir à s'identifier de nouveau, ce qui rend plus difficile le vole de session et ainsi l'accès pour des personnes non-autorisées à des informations confidentielles sur les patients.

Client	Serveur
1. Le client envoie un message d'ouverture de session au serveur avec les informations suivantes: • Id = id de la session • Commande = CLOSE • Option = CONNEXION • Arguments = NULL	
	2. Le serveur crée un nouveau thread pour traiter le message reçu 3. Le serveur crée un nouvel objet Request à partir de la chaîne de caractères reçue • L'objet n'a pas pu être correctement créé, un message d'erreur est retourné au client pour que celui puisse fermer sa connexion 4. Le serveur, dans le cas d'une demande CLOSE, vérifie que celle-ci est bien effectuée avec une id différente de 0. • Id égale à 0, un message d'erreur est

	retourné au client 5. Le serveur retire l'id de session correspondante de la table d'id de session. • L'id de session ne se trouve pas dans la table, un message d'erreur est retourné au client. 6. Le serveur renvoie en réponse le message avec les paramètres suivants: • Id = 0 • Commande = CLOSE • Option = CONNEXION • Arguments = NULL
7. Le client crée un nouvel objet Response à partir de la chaîne de caractères reçue. • L'objet n'a pas pu être correctement créé, fin de la procédure et attente d'une intervention humaine. 8. Le client vérifie que tous les champs sont comme attendus • Un champ n'est pas conforme, fin de la procédure et attente d'une intervention humaine. 9. Le client réinitialise son id de session à 0 et devra donc redemander une id s'il veut communiquer de nouveau avec le serveur.	

Tableau 4 - Diagramme de séquences pour la fermeture d'une session

6.3 Evolution vers le chiffrement des messages

Comme prévu dans le cahier des charges, nous aurions dû mettre ne place dans notre protocole de communication une étape permettant de chiffrer les messages avant de les envoyer et de déchiffrer avant de les traiter.

Au stade de développement où nous sommes arrivés, cette option n'a pas encore implémenté.

Dans le système que nous avons, lors de l'ouverture de la session, nous voyons qu'on réussi à identifier l'utilisateur par un moyen très simple en fournissant un identifiant que seul lui est supposé connaître. Une des solutions auxquelles nous avions pensées se base sur le fait qu'une fois identifier, on pourrait utiliser un chiffrement par algorithme symétrique avec une clé aillant été configurée lors de l'installation de l'application sur le mobile. Ainsi, dès lors que la session est ouverte, les messages suivants seraient chiffrés par une clé connue par les deux parties, le client et le serveur. De plus, du côté serveur, cette clé de chiffrement serait récupérée dans la base de

données en même temps que l'id du patient et rajouter comme information complémentaire dans la table d'id de session.

Une autre solution que nous avions en tête était d'implémenter une connexion SSL entre le serveur dans la mesure où nous établissons la connexion en TCP. Notre protocole et nos séquences ne changeraient pas et nos informations seraient chiffrées et signées de manière à assurer la confidentialité et l'intégrité des informations échangées.

8. Partie mobile

La partie mobile est la partie que le patient, aussi appelé client, peut voir et toucher. C'est sur son téléphone qu'il va consulter la liste des médicaments et signaler par une action qu'il les a pris. La partie mobile se charge aussi de lui signaler quand il a des médicaments à prendre.

8.1.Choix de l'équipement mobile

8.1.1. Approche théorique

Les téléphones n'étant pas des ordinateurs au sens puissance et rapidité, il faut essayer de gérer un maximum de choses en dehors du téléphone, et ainsi garder une application mobile qui est rapide, et réactive. Cette contrainte nous oblige segmenter notre solution, pour arriver à une solution ou le mobile est certes au cœur, mais sa principale fonction est de servir d'interface entre le patient et le serveur de médicaments.

Avant de décider sur quelle plateforme nous voulions développer notre solution mobile, il nous fallait tout d'abord déterminer la liste des matériels sur lesquels nous pouvions distribuer notre solution. De nos jours une quantité incroyable de matériels portables sont équipés de dispositifs réseaux, nous avons ainsi déterminé notre ensemble de discussion à savoir :

1. Téléphones portables
2. Smartphones
3. PDA
4. Netbook
5. EbookReader

En ce qui concerne la plateforme de développement nous avons retreint notre ensemble de discussion à cinq plateformes:

1. Java2Me
2. dotNet
3. Symbian
4. PalmOS
5. Android

En sachant que les Smartphones ont toutes les qualités des agendas personnels, il est facile de prédire qu'ils vont sûrement remplacer les téléphones mobiles classiques ainsi que les PDA dans un futur proche. En se basant sur ce constat, nous pouvons donc éliminer deux des

cinq équipements portables susmentionnés. En considérant maintenant la taille des équipements, il est évident que le Smartphone est de loin le plus pratique pour notre projet. Celui-ci est léger et possède une autonomie suffisante pour satisfaire nos exigences.

8.1.2. Approche marketing

Avec une approche un peu plus commerciale, le choix de notre équipement est appuyé par le succès grandissant des Smartphones au près du grand public. Ainsi avec une solution basée sur Smartphone on est assuré que notre application ne tournera pas sur des appareils marginaux, ou sans avenir.

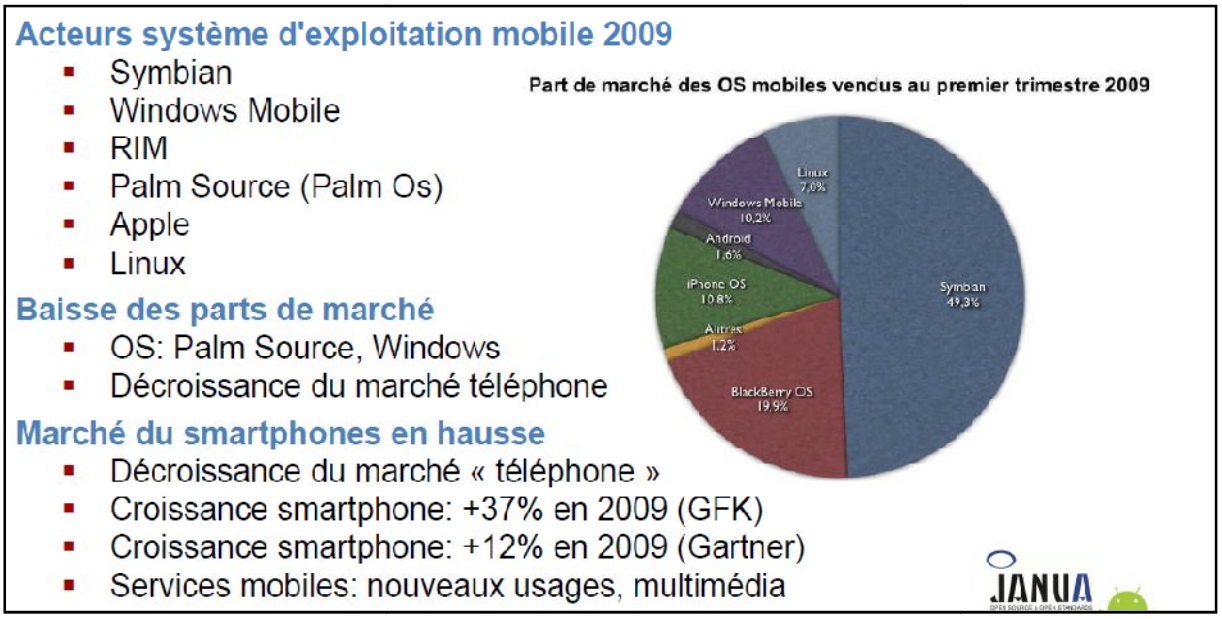

Figure 20 - Extrait de la présentation d'Android par la commission OpenSource - Telecom Valley / juillet 09

D'après l'extrait de la présentation d'Android ci-dessus, on constate que les Smartphones sont en pleine croissance, ceci au même rythme que les personnes adoptent des services mobiles. En effet, pour un prix presque identique les personnes peuvent tirer plus d'avantages d'un Smartphone que d'un simple téléphone, ne serait-ce que par le confort d'utilisation, ou par la facilité de synchronisation des données ou de connexion avec le PC. Ces nouvelles fonctionnalités facilitent la vie des professionnels et les capacités graphiques attirent les jeunes consommateurs, friands de jeux-vidéos et d'applications inutiles.

8.2.Choix de la plateforme

Ayant décidé de nous orienter vers un développement sur Smartphone, il nous faut encore décider sur quelle plateforme nous allons programmer notre solution, nous avons restreint notre ensemble aux plateformes suivantes : Java2me, Android ou Iphone.

8.2.1. Iphone

Vu le nombre de Iphones en circulation la plateforme d'Apple parait à première vue être une bonne solution pour notre application, cependant les restrictions de développement, les coûts entraînés par ces dernières et la politique fermée menée par Apple nous ont

découragés de développer sur Iphone. Nous pouvons aussi signaler qu'au contraire des deux autres plateformes retenues, l'Iphone ne tourne que sur des équipements Iphone.

8.2.2. Java2me

Java2me offre la puissance de java sur un large panel de téléphone portables cependant il n'y a pas de standard commun que tous les fabricants respectent. Ceci implique que nous avons une plateforme peu homogène.

De plus il y a certaines limitations qui nous on poussé à opter pour une autre plateforme signalons par exemple l'interface graphique pauvre, inesthétique et rudimentaire. Pour développer une interface conviviale sur java2me nous devons réaliser nous même l'intégralité des graphiques et l'interface GUI.

De plus comme signalé plus haut, les API de bas niveau étant très dépendantes du téléphone il serait possible que notre application refuse de fonctionner sur un type de téléphone nouveau, ceci impliquerait que nous devrions inclure les téléphones cibles dans nos tests, solution bien évidement très difficile à maintenir.

8.2.3. Android

Android, tout comme les Smartphones, a le vent en poupe, et le fait que l'API se base directement sur des fonctionnalités de l'OS, apporte un confort au niveau du déploiement de l'application sur différents équipements. Ainsi la même application peut être distribuée sur des équipements hétérogènes sans que cela n'ait dû être testé au préalable. Ceci assure la pérennité de notre application face aux évolutions des équipements mobiles.

L'interface graphique de base offerte par Android est soignée et possède bon nombre de composants graphiques paramétrables. De plus il existe la possibilité d'y appliquer un thème pour faire évoluer nos graphiques sans modifier beaucoup de code.

De manière plus pédagogique, nous nous sommes laissés séduire par l'attrait de la nouveauté que représente Android, ainsi que par l'enrichissement que notre développement va nous apporter. En outre ceci constitue une occasion à ne pas perdre pour se lancer dans le développement mobile en commençant par Android.

Même si ce n'est pas l'argument le plus convaincant, nous pouvons signaler pour terminer qu'Android à été porté sur ordinateurs portable notamment des netbooks, et de ce fait nous pouvons nous imaginer déployer la même application sur des média mobiles ainsi que sur des médias à priori fixes.

Remarque : Une idée serait par exemple de mettre un netbook tactile pour les personnes âgées, l'équipement étant plus grand, il leur serait plus aisé d'utilisation, et surtout sans la complexité d'utilisation d'un téléphone portable.

Notre choix se porte donc sur un équipement smartphone, avec une plateforme Android.

8.3.Plateforme de développement

Etant donné qu'aucun de nous ne possède de Smartphone avec Android, l'intégralité du code sera testée sur l'émulateur fourni avec le SDK de Google.

Ce petit détail implique quand même que certaines fonctionnalités ne pourront être testées comme par exemple le vibrateur du téléphone, l'appel, l'envoi de SMS, ou l'utilisation de l'IMEI pour identifier notre téléphone. Nous tiendrons donc compte de ces facteurs dans le développement sans pour autant les écarter sous prétexte que nous ne pouvons les tester dans l'immédiat. *(En ultime recours il est toujours possible d'emprunter un Smartphone à un de nos collègues.)*

Figure 21 - Emulateur Android avec interface HTC Magic

Nous avons le choix entre plusieurs méthodes de développement, de manière simplifiée :

- ⇨ Développement sous Eclipse avec les modules de développement et de debug fournis par Google.

- ⇨ Développement sous Netbeans avec un plugin non officiel, sans l'accès au debugger intégré.
- ⇨ Développement en mode éditeur de texte avec le SDK en ligne de commande.

Nous avons pris la peine d'essayer les différents modes, et nous en avons retenu que le développement sous Eclipse n'est de loin pas idéal, mais au vu des choix disponibles il constitue le meilleur choix. C'est donc sous Eclipse que nous avons réalisé l'intégralité des développements mobiles.

8.4.Interface graphique

Etant donné que notre application se destine au grand public à savoir des hommes et des femmes d'âge variés, et sans connaissances informatiques/techniques particulières, notre application se devait d'être simple conviviale et surtout intuitive.

8.4.1. Définition

Nous avons mis en place la règle des 60 secondes, à savoir : quelqu'un qui n'a jamais utilisé notre application se doit de comprendre comment elle marche en moins d'une minute. Ce test réalisé par des personnes de notre entourage, sur base papier, puis sur l'émulateur, nous a permis de réaliser une interface graphique minimaliste, épurée et surtout ultra fonctionnelle.

Nous avons émis plusieurs possibilités, et celle qui a rencontré le plus grand succès est celle qui affiche les médicaments sous forme de liste avec un minimum de détails affichés.

Voici donc l'esquisse de notre interface graphique :

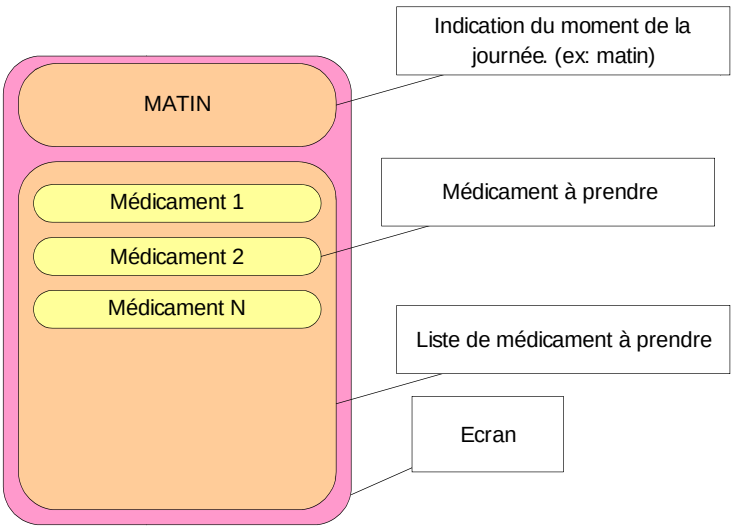

Figure 22 - Esquisse interface graphique

Sur cette base nous avons modélisé chaque élément de l'interface avec des composants du toolkit graphique d'Android.

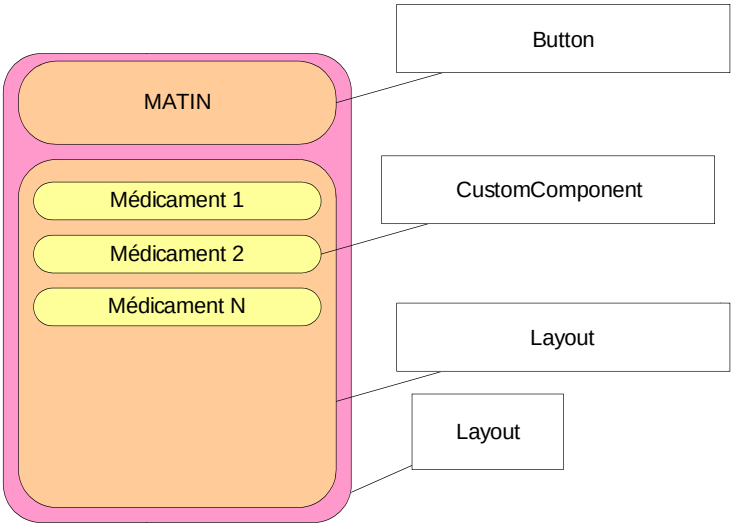

Figure 23 - Modélisation de l'interface graphique avec le toolkit graphique d'Android

Sur la figure ci-dessus nous pouvons constater que la plupart des composants de notre interface existent déjà. Cependant, il nous reste à définir l'élément central de notre interface graphique à savoir le composant médicament.

8.4.2. *Le composant médicament*

Les composants médicaments sont soumis à plusieurs contraintes, ils doivent afficher le nom du médicament, mettre en place un mécanisme pour consulter les informations relatives au médicament, disposer d'un mécanisme pour signaler comme quoi on a effectivement pris le médicament, et tout ceci dans une forme très compacte.

Nous avons décidé de définir un nouveau composant graphique qui serait lui-même composé d'éléments graphiques de base.

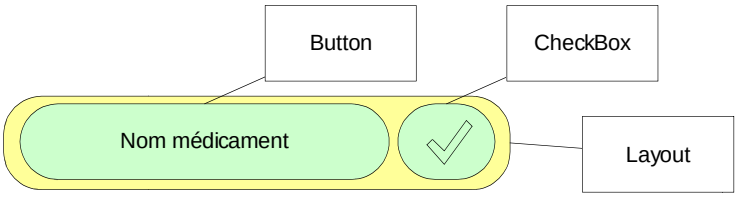

Figure 24 - Composant BoutonMedic

La figure ci-dessus montre notre nouveau composant graphique, il est constitué d'un bouton qui a pour texte le nom du médicament, et il est accompagné d'un checkbox. Sur ces éléments nous avons défini les comportements suivants :

- Le checkbox sert à indiquer la prise du médicament.
- Le bouton sert à afficher les informations relatives au médicament.

Les deux éléments sont regroupés dans un Layout, ceci nous permet de mettre une couleur de fond, une bordure et surtout mieux ajuster l'espacement entre les médicaments grâce à une marge commune.

Signalons ici que l'ensemble des images utilisées dans la partie mobile proviennent d'Internet, et n'ont été retenues que les images libres de droits, ou avec licence libre et/ou avec utilisation non restrictive.

8.4.3. Implémentation

8.4.3.1. Création d'une interface android

Il existe différentes façons d'implémenter une interface graphique sous Android.

- Nous pouvons utiliser l'outil intégré dans le sdk d'éclipse, ceci nous donne le choix entre le mode graphique et le mode xml pour créer la vue. Au final c'est le fichier xml qui est utilisé par l'activity. Cet outil présente quelques instabilités, dans son mode d'édition graphique, et on se retrouve assez souvent à éditer le fichier xml à la main.

- Nous pouvons utiliser un petit programme en ligne (droid draw beta[1]) ce programme a les mêmes fonctionnalités que l'éditeur graphique d'eclipse, à la seule différence qu'il sait gérer les view en absolu, et nous permet de voir les éléments graphiques avant de les mettre dans la vue. Ceci est le mode d'édition adopté par les programmeurs travaillant sous Netbeans, en effet aucun éditeur graphique n'a été ajouté dans le module de développement Netbeans. Cet outil est assez agréable d'utilisation, et produit des fichiers xml en tout point comparables à ceux des outils d'éclipse.

- Nous pouvons créer les vues à la main, en mode de programmation, en créant et en ajoutant à la vue les composants. Ce mode à l'avantage que l'on peut modifier une vue en cours de route, et surtout nous pouvons implémenter une vue dynamique, alors que les deux solutions précédentes sont plutôt orientées vues statiques.

Nous avons dans le cadre de notre application mobile eut l'occasion de tester les trois approches, nous avons initialement utilisé le composant d'éclipse, puis essayé droid draw, et pour finir nous avons dû utiliser l'ajout dynamique pour gérer l'ajout et la suppression de boutons médicaments dans la vue principale.

Nous en retenons que pour une vue orienté « imbrication des composants » l'outil d'éclipse est assez performant car il reste très proche du xml. Pour ce qui est de définir une interface jolie, droid draw est plutôt recommandé et pour tout ce qui est modification dynamique des composants d'une vue, nous sommes obligés de passer par l'ajout dynamique de composants par le code.

8.4.3.2. Icône de l'application

Pour que l'application soit facilement reconnaissable, nous lui avons attribué une icône représentant un médicament. Avec une icône parlante il est beaucoup plus simple de retrouver l'application parmi la multitude d'applications qu'un Smartphone contient ou risque de contenir.

[1] http://www.droiddraw.org/

Figure 25 - Icône de l'application

8.4.3.3. *Bouton Moment de la journée*

Le moment de la journée est composé d'un simple bouton auquel nous avons retiré l'animation du clic, et modifié la couleur d'arrière plan. Nous avons défini une couleur différente pour l'arrière plan du bouton, pour chaque moment de la journée.

Figure 26 - Bouton moment de la journée : midi

8.4.3.4. *Bouton Médicament*

L'implémentation d'un nouveau composant sous Android n'a pas été une tache aisée, en effet Android étant encore jeune, la documentation possède quelques lacunes, et il manque du code de référence pour comprendre comment structurer notre travail. Néanmoins, suite à une recherche empirique nous avons réussi à construire notre propre composant graphique.

De manière résumée voici en quelques points les étapes fondamentales du processus :

- Créer une nouvelle classe qui hérite de la classe LinearLayout (ou de toute autre classe Layout).
- Déclarer les composants de l'objet.
- Déclarer des LayoutParams pour paramétrer l'inclusion d'une vue dans une autre.
- Définir un constructeur qui va instancier les différents éléments graphiques puis les ajouter au layout principal.
- Définir les actions des différents éléments graphiques
- Redéfinir une série de méthodes comme par exemple onDraw, onLayout, onMeasure.

Une des principales difficultés est de bien comprendre la notion de View et de Layout, un layout est une spécialisation d'une vue. De manière générale un layout peut contenir d'autres layouts, ainsi que d'autres views. Une View quand à elle peut aussi contenir plusieurs layouts, et plusieurs views. Néanmoins, la view étant un élémentaire nous

avons plutôt intérêt à utiliser un layout pour nos composants graphique composé d'autres éléments, pour nous affranchir de la gestion d'un canevas.

Notre classe BoutonMedic construit un élément graphique (appelé widget sur android) sur la base d'un médicament. Ceci permet à la classe de récupérer les informations dont elle a besoin et de permettre une meilleure évolution de notre classe. En poussant à l'extrême nous aurions pu définir une interface qui comporterait les méthodes getNom, getInfos, getAcked et ainsi nous aurions pu généraliser notre BoutonMedic en une classe qui pourrait servir à d'autres applications, comme pour une liste de courses par exemple.

Nous avons aussi adopté un visuel différent suivant si le médicament à prendre est à l'heure, ou si c'est un médicament qui n'a pas été pris à temps et est par conséquent marqué comme en retard. Nous avons mis un fond blanc pour les médicaments à l'heure, et un fond rouge pour les médicaments en retard.

Figure 27 - Bouton médicament standard

Figure 28 : Bouton médicament en retard

8.4.3.5. *Affichage des informations complémentaires*

Une fois le BoutonMedic crée il nous fallait définir un moyen d'afficher les informations complémentaires, lors du clic sur un bouton. Initialement nous avions pensé à un système qui irait augmenter la taille du bouton avec une petite animation « growing up ». Cette solution comporte quelques problèmes notamment si un médicament possède une longue liste de contre-indications, le bouton risque d'occuper tout l'écran, et le sur étirement des graphiques devient très inesthétique et inconfortable d'utilisation. Il y a aussi le problème de la taille de la police, en effet la taille de la police d'un bouton est relativement petite, or pour améliorer la lecture il serait préférable d'en augmenter légèrement la taille. Ceci renforce le problème précédent en augmentant d'autant plus la taille finale du bouton.

Après cet essai infructueux, nous nous sommes penchés sur une solution plus « conviviale » et adaptée suivant le type de message : à savoir :

- la possibilité de consulter rapidement une information, « à titre de rappel ».
- la possibilité d'afficher un long message de manière permanente, et dont la lecture serait facilitée.

Nous avons donc ajouté deux manières de consulter les informations, pour répondre aux deux besoins cités précédemment.

- Un message temporaire, qui apparaît puis disparaît automatiquement, avec une police petite, pour la consultation rapide.
- Un message permanent, qui nécessite une action utilisateur pour valider sa fermeture, avec une police plus grasse et plus grande, pour faciliter sa lecture.

Ces deux manières de consulter ont été définies de la façon suivante :

1. Un simple clic sur le nom du médicament
   fait apparaître un message temporaire (appelé « Toast » sur Android), message qui affiche l'ensemble des informations complémentaires du médicament. Ce message a une durée d'affichage d'environ de 2 secondes.

2. Un long clic sur le nom du médicament
   affiche un dialogue muni d'un bouton « fermer » et d'un ascenseur si le texte est trop long.

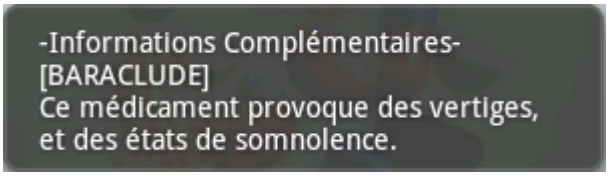

Figure 29 - Informations complémentaires " Toast "

Figure 30 - Informations complémentaires " Dialogue "

Il est à noter qu'une information complémentaire « sous forme de dialogue » contiendra les informations de prise de médicament originelles si celui-ci s'avérait être un médicament en retard. Ainsi l'utilisateur à un moyen de savoir de quand date ce médicament.

A ce stade le message « Toast » n'est pas très joli, les «-» pour mettre en évidence le titre du message ne sont pas très esthétiques, il serait intéressant de redéfinir une classe héritant de Toast pour y inclure la notion de titre. Ce détail n'étant pas crucial, nous l'avons laissé pour une prochaine version ou mise à jour.

Figure 31 - Informations complémentaires " Dialogue " médicament en retard

8.4.3.6. *Validation de la prise d'un médicament*

La validation d'un médicament se fait d'un simple clic sur le checkbox, lors de la validation du médicament, le médicament (BoutonMedic) disparaît de la liste des médicaments à prendre, néanmoins il est conservé tant qu'il n'a pas été communiqué au serveur comme quoi le médicament a été pris.

Comme il est évident que plusieurs médicaments vont être affichés en même temps, nous avons défini des ascenseurs automatiques lorsque le nombre de médicaments ne tient plus sur une seule page.

Lors de la validation d'un médicament un message de confirmation apparaît en même temps que le médicament disparaît de la liste. Ce message a deux principales fonctions :

1. Au début l'apparition du message de confirmation, facilite la compréhension de pourquoi le médicament disparaît, et sert ainsi à rassurer lors d'un clic sur un médicament.

2. Dans un deuxième temps, ce message peut aussi servir si la personne se trompe de médicament, dans l'idée que ses doigts soient assez gros pour qu'elle clique sur le mauvais médicament. Le message lui signalerait le médicament qu'elle doit prendre pour être en accord avec la liste des médicaments.

Les images suivantes montrent a quoi ressemble un message de confirmation, ainsi que la vue principale lorsque tous les médicaments on été soigneusement pris.

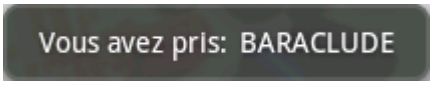

Figure 32 - Confirmation prise médicament " Toast "

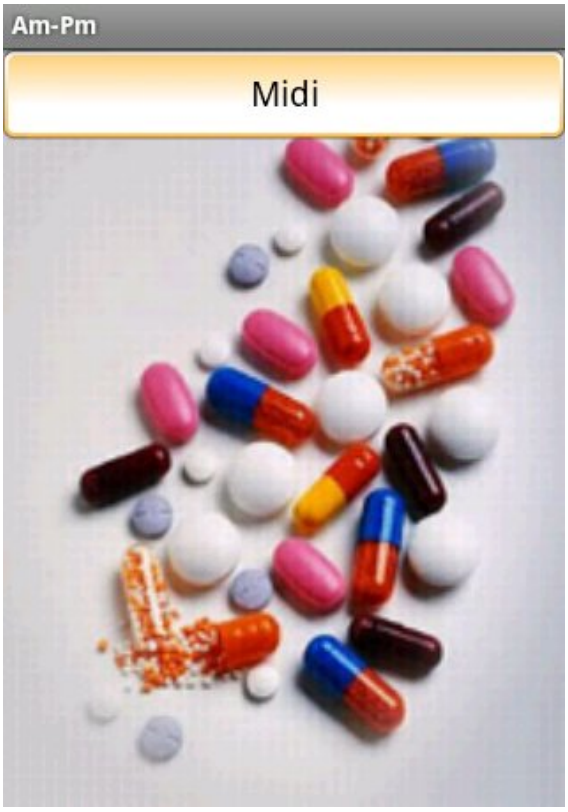

Figure 33 - Interface principale vide

8.4.3.7. *Images d'arrière plan*

En ce qui concerne les images de fond des layout, Android va regarder l'extension de l'image pour déterminer si l'image contient des informations sur l'étirement de l'image.

Par exemple l'image du moment de la journée définie pour le soir porte le nom : bjsoir.9.png.

Et l'image possède des pixels noirs encadrant l'image qui indiquent sur quels cotés on peut étirer l'image. Dans notre exemple, l'image est étirable sur tous ces cotés. L'étirement se fait sans que l'image soit redimensionnée ce qui permet de garder une qualité graphique même si la taille de l'écran varie d'un Smartphone à un autre.

Remarque : Il est aussi à noter que les noms d'images et plus généralement des ressources externes ne peuvent contenir que des minuscules et des lettres, tout autre caractère est très souvent mal supporté par Android.

Figure 34 - Image de fond pour le moment de la journée SOIR

8.4.3.8. *Adaptation de l'interface graphique en fonction de l'orientation*

Nous avons fait en sorte que notre application soit utilisable dans les deux modes d'affichage du téléphone à savoir « Horizontal » et « Vertical ». Ce choix à été motivé par le fait que certains médicaments au nom trop long ou aux informations complémentaires compliquées étaient plus faciles à lire sur un mode « Horizontal ». Voici quelques images illustrant l'application en mode « Horizontal ».

Figure 35 - Vue liste médicament en mode horizontal

Figure 36 - Informations complémentaires en mode horizontal

8.4.3.9. *Résultat final*

Après avoir mis en commun toutes les parties graphiques explicitées précédemment et apporté quelques touches de couleur pour rendre l'application plus agréable à utiliser, et retirer le coté monotone d'une application grisâtre, en tenant compte qu'une application colorée peut dans une certaine mesure remonter le moral des personnes devant prendre

un traitement médical long et ou compliqué. Nous sommes finalement arrivés à un résultat esthétique très proche de notre définition initiale.

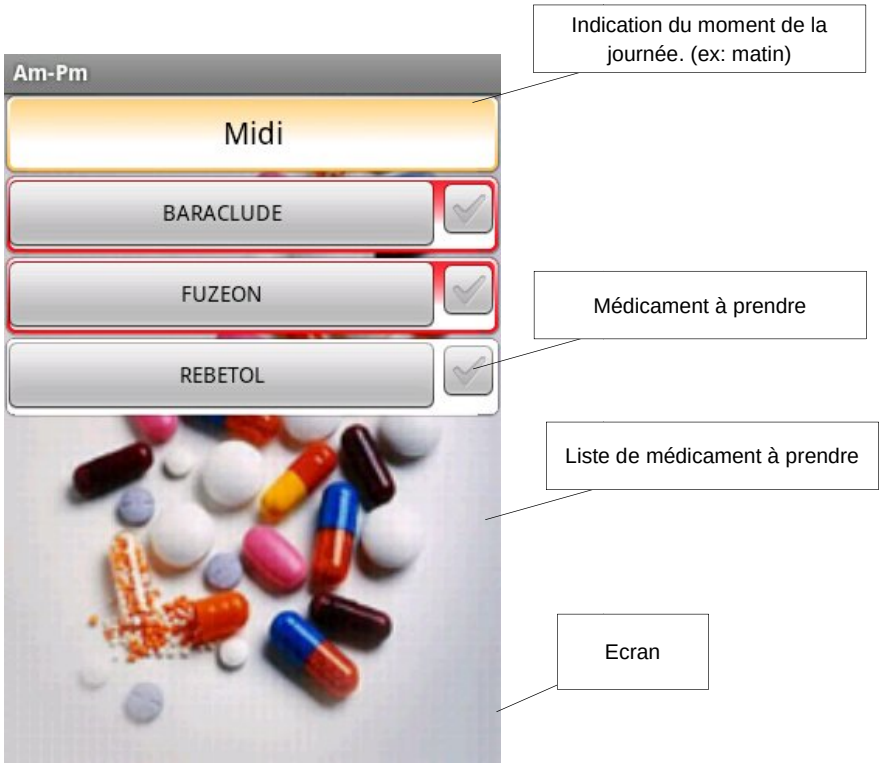

Figure 37 - Interface graphique liste médicaments

8.4.3.10. Système de notifications

Nous avons implémenté un système de notifications pour avertir l'utilisateur dans deux principales situations :

1. Lorsque l'application n'est pas lancée, (mais que le service de gestion de médicaments est actif) une notification averti l'utilisateur qu'il a des médicaments à prendre.
   (remarque il faut que l'application aie été lancée au moins une fois, pour activer le service en arrière plan).
2. Lorsque l'utilisateur quitte l'application, alors qu'il n'a pas pris tous ses médicaments, une notification est aussi lancée pour lui faire remarquer qu'il n'a pas pris tous ces médicaments.

Une notification affiche un message temporaire sur la barre de notifications :

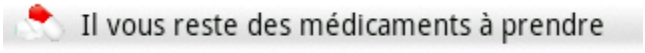

Figure 38 - Message de notification dans la barre de notifications

Puis laisse une notification permanente dans le cache des notifications :

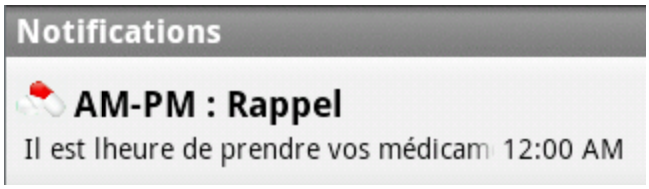

Figure 39 - Notification (1)

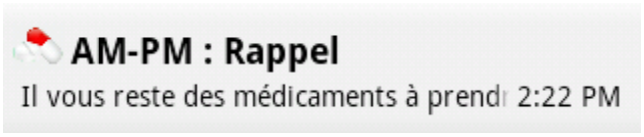

Figure 40 - Notification (2)

8.5.Architecture générale de l'application

Nous avons segmenté notre application en plusieurs états ou vues, qui sont les suivantes :

- Configuration
- Mise à jour
- Liste des médicaments
- A propos

8.5.1. Déclaration centralisée des chaînes de caractères

Android mettant à disposition un mécanisme de chaînes de caractères stockées dans un fichier xml, nous l'avons exploité. Ceci permet de déclarer tous les textes potentiellement affichables dans un fichier. Cette déclaration centralisée apporte deux grands avantages :

- En cas de faute d'orthographe (coquille) sur l'interface graphique il est aisé de retrouver le texte fautif, pour le corriger.
- En cas de traduction de l'application il n'est pas nécessaire de parcourir l'intégralité du code source pour remplacer les chaînes, il suffit de remplacer le fichier xml, et l'application change de nom automatiquement.

8.5.2. Menu

Le menu est l'élément qui permet d'accéder à toutes les vues de l'application ainsi que de quitter. Le menu n'est cependant accessible que sur la vue principale à savoir la liste des médicaments.

Il est déclaré dans le *onCreate* de l'activité principale à savoir le *Main*. Toutes les actions du menu sont définies dans le contrôleur. La taille des boutons du menu est automatiquement définie par Android, suivant le nombre d'éléments que possède le menu.

8.5.3. Moment de la journée

Le moment de la journée est mis à jour à chaque fois que l'application est affichée à savoir lors des appels *onCreate* et *onResume*. Pour bien comprendre le cycle de vie d'une application Android, une explication plus détaillée est présente plus loin dans le rapport. A ce

stade, il nous suffit de savoir que le moment de la journée n'est pas mis à jour si on laisse l'application tourner.

Ce choix est délibéré est appuyé par deux arguments :

1. Une utilisation standard du smartphone implique qu'il va servir à autre chose, ainsi notre application n'est pas la seule à s'afficher sur le téléphone. Il est inutile de tenir à jour un affichage qui n'est pas constamment affiché.
2. Les ressources occupées par un thread de rafraîchissement ne sont pas justifiées pour un changement d'affichage de quatre en quatre heures.

Figure 41 - Menu principal

8.5.4. Configuration

La vue configuration nous permet de spécifier les paramètres de l'application ainsi qu'à récupérer le champ IMEI (sans devoir retirer la batterie), pour que le médecin puisse remplir le formulaire du patient. Le médecin devra renseigner les champs serveur et mot de passe pour que l'application puisse communiquer avec le serveur maître, ainsi qu'un intervalle de mise à jour qui va définir combien de fois par jour l'application ira se mettre à jour.

Supposant que pour une maladie grave, nécessitant un traitement très spécifique et très réactif, on aura beaucoup de mises à jour, cela permet aussi de contrôler la fréquence de soumission des médicaments qui ont déjà été pris.

La configuration étant un élément essentiel de notre application elle est automatiquement lancée la première fois que l'application est lancée. On suppose le scénario suivant :

- Le médecin installe l'application sur le smartphone.
- Le médecin lance l'application, et configure l'application.

- Le médecin récupère l'IMEI, et inscrit le patient sur l'interface web depuis son PC, et lui défini un ou plusieurs traitements constitués d'un ou plusieurs médicaments.
- Le médecin valide la configuration du smartphone.
- Le médecin lance une mise à jour.
- Le médecin rend le smartphone au patient, en lui expliquant le fonctionnement.
- Le patient commence à utiliser l'application

Dans le cas où un disfonctionnement empêcherait l'application d'enregistrer la configuration, un mécanisme empêche d'avoir accès à autre chose tant que la configuration n'est pas remplie et validée. Ainsi on ne peut pas simplement fermer l'application en espérant que le 2e démarrage nous permette de contourner la phase de configuration et d'avoir un accès sur le menu principal par exemple.

Figure 42 - Configuration de l'application

Remarque : L'émulateur n'ayant pas de numéro d'IMEI, la méthode système récupérant l'IMEI retourne une chaîne ne contenant quinze zéros. Comme le champ IMEI est défini par le téléphone nous l'avons affiché dans un input qui est désactivé, ainsi même si une éventuelle modification n'aurait aucune incidence, nous faisons clairement passer le message que ce champ est invariable.

Dans notre scénario le patient n'a aucune idée des paramètres saisis dans la configuration et, à priori, n'y aura jamais accès. Et pour éviter que le patient n'aille toucher la configuration, un mécanisme de mot de passe protège l'accès à la configuration (sauf pour le premier lancement, en accord avec le scénario). Le mot de passe pour l'authentification du patient sur le serveur n'a aucun lien avec le mot de passe qui protège l'accès à la configuration.

Actuellement ce mot de passe est codé en dur, et a pour valeur « 1337 ». Dans une solution commerciale, il serait intéressant de protéger la configuration avec un mot de passe généré à partir de l'IMEI du téléphone ainsi que de la date de mise en service. Pour éviter que quelqu'un découvre le mot de passe, le publie sur Internet et que les patients aillent modifier la configuration, au risque de ne plus pouvoir utiliser leur application. Suivant la gravité de la maladie, cette maladresse pourrait coûter cher, d'où l'intérêt d'une bonne protection.

Figure 43 - Dialogue d'authentification

8.5.5. Mise à jour

La Mise à jour est la partie de l'application qui va initier une connexion avec le serveur, puis va synchroniser la base de médicaments locale avec la base de médicaments du serveur.

De manière générale une connexion ressemble à ceci :

- Initier une connexion
- S'identifier grâce à l'IMEI du téléphone
- S'authentifier avec le mot de passe
- Recevoir la liste des médicaments à prendre
- Envoyer la liste des médicaments pris
- Fermer la connexion

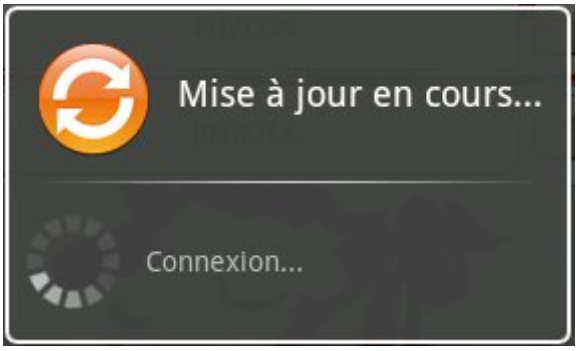

Figure 44 - Dialogue de mise à jour

Nous avons décidé de nous servir de l'IMEI comme identifiant, étant donné qu'il est unique pour chaque téléphone il en a fait un candidat parfait. Nous aurions aussi pu utiliser le numéro IMSI, qui lui est dépendant du fournisseur de réseau.

Le choix de l'IMEI s'est fait en se basant sur le fait que le patient était libre de changer d'opérateur, sans pour autant devoir faire une quelconque démarche auprès du médecin. Nous avons arbitrairement préféré la liberté de changer d'opérateur au lieu de la liberté de changer de smartphone, car cela nous paraissait plus important, mais c'est une appréciation purement subjective. Ceci en gardant à l'esprit que si le patient change de smartphone, un simple rendez-vous chez le médecin suffit à tout rétablir dans l'ordre.

La mise à jour peut se faire de différentes manières :

- Manuellement depuis le menu principal.
- Automatiquement au lancement de l'application si la dernière mise à jour est périmée.
- Automatiquement suivant l'intervalle défini dans la configuration grâce au service de background.

Il faut savoir que les applications sous Android sont mises en pause lors de la pression sur le bouton *retour*, ou sur le bouton *home*. Ceci nous oblige, soit à quitter l'application dans l'événement « *onPause* », soit à définir un bouton d'un menu pour le faire explicitement. Nous avons retenu la deuxième possibilité.

Il faut aussi dire que l'appareil mobile demande une mise à jour, mais c'est le serveur qui décide combien de médicaments sont envoyés. Avec ce système, c'est le serveur qui garde le contrôle sur les mises à jours, ainsi le patient pourrait faire une demande par téléphone à son médecin quelques jours avant de partir en vacances, et l'application irait faire une mise à jour pour une durée de x jours, où x correspond au nombre de jours pendant lesquels le patient ne peut avoir un accès Internet sur son appareil mobile par exemple.

Par manque de temps, il ne nous a pas été possible d'intégrer la gestion du réseau dans notre application, et par conséquent nous n'avons pu que « simuler » une mise à jour, mise à jour codée en dur. Ce point est signalé plus loin dans le point « Reste à faire »

8.5.6. Liste des médicaments

La liste des médicaments est le point central de notre application, c'est la vue affichée par défaut depuis laquelle on a accès au menu de l'application, qui lui nous permet d'accéder aux autres vues.

Figure 45 - Liste des médicaments

Chaque médicament à prendre est modélisé par un composant graphique (BoutonMedic) composé d'un bouton pour obtenir des informations sur le médicament et un checkbox pour valider la prise du médicament.

Cette vue est chapeautée par un bouton dont la couleur du fond change en fonction du moment de la journée, qui affiche le moment de la journée, à savoir :

- NUIT
- AUBE
- MATIN
- MIDI
- APRES-MIDI
- SOIR

8.5.7. A propos

Le dialogue à propos comme son nom l'indique affiche un message d'à propos avec le but de l'application, ses auteurs, la date, un copyright, et une mention sur le cadre du projet, à savoir le cours IFC2.

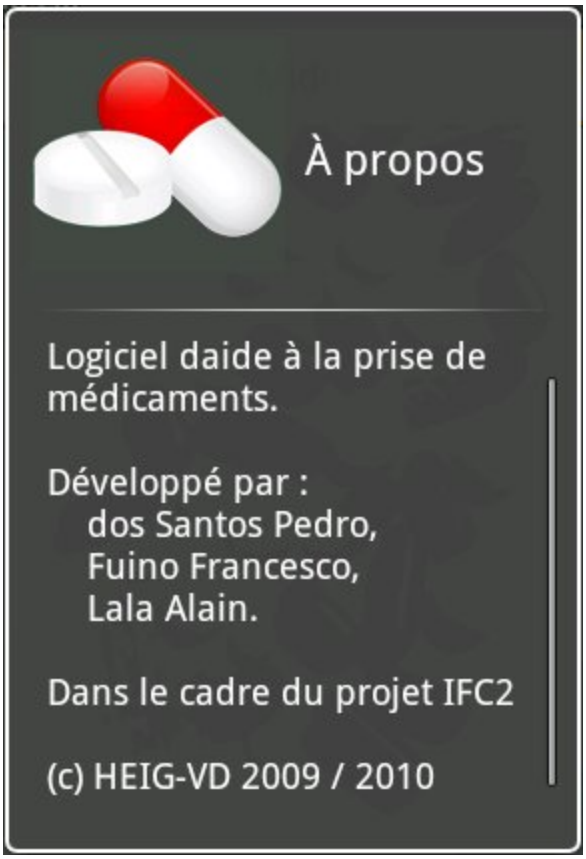

Figure 46 - Dialogue à propos

8.6.Fonctionnement de l'application

Afin de montrer les différentes capacités de l'utilisation, nous avons réalisé un diagramme de fonctionnement. Celui-ci schématise les différentes possibilités d'une application ainsi que leur effet en fonction de tests conditionnels.

Tel que le montre le schéma nous constations que nous avons deux manières d'entrer dans l'application.

- C'est soit un lancement, ceci implique en règle générale que l'application n'a pas été lancée depuis le démarrage du téléphone, ou qu'elle a été quittée.
- Soit une reprise, par conséquent l'application était en pause, en arrière plan, et nous l'avons réveillée.

Ce diagramme nous montre aussi que nos vues ne sont pas rafraîchies automatiquement, en effet pour des raisons d'efficacité nous avons rafraîchi les vues seulement si cela était nécessaire.

Ce schéma va par la suite servir de canevas pour les tests de l'application, ainsi pour chaque étape nous pouvons déterminer si l'action s'est bien déroulée.

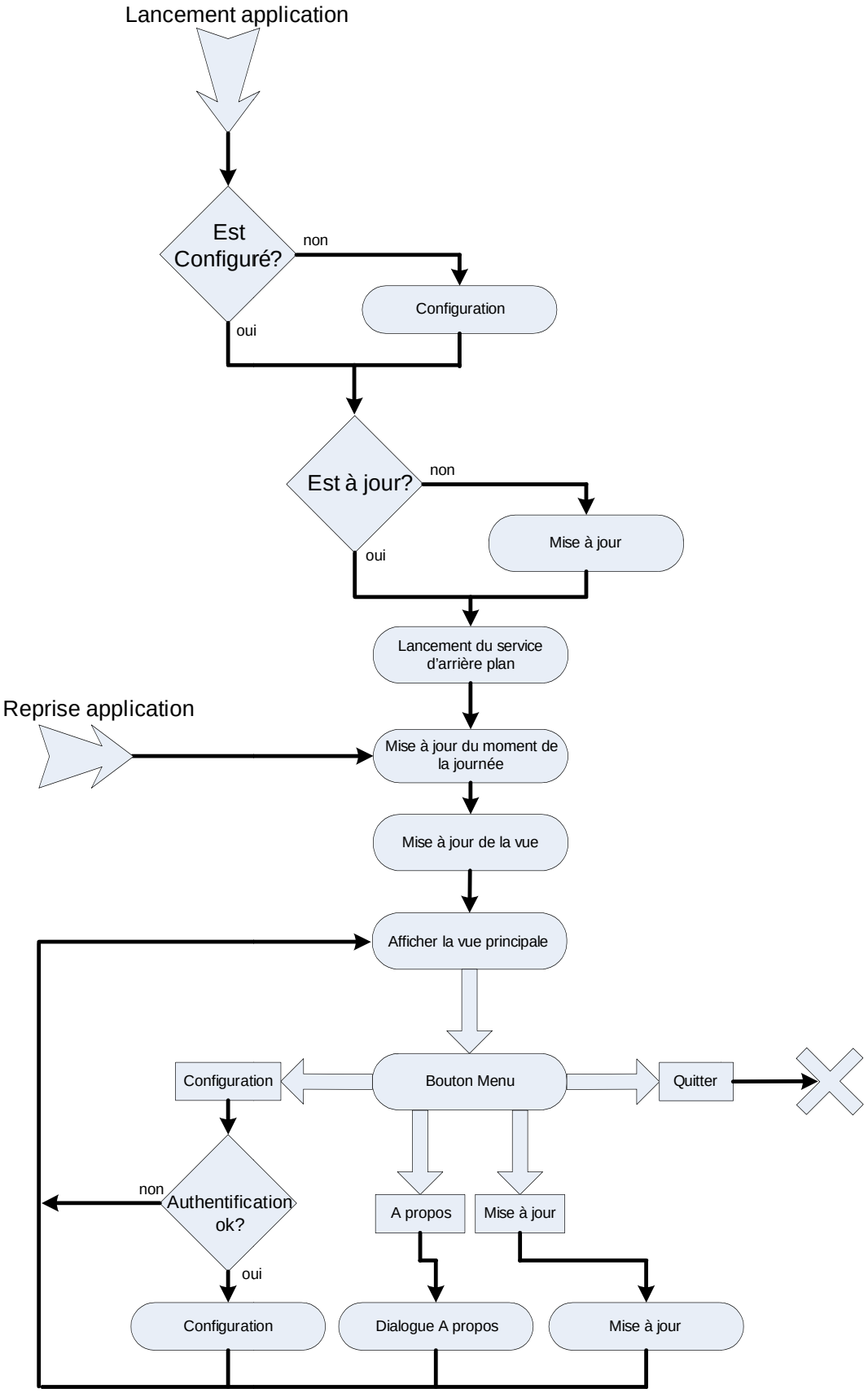

Figure 47 - Diagramme de fonctionnement

8.7.Implémentation de l'application

Une application mobile possède un cycle de vie, dans le cas d'Android il y a sept états :

- onCreate()
  cette méthode est appelée une fois, lorsque l'utilisateur clique le l'icône de l'application pour la première fois, ou après un redémarrage du téléphone, lors du lancement de l'application ou alors si l'application a été tuée par un composant d'Android pour gagner de la place mémoire. C'est en quelque sorte le constructeur de l'application.

- onDestroy()
  cette méthode est appelée juste avant que l'application ne soit détruite, après l'application est tuée, et au prochain lancement un onCreate est lancé.

- onStart()
  onStart est lancé juste après onCreate et toutes les fois que l'application est mise en sommeil pour relancée.

- onStop()
  onStrop est appelé juste avant onDestroy et surtout à chaque fois que l'application va passer en mode sommeil.

- onResume()
  est appelé juste après onStart() mais surtout à chaque fois qu'une vue de l'application occupe l'écran.

- onPause()
  est le contraire de onResume, elle est appelée avant chaque onStop, et surtout à chaque fois ou la vue de l'application est remplacée par une autre vue d'un autre application.

- onRestart()
  est lancé juste avant onStart(), à chaque fois qu'une vue de l'application occupe l'écran.

La documentation Android nous fourni une image assez explicite sur les différents cycles de l'application :

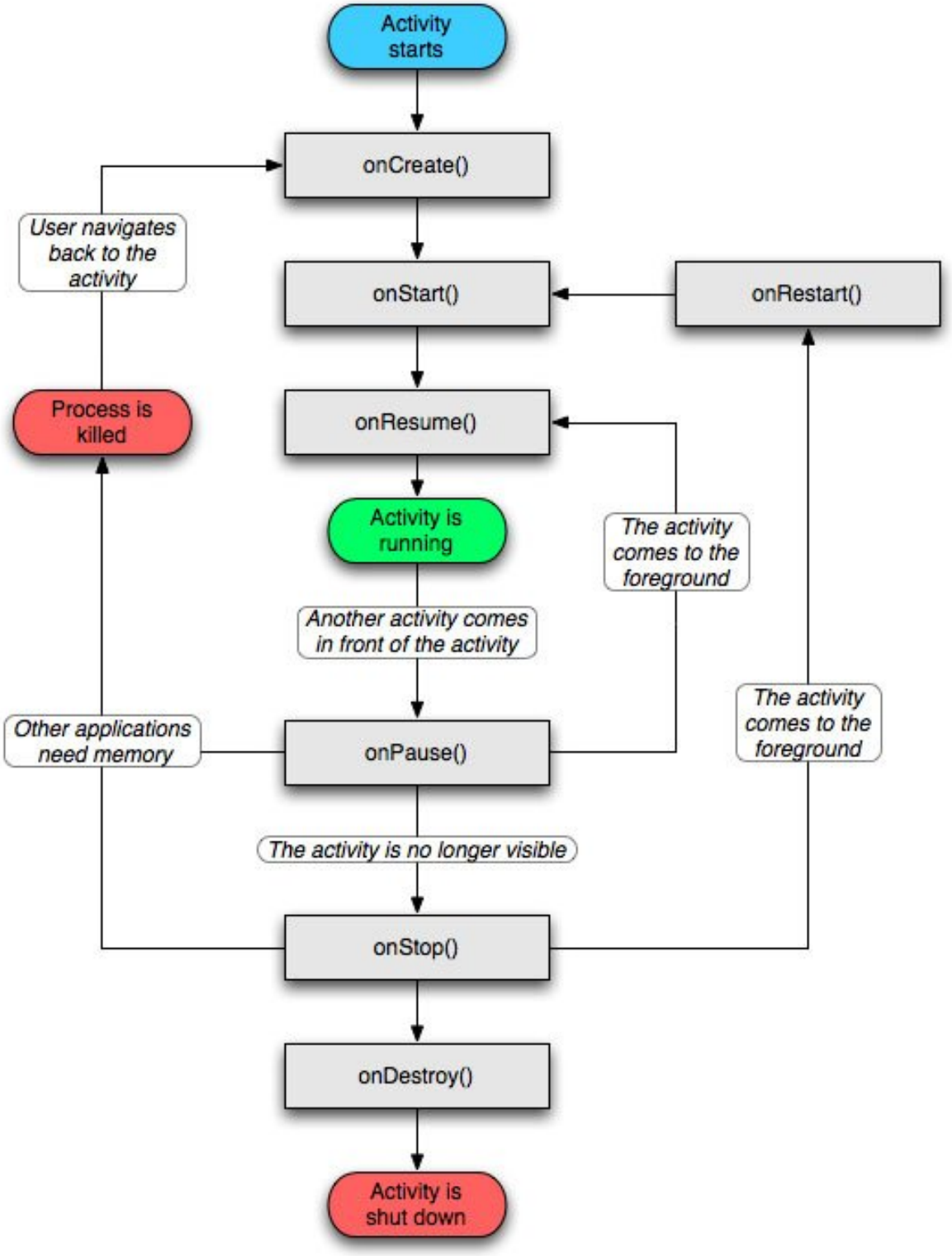

Figure 48 - Cycles de vie d'une application android[2]

Dans la solution livrée nous avons défini le *onCreate*, qui va initialiser l'ensemble de l'application. Comme nous le verrons plus tard la seule chose qui reste en mémoire volatile étant les

[2] http://developer.android.com

différentes instances d'objets, et un timestamp de la dernière mise à jour, nous n'avons pas jugé utile de redéfinir des comportements spécifiques pour les différents cycles de vie de l'application.

Dans l'espoir de développer une application qui puisse évoluer rapidement nous avons adopté une philosophie MVC (modèle – vue – contrôleur). Nous avons séparé notre application en deux parties distinctes la configuration et les médicaments. Chaque partie ayant son contrôleur, sa vue et son modèle. (La configuration a une vue jointe au contrôleur).

Grâce à ce modèle il nous est facile de faire évoluer la vue ou le contrôleur sans que cela n'ait d'impact majeur sur l'application, justement à cause de la séparation. Néanmoins, nous avons été contraints d'adapter un peu la philosophie d'Android, à cause de certaines limitations. Par exemple, Android ne permet pas à un thread de modifier une vue, si le thread n'est pas responsable de la dite vue. Ce type de limitation nous a obligé à adapter notre solution initiale vers une solution plus fonctionnelle et moins théorique.

Le fait que cette application soit notre entrée en matière dans le développement Android, elle nous a fait faire des erreurs de jeunesse, en effet nous nous sommes basés sur un développement d'application standard mais ceci était sans savoir que le développement Android est plus aisé si on respecte une certaine philosophie imposée par Google.

Cette philosophie est en réalité une méthodologie couplée à quelques recettes de cuisine, qui une fois assimilée ne parait pas compliquée, mais qui reste néanmoins assez complexe à acquérir, ce qui explique notre orientation vers un développement standard.

De manière très simplifiée il faudrait déclarer une activity qui regroupe le contrôleur et la vue, le modèle lui peut être facilement séparé, mais peut aussi être inclus. De manière générale, une vue se déclare sous forme de xml, dans un fichier spécial et l'activity va charger sa vue depuis le xml. En gardant à l'esprit qu'une vue peut être un affichage sur l'ensemble de l'écran, une boite de dialogue, etc. On constate que chaque élément est assez indépendant des autres, mais assez dépendant de fichiers xml décrivant la vue, ou d'autres ressources.

Par-dessus le concept d'activity vient le concept d'Intent, qui lui se trouve être l'élément déclencheur d'une activity.

Signalons ici que si aucun dispositif particulier n'est pris, la vue n'est pas rafraîchie automatiquement, et peut même être bloquée si un traitement de fond occupe le processeur, ce qui peut poser problèmes lorsque l'on veut calculer quelque chose et afficher des résultats partiels par exemple. Tout ceci pour dire que même si le code est écrit en java, il faut avoir assimilé la philosophie de programmation Android avant de se lancer dans un développement orienté production.

Au moment où la philosophie Android commençait à être acquise, il était trop tard pour refondre l'intégralité de l'application, ceci explique que notre solution est fonctionnelle certes mais ne respecte pas (en règle générale) les directives de la philisophie Android.

Il est important de signaler qu'Android possède un système de droits pour une application donnée. Ils sont définis dans le fichier Manifest.xml. Dans ce fichier sont définies quelles applications peuvent lancer tel ou tel intent, sont aussi déclarés la portée d'utilisation des applications à savoir, vont-elles lire les données internes du téléphone, peut-elle faire vibrer le téléphone ?

En sachant que si les droits ne sont pas définis lors de l'exécution, une exception est lancée empêchant ainsi l'exécution d'un service non autorisé.

Grâce à ce fichier, il est très simple de connaître le scope d'une application par rapport au matériel, et ainsi de pouvoir déceler des applications de type malware.

Dans notre projet nous avons dû ajouter une série de droits pour pouvoir lancer des activity, ainsi qu'un service d'arrière-plan, ainsi que des permissions relatives au matériel :

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

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

En ce qui concerne l'implémentation des classes, nous avons décidé de définir des éléments unitaires, qui eux-mêmes définissent en interne ce dont ils ont besoin, ainsi grâce au mécanisme de classes internes nous avons pu déclarer à l'intérieur des classes, des classes qui appartiennent entièrement à la classe mère.

Nous avons préféré la solution (nested) plutôt que l'agrégation pour trois principales raisons :

1. Les méthodes des classes internes peuvent directement accéder aux attributs de la classe conteneur. Ceci nous permet de déclarer un certain nombre de méthodes privées auxquelles les classes internes ont accès, mais pas les autres.

2. Les classes internes peuvent être cachées aux autres classes du même paquetage, ceci nous garantit une meilleure séparation du code tout en préservant sa sécurité, notamment sur des données sensibles. En d'autres termes ceci nous permet de cacher la complexité interne.

3. Les classes internes nous permettent réaliser une séparation franche des différents composants entre ceux qui sont réutilisables et ceux qui sont étroitement liés.

Remarque : nous avons utilisé une convention de nommage pour nos fichiers, ainsi ce qui commence par M_ est un modèle, ce qui commence par C_ est un contrôleur, ce qui commence par S_ est un service, ce qui commence par V_ est une vue et ce qui ne commence pas par une lettre distinctive est soit une classe utilitaire, soit une classe standard soit une classe d'activité comme « Main » par exemple.

Nous avons réalisé une classe service qui en théorie averti l'utilisateur par une notification et une vibration des que des médicaments doivent êtres pris. Et s'occupe de faire des mises à jour automatiques sans en avertir l'utilisateur, avec des intervalles de mise à jour dépendants de la valeur configurée dans la configuration.

La partie réseau étant encore en développement, et par conséquent pas encore fonctionnelle le service à l'heure actuelle ne fait qu'avertir l'utilisateur qu'il doit prendre des médicaments.

8.7.1. Structure de données

Le médicament étant le cœur de notre application, il est évident que la structure médicament soit notre structure principale. Un médicament est constitué des champs suivants :

- Nom : Nom du médicament.
- Informations : Informations relatives au médicament, des remarques sur la posologie, etc.
- Id : Entier identifiant de manière unique chaque médicament.
- IdDef : Entier identifiant de manière unique les informations complémentaires au médicament.
- Date : Date à laquelle un médicament doit être pris.
- Heure : Moment de la journée où le médicament doit être pris
- EnRetard : Booléen qui indique si un médicament a été pris à son heure juste, ou s'il est en retard.
- Ack : Booléen qui indique si la personne a déjà validé la prise du médicament.

Afin de regrouper toutes ces informations sous un même élément, nous avons créé une classe « Medicament ». En sachant que notre application manipule des « listes » de médicament nous avons eu recours à la classe « Vector » pour modéliser notre liste. La liste ne nécessitant pas de méthodes particulières, ni de gestion interne complexe, nous l'avons utilisé tel quel sans redéfinir une classe « ListeMedicaments » héritant de « Vector » par exemple.

La classe Date de Java (java.util) ne satisfaisant pas nos besoins, nous avons décidé de créer notre propre classe Date. Notre classe contient les éléments jour, mois et année, ainsi que quelques méthodes utiles comme la récupération d'un timeStamp de la date actuelle.

Initialement nous avions déclaré les moments de la journée sous forme d'un type énuméré. Cependant, sa gestion était lourde et peu pratique, nous avons donc décidé d'en faire une classe à part entière, et de lui ajouter une certaine quantité de méthodes utiles pour faciliter leur gestion. Nous pouvons signaler entre autre une redéfinition de toString() qui nous permet d'exporter le moment de la journée sous forme de chaîne de caractères, toInt() nous donne une valeur numérique en sachant que les moment de la journée sont ordonnés comme suit : nuit < aube < matin < midi < après-midi < soir. Nous avons aussi fourni des méthodes permettant de réaliser des conversions vers un type « MomentsJournee »

8.7.2. Implémentation des données

Andoid met à disposition plusieurs méthodes pour stocker des données de manière persistante.

- Stockage dans un fichier texte.
- Stockage dans une base de données SQLite.
- Stockage dans les préférences utilisateur.

- Stockage sur le réseau.

Le stockage sur le réseau a été d'office éliminé étant donné que notre application est sensée fonctionner partout, même si il n'y a pas d'accès wifi ou 3G.

Le stockage dans les préférences utilisateur, ainsi que dans un fichier texte, sont peu adaptés pour le stockage de grands volumes de données structurées.

Ainsi, en accord avec les capacités de la plateforme nous avons décidé de stocker nos informations sur une base de données SQLite. D'une part, une base de données apporte un confort d'utilisation et surtout rapidité d'accès à l'information comparé a une solution basée sur fichiers, d'autre part elle permet une scalabilité beaucoup plus importante qu'une gestion basée sur des fichiers.

La seule base qui pourrait être facilement remplacée par un fichier, est la base de données pour la configuration, en effet dans ce cas précis ambivalence peut être discutée. La seule raison qui nous a amené a préféré une base de données est le fait de pouvoir conserver un historique des modifications.

SQLite comme son nom l'indique est une base de données légère qui ne possède ni la puissance des grands SGBD, ni la souplesse de modification de ces derniers. De plus SQLite souffre de certaines faiblesses dont voici les principales :

- Aucune gestion de comptes utilisateur.
  Dans notre cas nous n'en avons pas besoin étant donné que seule l'application mobile a accès à la base de données, en mode root.
- Les possibilités de modifier la structure d'une table sont limitées au renomage d'une table et à l'ajout de colonnes. Aucune gestion en ce qui concerne la modification ou la suppression.
  Dans notre solution, les structures de la base sont statiques et ne risquent pas de changer pendant l'utilisation. Par conséquent cette limitation ne pose aucun problème.
- SQLite n'est pas conçu pour gérer plusieurs accès concurrents, ceci encore une fois n'est pas une limitation dans notre application car seule l'application va lire ou écrire dans la table, et tant bien même si plusieurs threads auraient des accès concurrents, SQLite saurait le gérer.

De manière générale nous saluons le fait qu'Android embarque SQLite, car il est un atout important pour des applications manipulant des données complexes ou en grand nombre.

L'implémentation d'une base de données SQLite sous Android est possible de différentes façons, tout dépend si on veut que nos données soient consultables depuis une autre application. Dans ce cas, il faudra créer une classe héritant de ContentManager. Comme nous avons simplement besoin des données en interne nous n'avons pas retenu cette option.

Nous nous sommes servis de « Helpers » qui nous permettent d'interfacer la base de données SQLite de manière relativement simple, nous avons ainsi défini une classe helper pour chaque table.

Les classes helper contiennent du code SQL pour la création, la suppression et la mise à jour d'une table. Une fois les helpers définis nous pouvons instancier un objet « SQLiteDatabase » et y faire des requêtes en respectant les conventions de l'API.

Il est à noter que nous avons ajouté quelques champs dans les helpers pour récupérer le nom de la table par exemple. Il faut aussi signaler que l'insertion se fait à travers de « ContentValues », et que le résultat d'une requête retourne un « Cursor », ceci n'est pas explicité plus en détail car cette méthode est très proche de la manière de faire de JDBC.

La classe modèle « M_Medicament » nous donne une abstraction totale sur comment les données sont implémentées en dessous, en nous permettant de récupérer les médicaments sous forme d'objets médicament, sans se soucier de comment sont implémentés les médicaments en interne. Ceci nous donne la possibilité de modifier le média de stockage de nos médicaments sans modifier l'intégralité de l'application.

Remarque : initialement dans la fonction de récupération des médicaments nous avions pensé récupérer tous les médicaments jusqu'à maintenant; ceci implique un opérateur de comparaison « plus petit ou égal ». Or nous n'avons pas réussi a faire passer cet opérateur de dates dans la requête. Le résultat est toujours nul. Nous avons laissé la méthode intacte, à la seule différence que nous récupérons tous les médicaments, puis sur chaque médicament récupéré nous décidons s'il fait partie de l'ensemble des médicaments à prendre ou non.

Ce bug nous oblige un traitement plus long, de par le fait que la requête de la base de données met plus de temps à s'exécuter, mais surtout le temps de traitement après coup.

Cette solution n'étant pas définitive, nous l'avons laissée, et mise de coté en attendant de trouver la solution de la requête SQL, ou de trouver une solution plus optimisée, pour nous concentrer sur d'autres parties de l'application.

8.7.3. Stockage des données

Un médicament est constitué de différentes informations, celles-ci ont été divisées en deux parties distinctes pour une question d'optimisation :

Le médicament, et sa définition, de la même manière qu'un médicament est livré avec une notice d'emballage commune à tous les médicaments d'une même boîte, la définition d'un médicament contient les informations relatives à un ensemble de médicaments et ceci sans que chaque médicament doive transporter avec lui des données redondantes.

Ainsi, un médicament est stocké en interne dans deux structures différentes à savoir une table *Medicaments* et une table *Definitions*, toute deux appartenant à la base *B_Medicaments*.

La table *Medicaments* contient donc les champs suivants :

- Id : identifiant du médicament.
- Heure : moment de la journée où le médicament doit être pris.
- Date : date de prise du médicament.

- idDef : identifiant de la définition du médicament.

La table *Definitions* contient les champs suivants :

- Id : identifiant de la définition.
- Nom : Nom du médicament.
- Infos : Informations complémentaires.

Cette séparation est très utile notamment pour les échanges réseau, en effet si la base de données contient déjà les définitions d'un médicament, il est inutile de les re-télécharger. Autre avantage une même définition peut être commune à plusieurs médicaments. Cette séparation nous évite aussi de gérer des données plus conséquentes sur un appareil mobile, ce qui pourrait nuire à la réactivité de l'application, ou à la consommation des batteries.

Un objet « *Medicament* » se sert des deux tables pour constituer un médicament avec tous les champs initialisés. A l'exception des champs *enRetard* et *acked* qui eux sont non permanents.

En effet le champ *enRetard* peut être facilement déterminé quand au champ *acked* n'est pas sensé être enregistré étant donné qu'un médicament soumis au serveur est ensuite supprimé de la base.

Lors de la mise à jour, on détermine si les définitions existent et dans le cas contraire, on demande au serveur de nous les transmettre.

De la même manière la date de la dernière mise à jour n'est pas stockée de manière permanente. Ceci aura pour conséquence que l'utilisateur va se mettre à jour à chaque fois qu'il ira démarrer l'application, c'est-à-dire à chaque fois qu'il va quitter l'application via le menu, ou s'il redémarre son Smartphone, puis lance l'application. En supposant que l'utilisateur aura un comportement normal, le nombre de mises à jour n'est pas ou peu affecté par ce choix. Dans le cas où l'utilisateur va systématiquement quitter l'application à chaque utilisation, une mise à jour sera faite à chaque démarrage ou par l'intermédiaire du service d'arrière-plan. Il serait, néanmoins, envisageable dans une solution future de stocker cette valeur dans les préférences utilisateur. Ainsi, la valeur serait permanente et le nombre de mise à jour serait diminué pour un utilisateur qui irait systématiquement quitter l'application.

La configuration est une autre donnée essentielle qu'il faut stocker de manière permanente.

Nous avons décidé de créer une structure « *ConfigurationAMPM* » pour modéliser une configuration. Elle est ensuite stockée dans une base de données différente des médicaments à savoir *B_Configuration*.

Nous avons aussi fait en sorte que l'on puisse enregistrer plusieurs configurations les unes par-dessus les autres, et la configuration effective serait la dernière. Ceci nous permet par exemple de revenir à une version antérieure, ou simplement de mettre à disposition plusieurs serveurs pour les sources de médicaments.

9. Ce qu'il reste à faire

- La gestion du réseau, ainsi que les méthodes de mise à jour.
- Améliorer la gestion d'erreur, avec des résolutions d'erreurs par scénario pour que l'application puisse se récupérer automatiquement en cas de problème.
- Corriger la méthode de récupération des médicaments.

10. Améliorations possibles

10.1. Partie Web

- Il serait aussi nécessaire de sécuriser la procédure de connexion en instaurant un mécanisme rendant impossible les replay attacks par exemple en introduisant un nounce.
- La vérification des champs de formulaire se fait uniquement sur le fait que le champ soit vide ou non et pour les nombre que ces derniers soient des entiers. Il serait bon de fixer des contraintes plus précises.
- La partie visuelle du site est aussi totalement à revoir. Le graphisme est basique mais fonctionnel.
- En ce qui concerne la classe de connexion à la base de données elle n'est fonctionnelle qu'avec une base MySQL. Cependant la classe peut être complétée pour la rendre compatible avec d'autres moteurs de base de données comme Oracle par exemple.

10.2. Partie Mobile

- Meilleure gestion du cycle de vie de l'application.
- Mettre sur les informations des médicaments en retard un bouton permettant de joindre le médecin traitant ou une hotline.
  (Ajouter un champ dans la configuration, pour que le médecin puisse y mettre le numéro de téléphone).
- Stocker la date de dernière mise à jour dans les préférences utilisateur, pour que l'application se mette à jour moins souvent.
- Lancer le service d'arrière plan au démarrage d'Android, sans être obligé de démarrer manuellement l'application pour activer le service.

11. Conclusion

L'application que nous avons développée répond à un besoin grandissant d'ordonner la liste des médicaments, qui se fait de plus en plus longue, dans les prescriptions. De plus il permet au médecin d'avoir un suivi du patient à distance, sans avoir à arranger un rendez-vous. La prise de contact avec des personnes du milieu médical, ainsi que l'avis d'utilisateurs potentiels nous ont permis de façonner une interface graphique, autant pour le mobile que pour l'interface web, au plus proche de ce que les futurs utilisateurs attendent d'un tel produit.

Il ne s'agit surement pas de la solution parfaite, et de nombreuses améliorations sont encore possibles. Cependant nous pensons qu'avec le système que nous avons mis en place, nous pouvons réellement aider des personnes ayant de grosses prescriptions, avec de nombreux médicaments à prendre à des heures précises, à mieux vivre leur maladie en se préoccupant moins de la prise de leurs médicaments.

Au-delà de cet aspect, ce projet nous a également permis de réaliser un travail d'ingénieur dans le domaine médical et nous a donc obligé à nous conformer aux exigences du milieu afin de répondre aux attentes.

12. Annexes

Diagramme de classe de la partie mobile

- *Diagramme_de_classes_Partie_Mobile.pdf*

Schéma relationnel de la base de données

- *Schema_Base_De_Donnees_Relationnelle.png*

12.1. Manuels

Manuel d'utilisation de la partie mobile

- *Manuel_d_utilisation_Partie_Mobile.pdf*

Manuel d'utilisation de la partie web

- *Manuel_d_utilisation_Partie_Web.pdf*

12.2. Feuilles de tests

Feuille de tests partie mobile

- *Feuille_de_Tests_Partie_Mobile.pdf*

Feuille de tests partie web

- *Feuille_de_Tests_Partie_Web.pdf*

13. Références

1. DIMARZIO, Jérôme, Android : A Programmer's Guide, Osborne/McGraw-Hill, 2008.

2. GARIN, Floren, Android : Développer des applications mobiles pour Googles phones, Dunod, 2009.

3. GRAMLICH, Nicolas, Andbook ! Android Programming, pdf : source *http://andbook.anddev.org/*

4. Icon Finder [En-Ligne]
   - http://www.iconfinder.net

5. Medi-Memo [En-construction]
   - http://www.medi-memo.com/

6. Online JavaScript Compression Tool [En-Ligne]
   - http://jscompress.com/

7. JQuery [En-Ligne]
   - http://jquery.com/
   - http://jqueryui.com/

8. PHP [En-Ligne]
   - http://php.net/index.php

9. PHP Manual [En-Ligne]
   - http://www.php.net/manual/en/

10. Standards – W3C [En-Ligne]
    - http://www.w3.org/standards/

11. OpenWeb pour les standards du web [En-Ligne]
    - http://openweb.eu.org/articles/changer_pour_utf8

12. MySQL reference manual [En-Ligne]
    - http://dev.mysql.com/doc/refman/5.5/en/

13. Aide Santé, les pilules bien être. [En-Ligne]
    - http://www.aidesante.fr

14. Distrimed : Le matériel médical sur le net [En-Ligne]
    - http://www.distrimed.com/

15. Andoid Developers [En-Ligne]
    - http://developer.android.com/
    - http://android-developers.blogspot.com/

16. Droid Draw Beta
    - http://www.droiddraw.org/

17. FrAndroid : La communauté des développeurs francophones autour d'Androide
    - http://dev.frandroid.com/

18. Android developement : Applications made for Android
    - http://www.androiddevelopment.org/

19. AndroidiBlogger
    - http://androidblogger.blogspot.com

13. Table des illustrations