République Tunisienne
Ministère de l'Enseignement Supérieur et
de la recherche scientifique

Université de Gabès
École Nationale d'Ingénieurs de Gabès
Département de Génie des
Communications et des Réseaux

ENIG

Projet de Fin d'Études
N° d'ordre: 2013

Mémoire de Projet de Fin d'Études

Développement d'une application Android pour médecins.

Présenté à

L'École Nationale d'Ingénieurs de Gabès

En vue de l'obtention du

Diplôme National d'Ingénieur en Communications et Réseaux

Réalisé par : Riadh HABBACHI

Encadré par : Mr. Ikbel AZAEIZ
Mr. Aymen ELJ

Soutenu le --/--/2013, devant la commission d'examen:

Mr.		*Président*
Mr.		*Membre*
Mr.	**Ikbel AZAEIZ**	*Encadrant*
Mr.	**Aymen ELJ**	*Invité*

AU : 2012/2013

Remerciements

Je tiens par le présent rapport à exprimer mes vifs remerciements à monsieur **Ikbel Azaeiz** mon encadreur de l'ENIG pour son encadrement exemplaire. Sa disponibilité indéfectible m'a été d'un grand soutien. Je le remercie pour son support, ses précieux conseils et ses judicieuses critiques.

Á Messieurs **Mohamed Anis Kallel** PDG de TUNAV et **Aymen Elj** Directeur Développement pour la confiance dont ils ont fait preuve à mon égard en me permettant la mise en œuvre de ce projet. Je tiens à leur exprimer ma profonde reconnaissance pour le soutien qu'ils m'ont accordé et les moyens de travail qu'ils m'ont fournis tout au long de la réalisation du projet.

Un remerciement spécial pour monsieur **Ali Frihida** de l'ENIT pour ses conseils précieux et m'avoir permis cette chance de travailler sur ce projet.

Table des matières

Table des figures

Liste des tableaux

Liste d'Abréviations

ADB Android Debug Bridge. 7, 52, 53
ADT Android Developer Tools. 7, 26, 52, 53
API Application Programing Interface. 7, 20, 22, 34, 39

DAL Data Acces Layer. 7
DDMS Dalvik Debug Monitor Server. 7, 52

E-OTD Enhanced Observed Time Difference. 7, 23

GPS Global Positioning System. 7, 23, 24
GPX GPS eXchange Format. 7
GSM Global System for Mobile Communications. 7, 23

JDK Java Development Kit. 7, 26

KML Keyhole Markup Language. 7, 34

LBS Location Based Services. 7, 24

MIAA Medical Information, Anytime, Anywhere. 6, 7, 14–16
MVC model–view–controller. 7, 37
MVP model–view–presenter. 7, 37

SDK Software Development Kit. 7, 20, 26, 34, 52

TDoA Time Difference of Arrival. 7, 23
TTM Time-to-Market. 7, 20

UI User Interface. 7, 22, 44
UML Unified Modeling Langauge. 6, 7, 28, 30, 32, 33, 37–39, 44, 45, 48–50
UX User Experience. 7

Introduction Générale

Ces temps-ci, le mobile s'est imposé et devient la norme pour les consommateurs. Les statistiques ne le cachent pas, c'était prévisible, mais tous les analystes le soulignent : "Le marché des PC s'effondre face aux smartphones et aux tablettes" [4]. Un des secteurs qui pourrait bien bénéficier de l'avantage des systèmes mobiles est le secteur médical. Les applications mobiles offrent un potentiel énorme pour supporter et activer des nouvelles opportunités pour les services médicaux. La localisation, l'instantanéité, l'efficacité, la personnalisation et une très grande accommodation vont offrir plusieurs moyens nouveaux pour améliorer l'expérience des services médicaux, du côté du patient sûrement, mais tendent aussi à rendre l'établissement plus convivial pour les médecins et en général, le staff médical. Investir dans une application mobile représente pour les hôpitaux, et les institutions qui les implémentent, un autre moyen pour étendre les outils numériques déjà en place, en offrant des fonctionnalités qui sont auparavant clouées aux ordinateurs des administrations. Ceci facilitera le processus de traitement des malades. Cependant, l'usage des smartphones dans les établissements soulève des questions, notamment sur le plan technique. Les techniques d'accès et de sécurisation des données des patients et divers technologies utilisées, surtout le manque de standardisation, posent un sérieux challenge pour les entreprises voulant offrir des solutions pour les établissements médicaux. Dans ce même thème se présente ce Projet de Fin d'Etudes sur la conception et le développement d'une application mobile sur plate-forme Android destinée aux médecins dans le but de faciliter l'accès aux dossiers médicaux des patients en intégrant les techniques de localisation.

Ce rapport s'articule comme suit : La première partie expose le cadre général du projet en présentant l'entreprise hôte ainsi que les objectifs de l'application. Ensuite la deuxième partie évoque les solutions similaires déjà présentes dans le marché ainsi qu'une présentation de la plate-forme sur laquelle l'application est à développer. Après on enchaîne avec les études des besoins, de la structure, du comportement, ainsi que le déploiement et de teste de l'application. En fin on termine par une conclusion générale.

Chapitre 1

Étude de Projet

Ce chapitre est subdivisé en quatre parties : la première partie est consacrée à la présentation de l'organisme d'accueil **TUNAV**. La deuxième partie est destinée à la présentation du projet en soit. Puis on présente une étude du marché en énumérant les applications dont les fonctionnalités sont équivalentes à la notre tout en soulignant les différences qui subsistent. Et en fin on présente la plate-forme ciblée et on passe en revu l'architecture d'une application Android™.

1.1 Présentation de l'organisme d'accueil

TUNAV se situe à la Cité Technologique des Communications, Parc Technologique El Gazala à l'ARIANA, et a été fondée par son Président Directeur Général Mohamed Anis Kallel.

En guise de présentation, rien de mieux que de l'avoir directement du patron lui-même [5] :

"Tunav est une société technologique, créée au mois d'août 2004, implantée à la technopole El Gazala et spécialisée dans la technologie GPS et ses diverses applications dans les domaines de navigation et de gestion de flotte."

Figure 1.1 – Logo Tunav.

"Tunav est connue en Tunisie par son système « LaTrace » de gestion de flotte par GPS, lequel a été commercialisée pour la première fois en Octobre 2005. Il s'agit d'un système articulé autour d'une application très évoluée de gestion de flotte, d'une gamme d'appareils GPS/GPRS et d'une base de données géographique richement renseignée."

Tunav possède un savoir faire reconnu dans le domaine de la localisation qui peut être exploité dans le domaine médical.

1.2 Présentation du projet

Ce projet s'inscrit dans un effort pour faciliter le travail des médecins en leurs rapprochant de leur patient à travers des technologies de localisation. Chaque médecin authentifier a accès à la liste de ses patients ordonnés dans l'ordre de leur cas (urgent ou non), leur proximité géographique dans l'établissement, et leur date d'admission dans l'hôpital. L'application doit être conçue de manière à accommoder aux différentes configurations des clients potentiels avec des modifications minimales.

Cette application vise principalement les médecins. Et malgré que, suite à des choix conceptuels, rien n'empêche qu'avec des modifications minimes une audience plus large dans le corps médical pourra être ciblée, ce n'est pas -pour le moment- le but de l'application. Les médecins, malgré leur formation prolongé dans le domaine médical, représente une cible sans une vraie profondeur technique, ce que requit de l'application d'être le plus simple possible.

1.3 Les Applications du marché

Plusieurs sociétés offrent des solutions en relation avec celle proposée par ce présent rapport. Malheureusement, la plupart d'entre elles sont des solutions commerciales et, faute de documentation disponible, on n'a pas pu les étudier d'un point de vu techno-technique et on s'est contenté de relayer leurs caractéristiques tel que présenté dans les sources citées.

NB : Les solutions présentées ici sont le fruit des sociétés bien établies avec des ressources considérables et des salariés professionnels. Les comparer avec le travail incubé dans ce rapport serait abusif, l'indulgence est de mise.

FIGURE 1.2 – MIAA sur un émulateur Cisco Cius

1.3.1 MIAA - Palomar Pomerado Health

MIAA (figure 1.2) est une application mobile issu d'un projet R&D chez *Palomar Pomerado Health*, l'institution publique la plus large dans l'état de Californie (USA). Elle permet aux médecins d'accéder rapidement au dossier médical complet du patient depuis une variété de sources différentes qui s'affranchient des frontières des organisations [6]. Elle vise les terminaux équipés avec le système d'exploitation Android™ comme les smartphones et les tablettes. *Palomar Pomerado Health* a choisi de déployer cette application dans le *Palomar Medical Center* à *Escondido* (319 lits) et le *Pomerado Hospital* à *Poway* (107 lits) sur des tablettes Cisco Cius [7], ce choix s'est basé sur le support qu'offre Cisco pour ces équipements.

Les avantages de MIAA sont : [8]

- Application mobile facile à utiliser conçue spécifiquement aux médecins, tournant sur la plate-forme Android™.
- Un service *Cloud* qui fournit un accès permanent à l'historique médical des patients à partir de divers sources de données qui s'affranchient des frontières des organisations.
- Interopérabilité avec les pionniers des systèmes électroniques de l'historique médical tel que Cerner - **Millennnium**™, **NextGen**™, et *Veterans Administration* - **VistA**™.
- Intégration en temps-réel des technologies de surveillance des signes vitaux sans fils comme l'ECG, SPO_2, rythme cardiaque, température, respiration, et pression du sang à partir des équipements sans-fils.
- Affichage des informations génétiques personnelles.
- Application dynamique qui s'ajuste automatiquement à l'hôpital, clinique, et à la maison.
- Simple, facile à utiliser, avec une tactile de nouvelle génération.
- Intégration d'une messagerie inter-médecins sécurisée tout en maintenant le contexte du patient.
- Des plan futurs pour intégrer NHIN *Connect* et les services *Direct*.

1.3.2 PowerChart Touch™ - Cerner

PowerChart Touch™ est une solution mobile conçue par le laboratoire Cerner qui fait parti de l'ensemble de solutions **Millennium+**™ et qui

permet de faciliter le travail des médecins. Elle offre une expérience native sur iPad pour gérer les visites médicales et permet aux médecins d'effectuer tout une visite typique qui inclue : [9]

- Consultation des emplois du temps et les chartes des patients.
- Satisfaire les demandes récurrentes comme les commandes simples et les recharges des médicaments.
- Consultation des diagnostics et résultats cliniques.
- Documenter les allergies, les problèmes de santé et l'historique du patient.
- Créer et signer les notes de progressions.

Dès la fin du flux de travail du médecin ambulant. Cerner étend ces mêmes fonctions et les adaptes aux établissements hospitaliers, les urgences et les divers spécialistes. Les avantages clés du PowerChart Touch^{TM} sont : [9]

- Des réponses instantanées avec un flux de travail aisé.
- Pas besoin de configurer l'application.
- Adapter pour les visites médicales, aux patients et aux conditions de la consultation.
- Transmission sécurisée des données.
- Des capacités de reconnaissance vocale.

1.4 Le système d'exploitation Android^{TM}

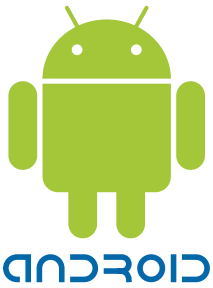


FIGURE 1.3 – Logo et sigle d'Android^{TM}

Android^{TM} est un système d'exploitation basé sur Linux conçu pour les équipements mobiles avec un écran tactile comme les *smartphones* et les tablettes. Développé à l'origine par *Android^{TM}, Inc.* que *Google* a supporté

FIGURE 1.4 – *Google* Nexus 7, un terminal Android™

financièrement et plus-tard acquis en 2005. Android™ a été dévoilé en 2007 parallèlement à la fondation de l'*Open Handset Alliance* : un consortium composé de sociétés dévoué à l'avancement des standards ouverts pour les équipements mobiles. Le premier téléphone Android™ est sorti en Octobre 2008.

La dernière version stable d'Android™ en date (Mai 2013) est 4.2.2 *Jelly Bean* sortie le 11 Février 2013.

Android™ est basé sur le Kernel Linux et utilise pleinement ses capacités de supports matériels exhaustifs. Mais la comparaison avec les distributions Linux, embarqué ou même destiné aux bureaux, s'arrête à ce niveau. [10]

1.4.1 Parts du marché

L'adoption du système d'exploitation Android™ suit une courbe exponentielle depuis quelque temps et la tendance n'est pas prête de s'inverser, selon le dernier rapport du cabinet d'analyse *Strategy Analytics*, Android™ a réussi à capturer environ 68.4% du marché global [4].

Système d'exploitation	Volume de production 3Q2012[13]	Parts du Marché 3Q2012[1]	Volume de production 3Q2011[23]	Parts du Marché 3Q2011[2]	Différence
Android™	136.0	75.0%	71.0	57.5%	91.5%
iOS	26.9	14.9%	17.1	13.8%	57.3%
BlackBerry	7.7	4.3%	11.8	9.5%	-34.7%
Symbian	4.1	2.3%	18.1	14.6%	-77.3%
Windows Phone 7/ Windows Mobile	3.6	2.0%	1.5	1.2%	140.0%
Linux	2.8	1.5%	4.1	3.3%	-31.7%
Autres	0.0	0.0%	0.1	0.1%	-100.0%
Totales	181.1	100.0%	123.7	100.0%	46.4%

TABLE 1.1 – Les six major systèmes d'exploitation mobile en termes de Volume et de parts de marché en 3etrimestre 2012 [3]

	2008	2009	2010	2011	2012[4]
Unités Android™ produites	0.7	7.0	71.1	243.4	333.6
Parts de marché Android™	0.5%	4.0%	23.3%	49.2%	68.2%

TABLE 1.2 – Production et parts de marché entre 2008 et 2012 [3]

1.4.2 Versions Android™ en circulation

Le tableau 1.3 représente les différentes versions d'Android™ et leurs taux d'utilisation respectifs. On remarque que la plupart des terminaux mobiles Android™ sont sous la version 2.3 *Gingerbread* sortie le 6 Décembre 2010, Ceci est du aux fait que plusieurs téléphones bas de gamme sont équipés de cette version et sont encore en production.

1. 3etrimestre 2012
2. 3etrimestre 2011
3. En million d'unité
4. Estimation

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.2%
2.2	Froyo	8	8.1%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	45.4%
3.1	Honeycomb	12	0.3%
3.2		13	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.0%
4.1	Jelly Bean	16	12.2%
4.2		17	1.4%

TABLE 1.3 – Distribution des versions Android™ en circulation qui ont accédé au *Google Play*[5]

1.4.3 Les raisons du succès d'Android™ [10]

Les raisons pour le succès Android™ peuvent être dénombrées comme suit :

Un *Framework* d'Application Riche. Android™ fournit un excellent Software Development Kit (SDK) avec des Application Programing Interface (API) stable à long-terme, ce qui assure aux partenaires tiers un écosystème standardisé. Alors que le système en lui même est en constante évolution, la stabilité des API pour la plupart est préservée, ce qui permet d'investir dans le long-terme. Concevoir et construire des applications pour les distribuer sur différentes plate-formes permet des réductions drastiques en termes des coûts et effort pour les entreprises.

Un Time-to-Market (TTM) Agressif. Concevoir des appareils avec Android™ peut réduire le TTM d'une manière significative. Il suffit de se procurer les sources, les adapter pour le matériel en question et vendre. Et dans le cas ou les schémas et usages de référence sont appliqués, la sortie d'un nouveau produit est possible au cours de quelque mois. Seulement voilà, ce n'est pas aussi facile et une certaines expertises et connaissances dans ce domaine sont requises. Et même si sortir un système basé sur Android™ peut être plus rapide comparé à d'autres solutions, le suivi des évolutions du système ainsi que maintenir le code à long terme est une autre histoire.

5. Données récoltées pendant une période de tests de 14 jours arrêtée le 4 Février 2013.

Concentrer sur « Ce qui compte réellement ». En fournissant un *Framework* pratique, Android™ permet aux développeurs de se concentrer sur les aspects à valeur commerciale. L'assemblage d'un appareil est une activité qui consomme énormément de temps et de ressources et n'a pas à réinventer un - encore - autre système d'exploitation permettant d'éviter un autre gaspillage de temps.

Open Source. Bien qu'il ne soit pas développé d'une manière communautaire, Android™ reste 100% modifiable et diffuse un sentiment de sécurité parmi les entreprises contre les menaces légales.

1.4.4 La pile logicielle d'Android™ [11]

D'une manière simple. La pile logicielle d'Android™ est un Kernel Linux et une collection de bibliothèques C/C++ exposée à travers un framework d'application qui fournit des services pour l'environnement d'exécution et les applications. On peut énumérer les éléments composant la pile logicielle comme suit :

Kernel Linux : Services de base qui incluent les pilotes matériels, gestion des processus et de la mémoire, sécurité, réseaux et gestion d'autonomie ; fourni aussi une couche d'abstraction entre le matériel et le reste de la pile.

Bibliothèque : Se situe au dessus du Kernel, Android™ inclue divers bibliothèques C/C++ de base comme *libc* et *SSL* ainsi que :

- Une bibliothèque multimédia pour la lecture des fichiers audio et vidéo.
- Un *Surface manager* pour la gestion de l'affichage.
- Des bibliothèques graphiques qui incluent le *SGL* et *OpenGL* pour les graphiques 2D et 3D.
- Un support natif de base de données à travers la base de données *SQLite*.
- *SSL* et *WebKit* pour le navigateur web intégré et la sécurité internet.

Environnement d'Execution (runtime) Android™ : L'environnement d'exécution et le facteur qui sépare un terminal Android™ d'une implémentation Linux mobile. En cohérence avec les bibliothèques de base et la machine virtuelle *Dalvik*, l'environnement d'exécution Android™ est le moteur qui fait fonctionner les applications et, avec les bibliothèques, forme les bases du framework application.

Bibliothèque de Base : Même si la plus part des applications Android™ sont écrits avec du langage *Java*, *Dalvik* n'est pas une machine virtuelle java. Les bibliothèques Android™ de base fournit la plus part des fonctionnalités qu'on retrouve dans les bibliothèques de base *Java*, en plus de quelque bibliothèques spécifiques à Android™.

La Machine Virtuelle devDalvik : *Dalvik* est une machine virtuelle qui a était optimisée pour s'assurer que chaque terminal peut faire fonctionner plusieurs instance d'une manière efficace. Il s'appuie sur le Kernel Linux pour le threading et la gestion bas niveaux de la mémoire.

Le *Framework* Application : Le *Framework* application fournit les classes utilisées pour créer les application Android™. Il fournit une abstraction générique pour l'accès matériel et gère l'User Interface (UI) et les ressources de l'application.

Couche Application : Toute application, quelle soit native ou produite par un tiers, est construite sur la couche application via les même API. La couche application opère à l'intérieur de l'environnement d'exécution Android™, utilisant les classes et les services mis à disposition par le *framework* application.

1.4.5 Architecture des applications Android™ [11]

L'architecture d'Android™ encourage la réutilisation des composants, ce qui nous permet de publier et de partager des *Activities*, services, et données avec d'autres applications. Avec une gestion d'accès gérée par les restrictions de sécurité que nous définissons.

Le même mécanisme qui nous permet de produire un gestionnaire de contact alternatif ou un compositeur de numéros nous permet aussi d'exposer les composants de notre application pour permettre à d'autres développeurs de les réutiliser en créant des nouveaux UI ou d'étendre des fonctionnalités.

Les services application suivants représentent les bases architecturales de toute application Android™, fournissant le *Framework* qu'on va utiliser pour notre application.

L'*Activity Manager* et le *Fragment Manager* : Contrôle le cycle de vie de nos *Activities* et nos *Fragments* respectivement, y inclue la gestion de la pile des *Activities*.

Views Utilisé pour construire l'UI de notre *Activities* et *Fragments*.

***Notification Manager* :** Fournit un mécanisme consistant et non-intrusif de signalisation pour l'utilisateur.

***Content Providers* :** Permet à notre application le partage des données.

***Resource Manager* :** Offre un moyen d'externaliser les ressources (comme par exemple les chaînes de caractères et les images.)

***Intents* :** Présente un mécanisme pour transférer les données entre les applications et leurs composants.

Parmi les fonctionnalités les plus intéressantes pour l'aboutissement de notre projet offerte par Android™ sont ses capacités de localisation, étudiées dans la partie suivante.

1.4.6 Location Based Services

1.4.6.a Concept

Pour positionner un terminal, on spécifie ses coordonnées géographiques en utilisant le géo-codage.

Géo-codage [12] Le géo-codage est le processus de retrouver les coordonnées géographiques associées (exprimées souvent en terme de *latitude* et *longitude*) d'après d'autre données géographiques comme l'adresse de la rue, code postal. Ces coordonnées géographiques peuvent être insérées dans un système d'informations géographiques ou intégrés dans des médias comme les photos numériques par le biais de géo-marquage. Cette opération est communément appelée le *Forward Geocoding*.

Le *Reverse Geocoding* est la procédure inverse : retrouver les lieux textuel comme l'adresse de la rue d'après les coordonnés géographiques. Car même si l'usage des paramètres comme la longitude et la latitude fournit un moyen pratique pour localiser l'individu d'une manière relativement précise. Les utilisateurs penchent à penser en terme de rues et adresses.

A fin de déterminer la position du terminal, plusieurs technologie de localisation sont à notre disposition.

Localisation par GSM On peut retrouver la position du terminal mobile par le biais de sa cellule Global System for Mobile Communications (GSM). Cette technique fait intervenir divers moyens de triangulation des signaux parvenant depuis les cellules qui desservent un téléphone mobile. La position géographique du terminal est déterminée par une multitude de méthodes comme la Time Difference of Arrival (TDoA) ou l'Enhanced Observed Time Difference (E-OTD).

Localisation parGPS [13] Global Positioning System (GPS) est un système de navigation par satellites qui fourni la localisation et le temps dans toute condition météorologique et partout sur terre s'il existe un accès non bloquant à 4 satellites GPS ou plus. Ce Système fournit des services essentiels dans le domaine militaire, civil et commercial partout dans le monde. Il est maintenu par les États Unis d'Amérique et accessible à quiconque possédant un récepteur GPS.

1.4.6.b La localisation dans Android™ [14]

L'accès aux Location Based Services (LBS) se fait essentiellement via deux objets :

Location Manager [1] Permet d'exploiter les services basés sur la localisation.

Location Providers [2] Chaque *Providers* représente une technologie de localisation utilisé afin de déterminer la localisation actuelle du terminal.

On utilise ces deux Classes pour les fins suivantes :

- Obtenir la position actuelle.
- Suivre les mouvements.
- Alerte de proximité dans le cas ou l'on approche ou l'on s'éloigne d'une zone spécifique.
- Retrouver les fournisseurs de localisation disponible.
- Observer le status du récepteur GPS.

Généralement deux techniques de détection de localisation sont disponibles dans le terminal : détection par le réseau *Network Provider* et la détection par GPS (*GPS Provider*). Le choix de la technologie à utiliser est soit explicite ou automatique suivant des critères prédéfinis par le développeur de l'application. Avant de pouvoir exploiter un service de localisation, un niveau de précision doit figurer dans le manifeste de l'application via les *uses-permission tags*.

Listing 1.1– Permission pour la localisation par le réseau.

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

1. android.location.LocationManager
2. android.location.LocationProvider

Listing 1.2– Permission pour la localisation par GPS.

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

A noter qu'une application ayant la permission *FINE* possède implicitement la permission *COARSE*.

Avoir la localisation dans la plate-forme Android™ se fait par le biais de *callback*, on indique au *LocationManager* qu'on veut recevoir des mises à jour de la localisation par la fonction *requestLocationUpdates()* en lui passant une implémentation de *LocationListener*[3]. Cette interface contient plusieurs fonctions de *callback* que le *LocationManager* appelle quand la localisation de l'utilisateur change ou l'état du service change [15].

3. android.location.LocationListener

Chapitre 2

Analyse des besoins

Dans ce chapitre on présente les environnements logiciels et matériels dans lesquelles s'est dérouler le développement de l'application ainsi qu'une étude des besoins effectué pour cerner nos objectifs.

2.1 Environnement de développement

Plusieurs outils ont été mises à contribution pour développer l'application, tant sur le plan logiciel que matériel.

2.1.1 Environnement Logiciel

Voici une liste des outils logiciels utilisés pendant le développement de l'application.

Ubuntu 12.04 Système d'exploitation.[1]

OpenJDK 6 Java Development Kit (JDK) version 6.[2]

Eclipse Juno Environnement de Développement Intégré dans sa version *Service Release 2*.[3]

Android Developer Tools (ADT) (plugin Eclipce) Intégration des outils de développement fournit dans l'SDK Android™.[4]

ObjectAid (plugin Eclipce) Génération des diagrammes de classes.[5]

PlantUML (plugin Eclipce) Génération des diagrammes de séquences.[6]

Git Gestionnaire des versions[7].

EGit (plugin Eclipse) Intégration du gestionnaire de version.[8]

Evolus Pencil Génération de prototypes et des sketchs[9]

2.1.2 Environnement Matériel

Le développement de l'application est fait avec une tablette Asus Nexus 7 (mise à jour à Android™ 4.2.2 *Jelly Bean*).

2.2 Étude des Besoins

2.2.1 Besoins fonctionnels

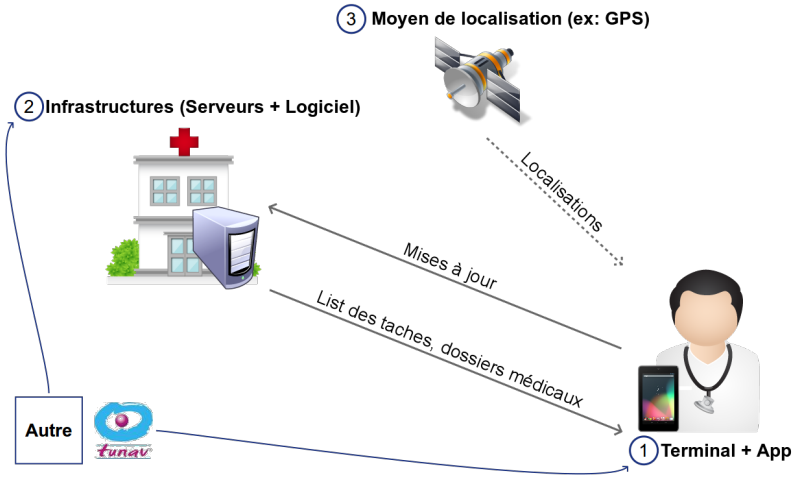


FIGURE 2.1 – Illustration des besoins fonctionnels.

- Le médecin doit être capable à partir de son terminal d'avoir des informations sur les patients qui lui sont assignés en fonction de leur position géographique.
- L'application doit être capable de détecter la proximité d'un patient en fonction de la position actuelle du terminal.
- Le médecin peut télé-consulter le dossier médical du patient.

1. http ://www.ubuntu.com
2. http ://openjdk.java.net
3. http ://www.eclipse.org
4. https ://developer.android.com/tools/sdk/eclipse-adt.html
5. http ://www.objectaid.com/
6. http ://plantuml.sourceforge.net/
7. ttp ://git-csm.com
8. http ://www.eclipse.org/egit/
9. http ://pencil.evolus.vn/

2.2.2 Besoins non fonctionnels

– Une bonne ergonomie qui vise à faciliter l'obtention de l'information, avec un minimum d'effort pour l'utilisateur cible et avec le moindre risque d'erreur. Les choix graphiques et conceptuels sont des considérations à tenir en compte.

2.2.3 Besoins techniques

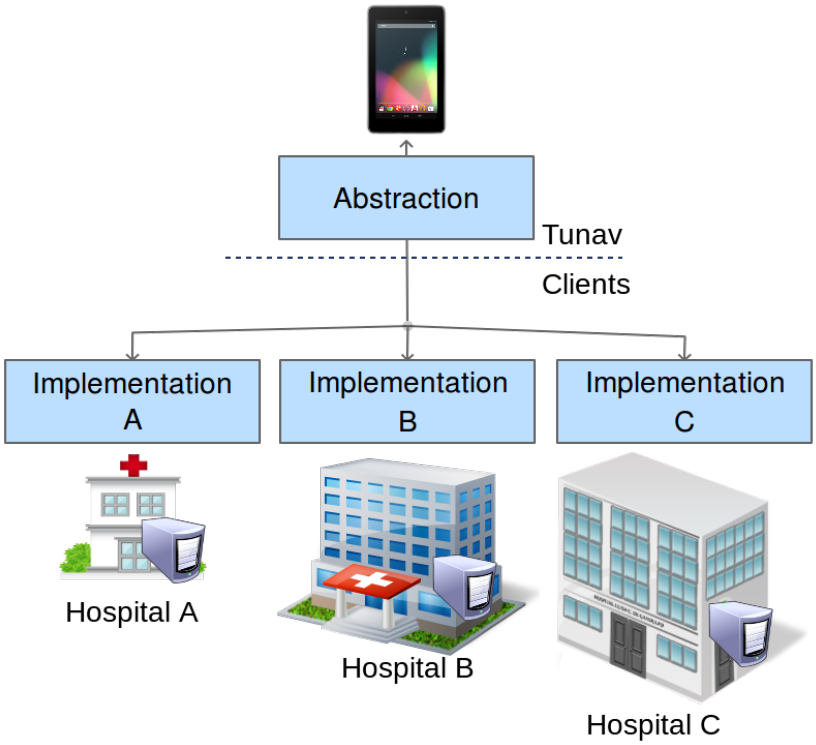


FIGURE 2.2 – Illustration des besoins techniques.

– L'application mobile vise à utiliser les systèmes déjà en place des établissements clients pour réduire les coûts. Vu l'absence d'un protocole standards et les différentes implémentations possibles des différents clients, l'implémentation d'une couche d'accès abstraite est requise pour pouvoir déployer l'application avec le minimum de modification.

2.2.4 Identification des acteurs

Notre système interagit essentiellement avec trois acteurs différents :

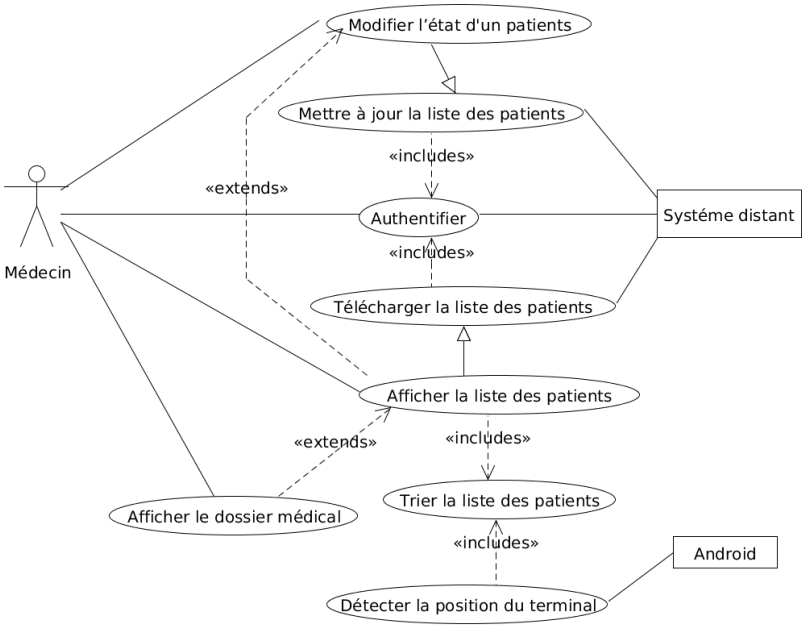


FIGURE 2.3 – Diagramme des cas d'utilisation UML globale.

Le médecin C'est l'acteur principal de notre système.

Le service web Source des données à acheminer vers le médecin.

Système d'exploitation Communique à notre système les informations recueillies des divers composants qui nous intéressent (localisation GPS/-Network, état de la connectivité, état de la batterie).

2.2.5 Cas d'utilisations

On peut présenter les interactions fonctionnelles entre les acteurs gouverné par leurs besoins avec un diagramme des cas d'utilisations (figure 2.3).

Chapitre 3

Conception

Dans ce chapitre on présente la structure globale de l'application en utilisant les diagrammes structurels de la modélisation UML.

3.1 Couche D'Accés aux Données

Un des objectifs principaux de ce projet étant de fournir une solution d'accès aux données flexibles afin de couvrir les besoins de chaque client de manière individuelle. On a opté donc pour un modèle basé sur l'implémentation de deux interfaces (figure 3.1) :

- Interface d'authentification.
- Interface d'accès à la liste des patients.

L'idée est simple : pour chaque client, une implémentation spécifique à son infrastructure sera développée soit par son propre effectif, soit par une des équipes de Tunav, ou dans le cas idéal par une alliance formé par des agents des deux camps qui garantit une collaboration plus poussée pour des résultats meilleurs. Ces ensembles d'interfaces nous permettent de construire notre application.

3.1.1 Interface d'authentification

AuthenticationHandler (figure 3.1) est une classe abstraite comportant les méthodes requises par notre application pour effectuer les actions d'authentification, de dé-authentification, de vérification d'authenticité ainsi l'obtention des informations associées à l'utilisateur authentifié.

0. com.tunav.tunavmedi.dal.abstraction.AuthenticationHandler

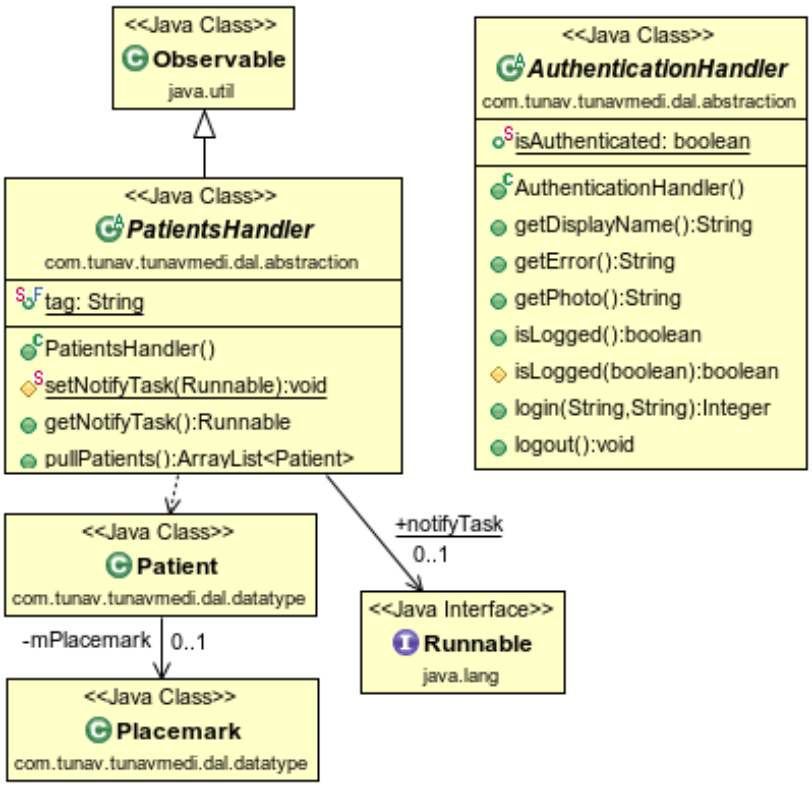


FIGURE 3.1 – Diagramme de classes UML des interfaces de la couche d'accès.

Malgré la variété des techniques d'authentification utilisée dans le domaine informatique, l'étape d'acquisition des identificateurs de l'utilisateur représente un point de départ commun. On utilise ce caractère dans l'interface d'authentification en demandant à nos clients d'implémenter la méthode *login()* qui prend en argument l'identifiant et le mot de passe fourni par le médecin, dans le cas d'une éventuelle erreur d'authentification, l'implémentation met a notre disposition un message d'erreur accessible par la méthode *getError()*. Pour effectuer l'opération inverse le client implémente la méthode *logout()* supposée annoncée au service distant la dé- authentification de l'utilisateur du terminal. Pour vérifier le l'état actuel de la relation du terminal avec la base distante, on utilise le booléen retourné par *getStatus()*, utile dans les cas de déconnexion temporaire ou du redémarrage de notre application. Les méthodes *getDisplayName()* et *getPhoto()* retournent respectivement le nom de l'utilisateur et sa photo.

3.1.2 Interface d'accès à la liste des patients

PatientsHandler (figure 3.1) est une classe abstraite comportant les méthodes requises par notre application pour effectuer les actions de mise à jour de la liste des patients dans le deux sens (terminal → service et terminal ← service), elle contient aussi un objet de type *Runnable* associé au mécanisme de notification.

3.1.2.a Mécanisme de notification

Le patron **Observateur** (*observer pattern*) (fig 3.2) et un patron de conception couramment utilisé et qui nous permet d'avoir une relation 1→N entre divers objets. Le patron observateur assume que l'objet qui contient les données est séparé de l'objets qui les affiche et ces dits objets observent le changement de ces données [16]. Quand on implémente le patron observateur, on se réfère communément à l'objet contenant les données par "Sujet" ; et chacun des consommateurs des données par "Observateur", Et chaque Observateur implémente une interface préconçue que le Sujet invoque quant les données changent [16].

0. com.tunav.tunavmedi.dal.abstraction.PatientsH andler
0. java.lang.Runnable

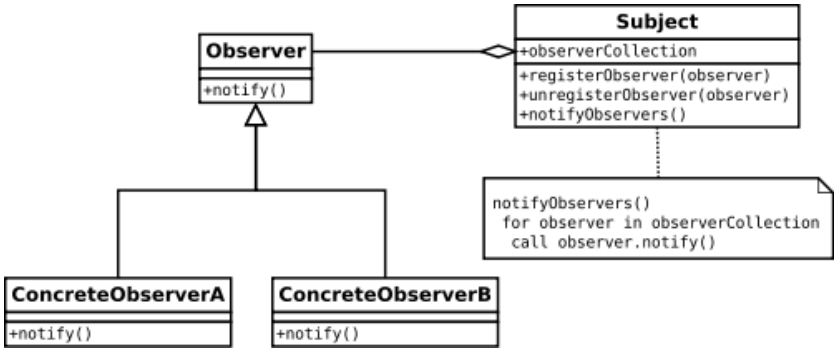


FIGURE 3.2 – Diagramme UML du patron de conception Observateur [1]

Dans le langage Java, ce patron est réalisé à travers la classe *java.util.Observable* et l'interface *Observer*[1]. Le Sujet hérite de la classe *Observable* et les changements sont signalés par les méthodes *setChanged()* et *notifyObservers()* ou *notifyObservers(Object message)*.

3.1.2.b Les objets de données

La communication des données avec le service distant se fait à travers l'objet *Patient*[2] (figure 3.3. Cet objet contient toutes les informations requises pour la synchronisation et l'affichage et la gestion des patients, en particulier le dossier médical et la position actuelle du patient.

Synchronisation : Étant sujet aux modifications de la part de l'application et du serveur distant, un problème se pose pour connaître la version la plus à jour. Pour cela chaque modification apportée est suivi par une mise à jour de variable *mUpdated* par le temps à cet instant précis (cette opération est interne à l'objet). En cas oú deux versions différentes de l'objet *Patient* avec le même *mID* et *mUpdated*, le service est supposé favoriser sa version.

Dossier médical : Le dossier médical est fournit sous le format *HTML*. Cette représentation est idéale car elle nous permet de faire abstraction sur le format du dossier tel que sauvegardé par l'établissement client, la couche d'accès assurant éventuellement la conversion.

Position : La position actuelle du patient est représentée par un objet *Placemark*[3] inspiré par la notation XML Keyhole Markup Language (KML).

1. Java.util.Observer
2. com.tunav.tunavmedi.dal.datatype.Patient
3. com.tunav.tunavmedi.dal.datatype.Placemark

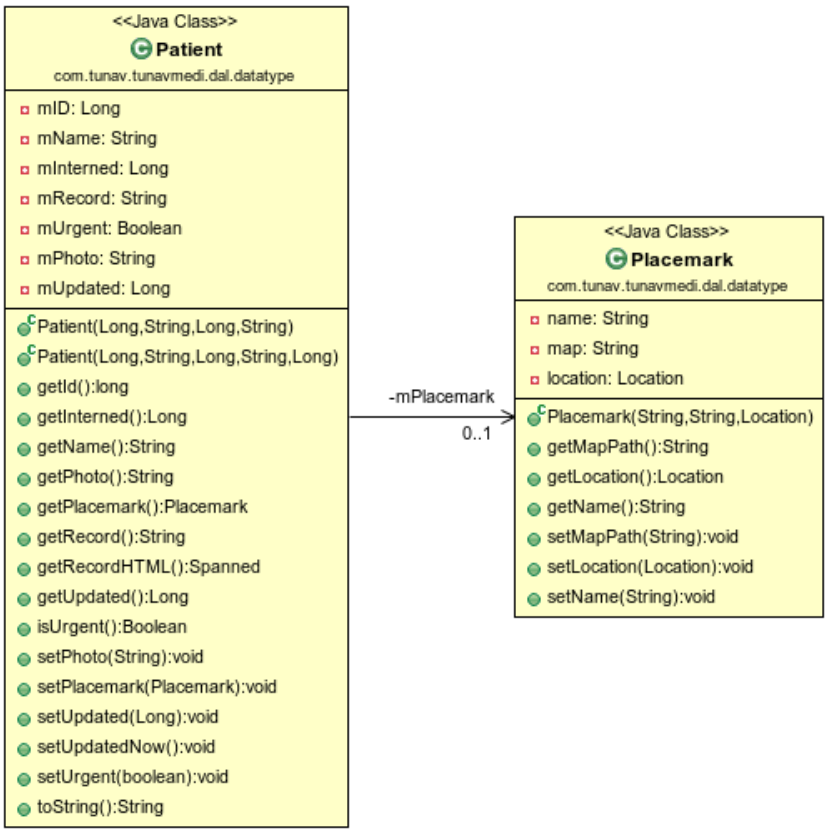


FIGURE 3.3 – Diagramme de la classe UML *Patient*

Les coordonnées sont représentées par un objet de type *Location*[4].

3.1.3 Implémentation de tests

Le package *com.tunav.tunavmedi.dal.sqlite* contiens une implémentation de la couche d'accès abstraite (figure 3.4) réalisée dans le cadre de ce projet pour pouvoir tester la solution. Cette implémentation est de caractère local à l'application à travers les API de la base de données *SQLite* qui fait partie de l'SDK Android™. En fait une implémentation locale nous affranchie des problèmes qui peuvent se produire et dont la corrélation avec l'application est faible. Cette même idée a influencé la mise en place même de cette implémentation qui à su rester la plus simple possible en restant très proches des objets de base de notre application.

Cette implémentation peut être subdivisée en trois éléments : Les *Contrats*, les *Helpers*, et la classe *DBSetup*.

Les Contrats : Représente les contrats relatifs aux tables dans notre implémentation de tests. Chaque contrat implémente l'interface *BaseColumns*[5] et contient - entre autre - les commandes SQL de création et de suppression de la dite table, des éventuels index, et les commande d'insertion des données de test.

Les *Helpers* : Ce sont les implémentations des classes abstraites qui définissent la couche d'accès et présentent les procédures d'extraction des données préinsérées dans nos tables fictives en faisant appel à la classe *DBSetup* .

La classe *DBSetup* : Elle hérite de la classe *SQLiteOpenHelper* et est destinée à contrôler la création et l'accès à notre base de données de tests.

3.2 Structure de l'Application

L'application est subdivisée en deux parties majeures représentées par deux classes de type *Activity* :

LoginActivity : C'est une entité indépendante qui implémente la logique d'authentification,

4. android.location.Location
5. android.provider.BaseColumns
5. android.app.Activity

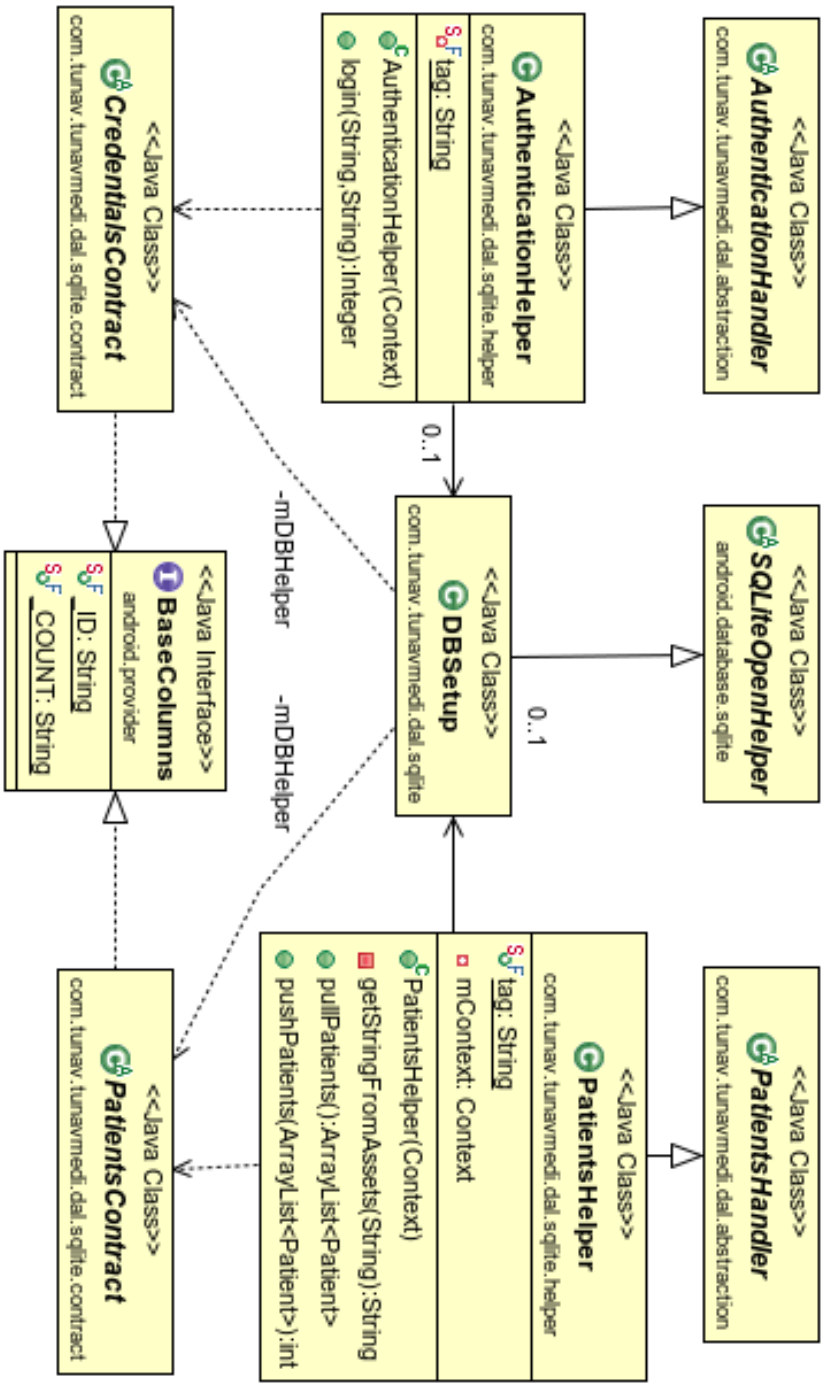


FIGURE 3.4 – Diagramme de classe de l'implémentation de la couche d'accès de tests à base de SQLite.

MainActivity : C'est l'entité principale de notre solution mobile, elle relie les divers composants utilisés dans la transmission de la liste des patients, de la localisation et de la Conscience de l'état du terminal.

3.2.1 LoginActivity

La *LoginActivity* s'occupe de la partie d'authentification et de dé-authentification. Elle utilise un *AsyncTask*[6] pour effectuer ces opérations de manière non bloquante.

Listing 3.1– Déclaration de LoginActivity dans AndroidManifest

```
<activity
    android:name="com.tunav.tunavmedi.activity.
       LoginActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait" >
    <intent-filter>
        <category android:name="android.intent.
           category.DEFAULT" />

        <action android:name="com.tunav.tunavmedi.
           action.LOGOUT" />
        <action android:name="com.tunav.tunavmedi.
           action.LOGIN" />
    </intent-filter>
</activity>
```

3.2.2 MainActivity

Listing 3.2– Déclaration dans AndroidManifest de MainActivity

```
<activity
    android:name="com.tunav.tunavmedi.activity.
       MainActivity"
    android:label="@string/title_activity_main"
    android:screenOrientation="portrait" >
    <intent-filter>
        <action android:name="android.intent.action.
           MAIN" />
```

6. android.os.AsyncTask

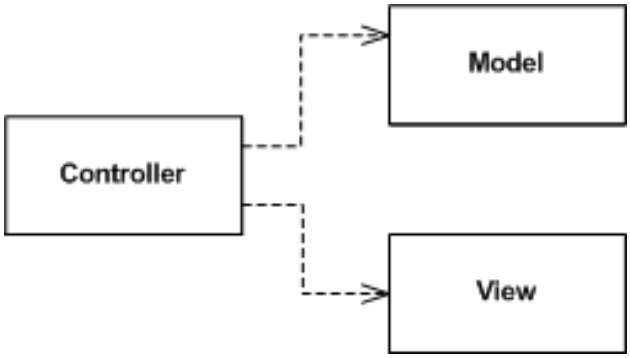


FIGURE 3.5 – Diagramme UML de composants du patron *Passive View* [2]

```
            <category android:name="android.intent.
                category.DEFAULT" />
            <category android:name="android.intent.
                category.LAUNCHER" />
        </intent-filter>
    </activity>
```

L'architecture globale du *MainActivity* (figure 3.6) est calquée sur Le patron "Vue Passive" (Passive View Pattern). Le patron *Passive View* (fig 3.5) est une variation des patrons model–view–controller (MVC) et model–view–presenter (MVP), de ce qui ce passe dans ces patrons.

L'interface utilisateur est divisée entre une Vue qui s'occupe de l'affichage des données et un contrôleur qui répond aux interactions de l'utilisateur. La différence majeur avec le *Passive View* est que la Vue est complètement passive et n'est pas responsable de sa mise à jour depuis le modèle. Dans ce cas toute la logique de la Vue est dans le contrôleur et aucune dépendance ni dans un sens au dans un autre entre la Vue et le modèle [2].

Ce patron est idéal dans notre cas pour deux raisons majeures :

- Dans notre projet la Vue n'est pas la partie la plus importante dans la mesure où l'objectif est d'intégrer un système développé parallèlement, donc éventuellement avec une autre logique de présentation. Déporter les interactions avec le modèle dans le contrôleur permet d'intégrer d'autres implémentations d'affichage plus facilement.
- La nature même de cette procédure d'accès - à savoir l'aspect abstrait, donc plus fragile - nous conduit à réduire les composants en relations pour réduire la marge d'erreur possible et faciliter les tests d'intégration.

Dans la suite de ce chapitre, on procède à l'explication détaillée du Contrôleur et de la Vue, pour le Modèle se vous reporter au point 3.1.3.

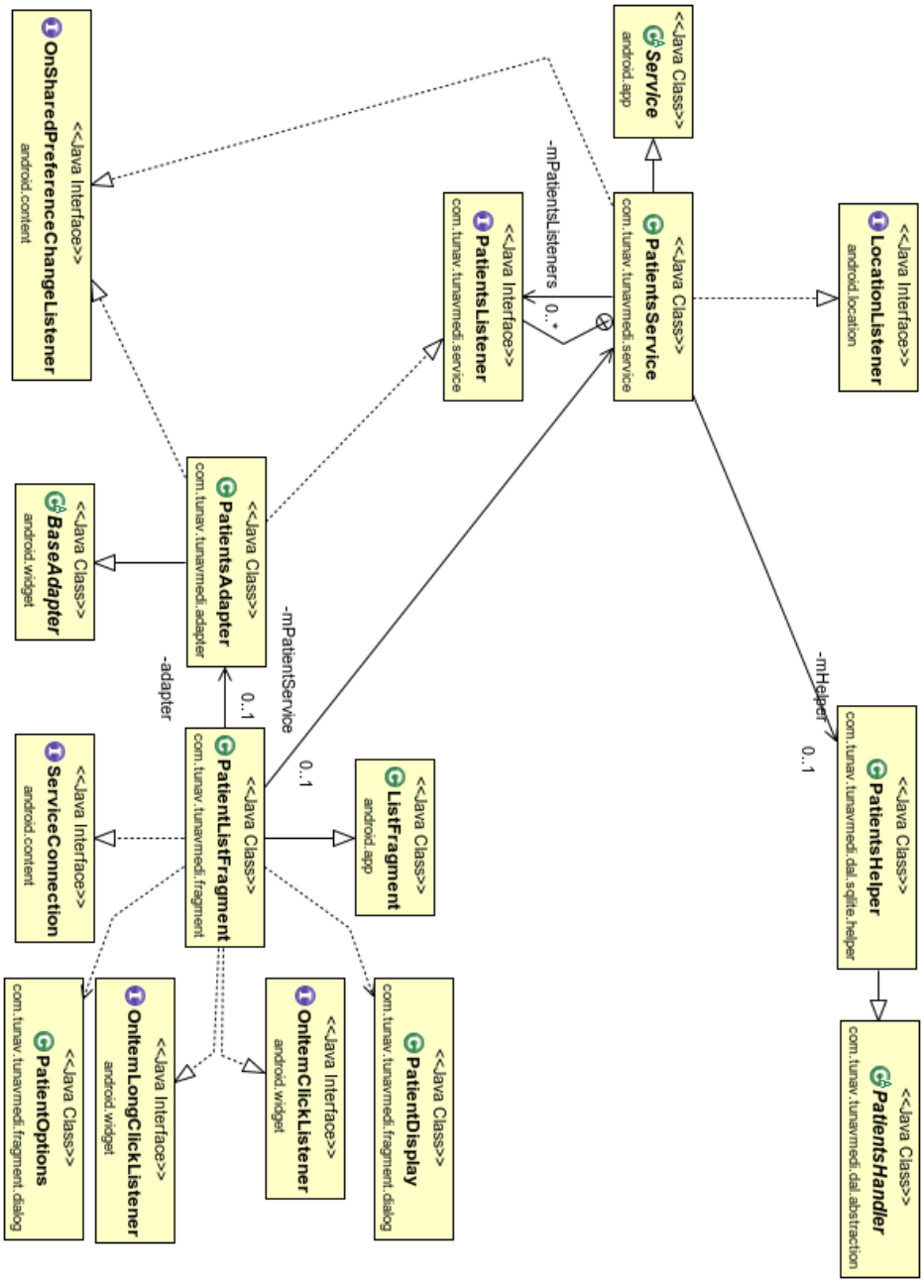


FIGURE 3.6 – Diagramme de classes UML de l'architecture générale de l'application.

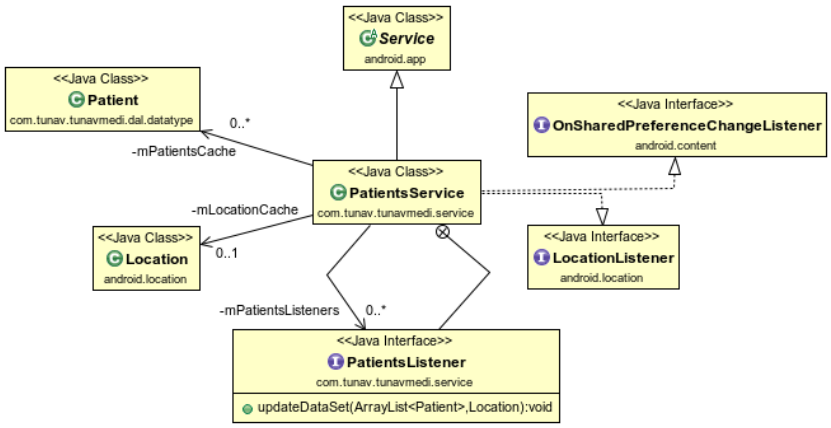


FIGURE 3.7 – Diagramme UML de classe du contrôleur.

3.2.2.a Le Contrôleur

Listing 3.3– Déclaration dans AndroidManifest du PatientService

```
<service
    android:name="com.tunav.tunavmedi.service.
       PatientsService"
    android:exported="false" >
    <intent-filter>
        <category android:name="android.intent.
           category.DEFAULT" />
    </intent-filter>
</service>
```

Notre contrôleur est matérialisé par l'objet *PatientService*[7] qui hérite de la classe *Service*[8]. L'API Android™ définie un service comme étant un composant de l'application qui représente soit la volonté de cette application de faire des longues opérations sans interagir avec l'utilisateur ou d'offrir des fonctionnalités à l'intention des autres applications [17].

Localisation : Pour la localisation, le contrôleur implémente les techniques présentées dans 1.4.6.b

Conscience de l'état du terminal Le tableau 3.1 représente la configuration que le contrôleur suit dans le cas d'un changement d'état du terminal. En particulier l'inutilité de chercher la position actuelle du terminal dans le cas ou celui ci est en charge, encore en cas ou le terminal annonce que la

7. com.tunav.tunavmedi.service.PatientService
8. android.app.Service

batterie est faible, on procède à un mode d'économie d'énergie pour éviter entre autres la corruption des données.

	Mises à jour	Localisation
Batterie Faible	déactivé	déactivé
Batterie en Charge	activé	déactivé
Pas de connectivité	déactivé	activé

TABLE 3.1 – Configuration du contrôleur en réponse au changement d'état du terminal

Connectivité : Les permissions citées dans le listing 3.4 sont nécessaires pour s'abonner aux événements liés à la connectivité du terminal. Le *NetworkReceiver*[9] (listing 3.5) s'occupe de notifier les intéressés, dans notre cas le contrôleur, via des *SharedPreferences*.

Listing 3.4– Déclaration dans AndroidManifest des permissions d'accès à l'état des interfaces réseaux.

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

Listing 3.5– Déclaration dans AndroidManifest du NetworkReceiver

```
<receiver
    android:name="com.tunav.tunavmedi.
        broadcastreceiver.NetworkReceiver"
    android:enabled="true" >
    <intent-filter>
        <action android:name="ConnectivityManager.
            CONNECTIVITY_ACTION" />
    </intent-filter>
</receiver>
```

Batterie Le *BatteryReceiver*[10] (listing 3.6) fait savoir au contrôleur via des *SharedPreferences* si la batterie est faible ou pas.

Listing 3.6– Déclaration dans AndroidManifest du BatteryReceiver.

9. com.tunav.tunavmedi.broadcastreceiver.NetworkReceiver
10. com.tunav.tunavmedi.broadcastreceiver.BatteryReceiver

```
<receiver
    android:name="com.tunav.tunavmedi.
       broadcastreceiver.BatteryReceiver"
    android:enabled="true" >
    <intent-filter>
        <action android:name="android.intent.action.
           BATTERY_LOW" />
        <action android:name="android.intent.action.
           BATTERY_OK" />
    </intent-filter>
</receiver>
```

Mobilité : Le *ChargingReceiver*[11] (listing 3.5) est destiné à notifier le contrôleur quand le terminal est connecté ou déconnecté du chargeur via des *SharedPreferences*.

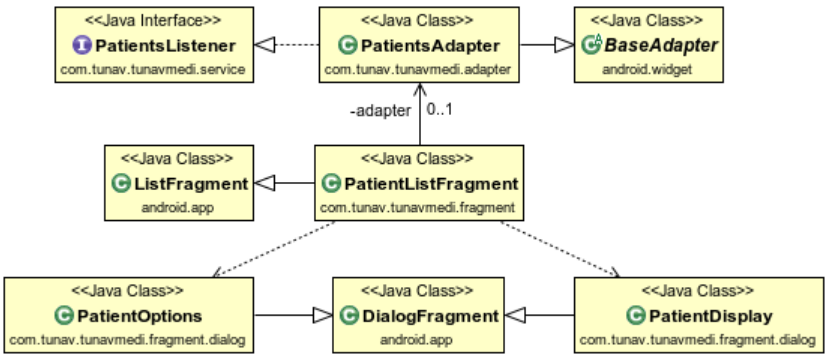
Listing 3.7– Déclaration dans AndroidManifest ChargingReceivers.

```
<receiver
    android:name="com.tunav.tunavmedi.
       broadcastreceiver.ChargingReceiver"
    android:enabled="true" >
    <intent-filter>
        <action android:name="android.intent.action.
           ACTION_POWER_CONNECTED" />
        <action android:name="android.intent.action.
           ACTION_POWER_DISCONNECTED" />
    </intent-filter>
</receiver>
```

3.2.2.b La Vue

Le système d'exploitation Android™ rend facile le développement des applications qui tournent sur des appareils qui possèdent des formes et des tailles d'écrans différentes, une des améliorations apportées dans Android™ 3.0 Honeycomb sont les *Fragments* censés décomposer les fonctionnalités et les interfaces utilisateur d'une l'application Android™ en des modules réutilisables. Notre implémentation de la Vue prend avantage de cette introduction en utilisant des *Fragments* et se compose essentiellement de 4 composants :

11. com.tunav.tunavmedi.broadcastreceiver.ChargingReceiver



PatientAdapter Représente un *BaseAdapter* qui joue le rôle d'un adaptateur entre la *PatientListFragment* et notre contrôleur, la communication avec celui-ci est assuré à travers l'interface *PatientsListener*.

PatientListFragment Hérite de l'objet *PatientListFragment* et s'occupe de l'affichage de la liste des patients.

PatientDisplay Un *DialogFragment* qui s'occupe de l'affichage du dossier médical du patient

PatientOptions Un *DialogFragment* qui permet au docteur de modifier la condition d'un patient.

Algorithme de Trie Lors de l'affichage de la liste des patients, une opération de trie est appliquée pour faciliter la tache du docteur en mettant en valeur les cas qui requièrent le plus son attention. L'algorithme se base sur les conditions suivantes (dans l'ordre) :

1. Le patient est un cas urgent ou non.
2. Le patient est à proximité ou non.
3. La date d'admission du patient.

Cette opération est réalisé la méthode static *java.util.Collections.sort()* en utilisant notre propre objet de type *java.util.Comparator¡Patient¿* qui respecte les conditions citées ci-dessus.

Affichage du dossier médical Le dossier étant sous la forme d'un document HTML, pour l'afficher on utilise un *WebView*[12] (listing 3.8)qui nous offre les capacités d'affichage d'un vrai navigateur web.

Listing 3.8– Déclaration XML du WebView utilisé par PatientDisplay

12. android.webkit.WebView

```
<WebView
    android:id="@+id/task_dialog_description"
    style="@style/ListDescription"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/task_dialog_separator"
    android:textIsSelectable="true"
    android:singleLine="false"/>
```

Chapitre 4

Réalisation et Tests

4.1 Navigation dans l'interface utilisateur

La figure 4.1 modélise par un diagramme UML d'états le schéma que l'utilisateur suit pour naviguer dans l'interface utilisateur de notre application. Quelque Capture d'écran sont inclue pour illustrer le processus (figures 4.2, 4.3, 4.4, 4.5).

4.2 Authentification

La figure 4.6 présente le comportement de l'application pour réaliser l'opération d'authentification de l'utilisateur à travers un diagramme de séquence UML. On peut décrire textuellement le processus par les points suivants :

Acteurs : Docteur.

Pré-condition : Le docteur est déjà inscrit dans la base de données du service et son identifiant et mot de passe lui sont fournits.

Post-condition : L'utilisateur est authentifié.

Scénario nominal :

1. L'utilisateur lance ou retourne à l'application mobile donc *MainActivity*.
2. *MainActivity* détecte que l'utilisateur n'est pas déjà authentifié et actionne l'UI d'authentification (appel à *LoginActivity*).
3. L'utilisateur saisit son identifiant et mot de passe.
4. L'application interpelle le service pour vérifier que la combinaison identifiant / mot de passe est correcte.
5. Le service distant retourne une réponse favorable, *LoginActivity* enregistre les données relatives à l'utilisateur.

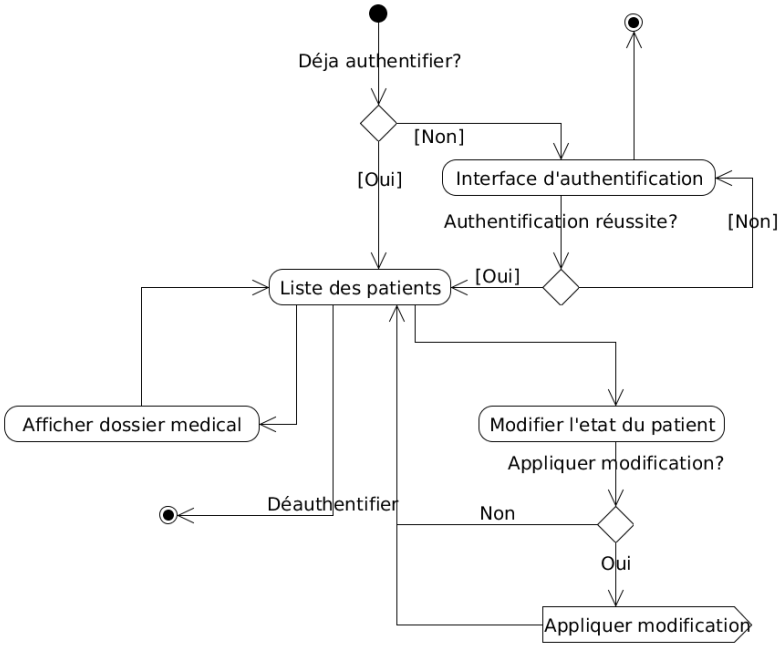


FIGURE 4.1 – Diagramme UML d'activités de la navigation dans l'interface utilisateur.

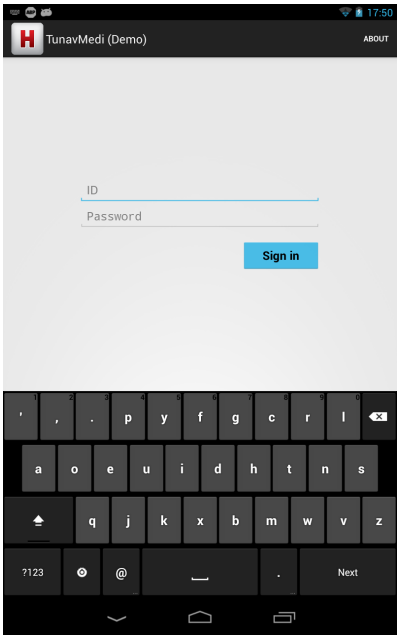


FIGURE 4.2 – Interface graphique d'authentification.

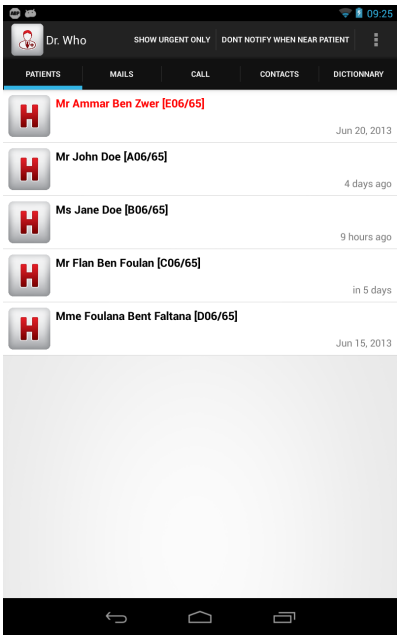


FIGURE 4.3 – Capture écran de l'interface utilisateur de la liste des patients

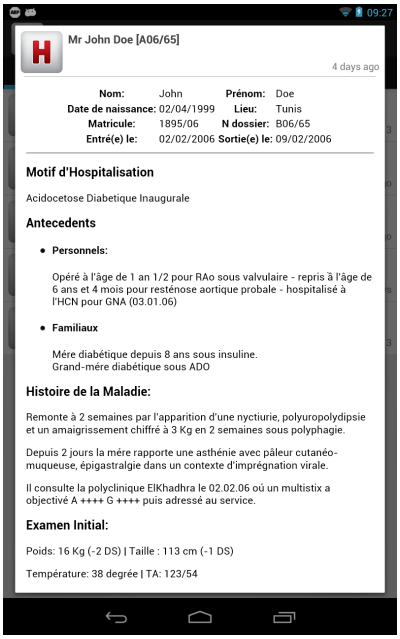


FIGURE 4.4 – Capture écran de l'interface utilisateur relative à l'affichage du dossier médical du patient.

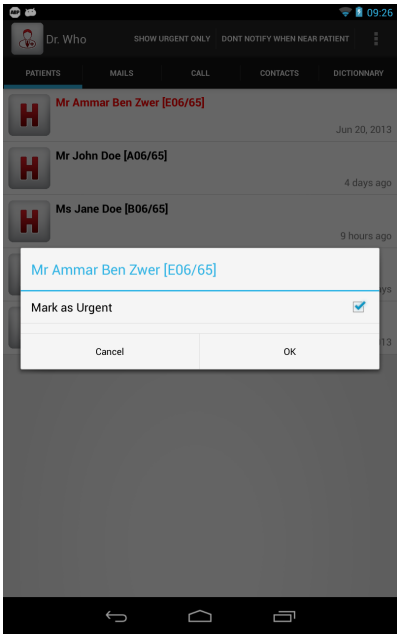


FIGURE 4.5 – Capture écran de l'interface utilisateur affichée lors de la modification de l'état du patient

6. *LoginActivity* invoque *MainActivity*.

Enchaînement alternatif :

– 2.a L'utilisateur est déjà authentifier :

1. La séquence d'authentification est sautée.

– 3.a L'identifiant et / ou le mot de passe comporte des erreurs (champs vide, mot de passe comporte moins des caractères que le minimum) :

1. Affichage d'un message d'erreur.

– 5.a Le service distant retourne une réponse défavorable :

1. Le message d'erreur est extrait de l'interface d'authentification.
2. *LoginActivity* affiche le message d'erreur.

4.3 Afficher La Liste des Patients

4.4 Afficher Le dossier médical du patient

La figure 4.7 présente le comportement de l'application pour réaliser l'opération de l'affichage du dossier médical du patient à travers un dia-

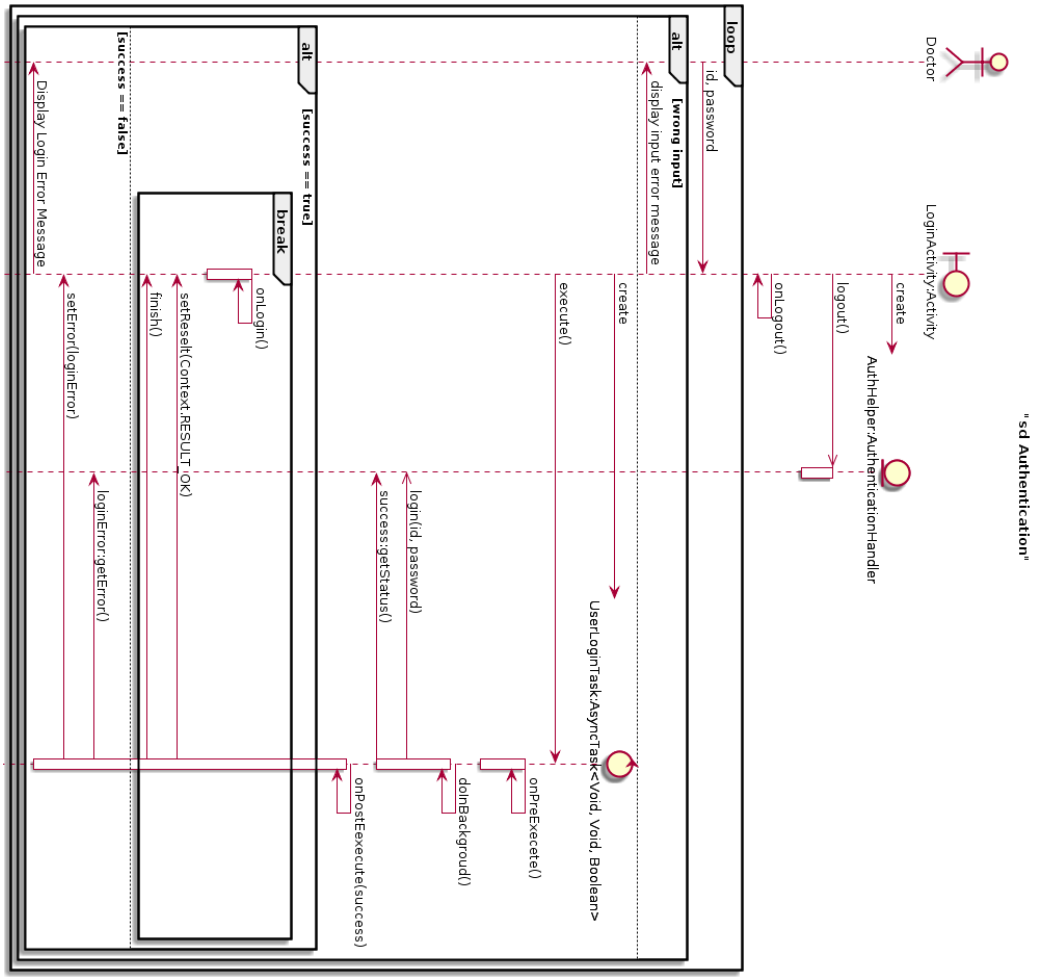


FIGURE 4.6 – Diagramme UML de séquence d'authentification.

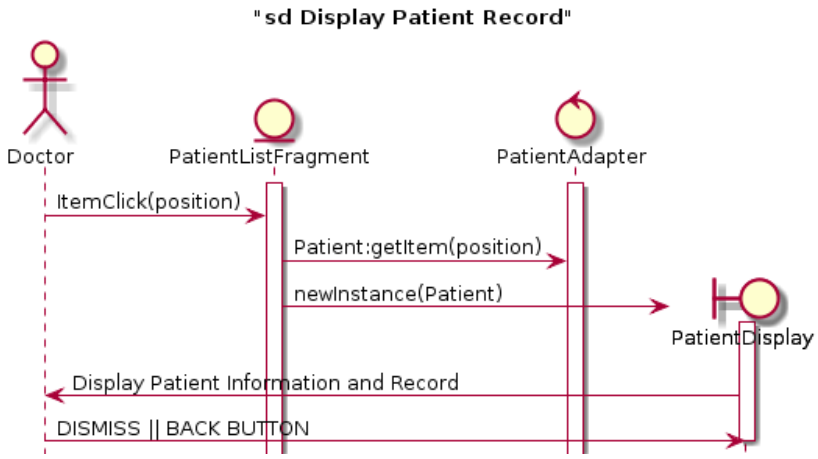


FIGURE 4.7 – Diagramme de séquence UML de l'affichage du dossier médical du patient.

gramme de séquence UML. On peut décrire textuellement le processus par les points suivants :

Acteurs : Docteur.

Pré-condition : Le docteur s'est déjà authentifié et il se trouve sur l'interface de la liste des patients.

Post-condition : Le docteur peut visualiser le dossier médical du patient qu'il à sélectionné.

Scénario nominal :

1. Le docteur clique sur un patient de la liste
2. *PatientListFragment* détecte le clic et demande l'objet de type *Patient* qui correspond au patient sélectionné par la méthode *PatientAdapter.getItem()*.
3. L'objet *Patient* reçu, *PatientListFragment* crée une boite de dialogue de type *PatientDisplay* en lui passant l'objet Patient
4. *PatientDisplay* affiche le dossier médical du patient.
5. Le docteur peut retourner à la liste des patients par un clic en dehors de la boite de dialogue ou par le bouton (retour).

4.5 Modification de l'Etat du patient

La figure 4.8 présente le comportement de l'application pour réaliser l'opération de modification du status du patient (état normal ou état critique) à travers un diagramme de séquence UML. On peut décrire textuellement le processus par les points suivants :

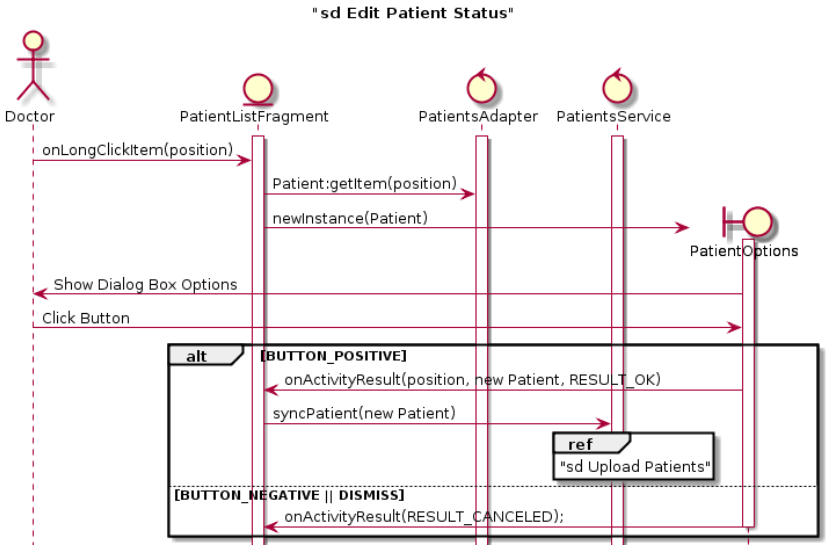


FIGURE 4.8 – Diagramme de séquence UML de modification du status du patient.

Acteurs : Docteur.

Pré-condition : Le docteur s'est déjà authentifié et il se trouve sur l'interface de la liste des patients.

Post-condition : Le docteur à peut modifier le status du patient qu'il a sélectionné.

Scénario nominal :

1. Pour modifier le status d'un patient (ce patient est un cas urgent ou non) le médecin localise le patient dans la liste est fait un clic long sur son entrée.
2. Le *PatientListFragment* détecte le clic long sur l'entrée du patient et demande au *PatientAdapter* un objet patient à partir de sa position dans la liste à travers la méthode *getItem()*.
3. L'objet *Patient* correspondant au patient sélectionner reçu, le *PatientListFragment* crée une boite de dialogue de type *PatientOptions*.
4. La boite de dialogue *PatientOptions* présente à l'utilisateur un checkbox avec deux boutons : bouton (OK) et bouton (Cancel) (figure 4.5).
5. Le médecin effectue le changement selon ses souhaits et clique sur le bouton (OK).
6. *PatientOptions* retourne à la *PatientListFragment* l'objet Patient

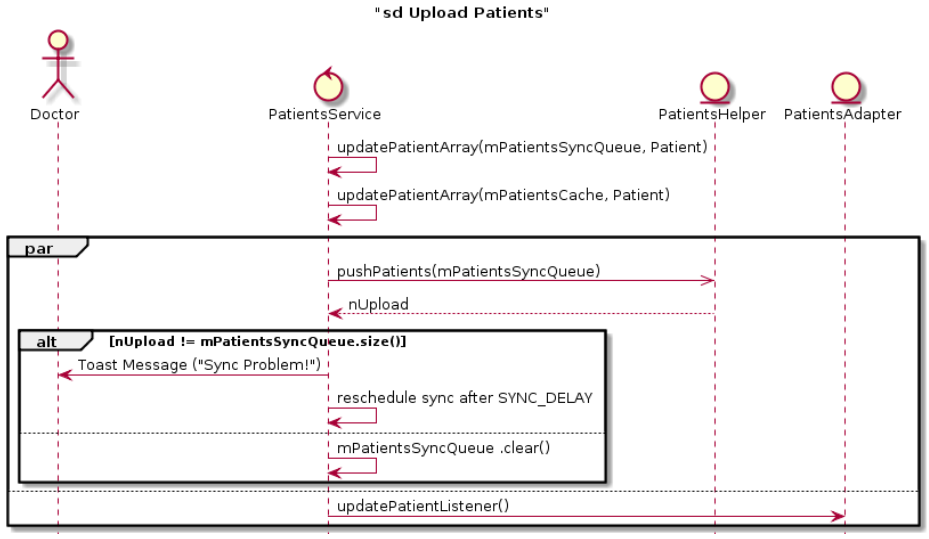


FIGURE 4.9 – Diagramme de séquence UML Mise à jour des patients.

mis à jour ainsi que sa position dans la liste et un drapeau (RESULT_OK).

7. La *PatientListFragment* fait appel au *PatientService* à travers sa méthode *syncPatient()* et lui passe le patient modifier.
8. Le *PatientService* effectue la séquence de mise à jour des patients (voir 4.6).

Enchaînement alternatif : – 5.a Le docteur clique sur le bouton (Cancel) au lieu du bouton (OK).

1. *PatientOptions* retourne à la *PatientListFragment* avec le drapeau (RESULT_CANCELED).

4.6 Mise à jour des patients à partir du terminal

La figure 4.9 présente le comportement de l'application pour réaliser l'opération de mise à jour d'un patient à travers un diagramme de séquence UML. On peut décrire textuellement le processus par les points suivants :

Acteurs : Docteur.

Pré-conditions : Le patient mis à jour par le docteur est disponible au *PatientsService*, Le terminal est connecté au service distant.

Post-conditions : Le patient mis à jour par le docteur est transféreré vers le service distant ou en cas de problème sauvegarder pour une mis à jour ultérieure.

Scénarios nominal :

1. Le *PatientsService* procède à la mise à jour de deux *ArrayList* : *mPatientSyncQueue* qui sert pour le fils d'attente pour les mises à jour et *mPatientCache* qui contient la liste des patients utilisé par notre application.
2. Deux démarches sont exécutées en parallèle :

   2.A Premiére démarche.

   2.A.1 *PatientService* fait appel au *PatientHelper* pour le transfert, en retour il reçoit le nombre des patients mis à jour.

   2.A.2 Le nombre des patients reçu correspond au patients dans la file d'attente alors on vide celle ci.

   2.B Deuxieme démarche.

   2.B.1 Le *PatientAdapter* est notifié par les changements effectués dans la liste des patients courante.

Scénarios alternatifs :

2.A.2.a Le nombre des patients mis à jour tel que retourné à par *PatientsHelper* est différant de celui des patients dans la file d'attente :

2.A.2.a.1 *PatientsService* notifie le docteur de ce problème.

2.A.2.a.2 Puis une autre mise à jour est configurée pour se déclencher aprés un SYNC_DELAY.

4.7 Déploiement et Tests

4.7.1 Déploiement

Pour transférer notre application sur le terminal Android™ on utilise un outil fourni dans l'SDK : Android Debug Bridge (ADB).

ADB [18] est un outil versatile en ligne de commande qui nous permet de communiquer avec une instance d'un émulateur ou un équipement Android™ connecté. C'est un programme de type client-serveur qui inclue 3 composants :

- Un client, qui tourne sur notre machine de développement. On peut invoquer un client depuis une invite de commande par l'envoi d'une commande ADB. D'autre outils Android™ comme le plugin ADT et le Dalvik Debug Monitor Server (DDMS) crée eux aussi des clients ADB.
- Un serveur, qui tourne comme un processus de fond dans notre machine de développement. Le serveur gère les communications entre le client et le démon ADB qui tourne dans une instance d'un émulateur ou un terminal.
- Un démon, qui tourne comme un processus de fond dans chaque instance de l'émulateur ou terminal.

On peut retrouver l'outil ADB dans le dossier $< sdk > /platform - tools/$.

On peut utiliser ADB pour copier une application depuis notre machine de développement et l'installer dans une instance d'un émulateur ou un terminal, pour cela on utilise la commande **install**. Cette commande exige comme paramètre le chemin du fichier .apk que nous voulons installer.

Listing 4.1– Exemple d'utilisation du commande adb install

```
$adb install ~/tunavmedi.apk
```

Notant qu'avec Eclipse équipé du plugin ADT on n'a pas besoin d'utiliser ADB directement pour installer notre application sur l'émulateur ou le terminal. Le plugin ADT s'occupe du packaging et de l'installation de l'application pour nous.

Pour désinstaller une application on utilise le *Package Manager*. On peut envoyer des commandes avec le *Package Manager* pour effectuer des actions et des opérations de recherches sur les paquetages des applications installées dans l'émulateur ou le terminal. Listing 4.2 présente la syntaxe générale de l'outil tandis que le listing 4.3 présente la syntaxe utilisé pour désinstaller notre application.

Listing 4.2– Syntaxe générale de l'utilisation du Package Manager

```
$pm <command>
```

Listing 4.3– Exemple de désinstallation

```
$adb shell pm uninstall com.tunav.tunavmedi
```

4.7.2 Détecteur de bugs : Android Lint

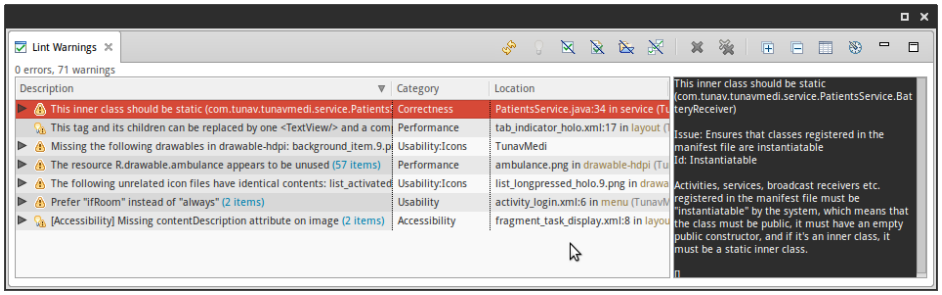


FIGURE 4.10 – Problèmes potentiels dans notre application détectés par Android Lint.

Android™ *Lint* (figure 4.10) est un outil introduit dans la version 16 de ADT qui scanne les code sources des projets Android™ afin d'y détecter des mal-fonctions potentielles.

Quelques exemples de types d'erreurs que cet outil permet de détecter sont :

- Translations manquantes ou inutilisés.
- Les problèmes de performance dans les *Layout*.
- Ressources inutilisées
- Tableau de taille inconsistante (dans le cas ou le tableau est défini dans des configurations différentes).
- Problème d'accessibilités et d'internationalisation.
- Problème d'icônes (Tailles manquantes, doubles, fausse résolution).
- Problème d'usabilité .
- Erreurs dans le *Manifest*.

Dans Eclipse, Android™ Lint est disponible à travers le menu Window → Show View → Other... puis on sélectionne *Lint Warning* dans la fenêtre qui s'affiche (figure 4.11).

4.7.3 *UI/Application Exerciser Monkey*

Monkey [19] est un programme qui tourne sur notre émulateur ou terminal Android™ et qui génère des flux pseudo-aléatoire d'événements utilisateur comme par exemple les clics, les touchés, les gestes, ou encore un nombre d'événements de niveau système. On peut utilisés *Monkey* pour effectuer des tests de stresse sur notre application dans une manière aléatoire et répétitive.

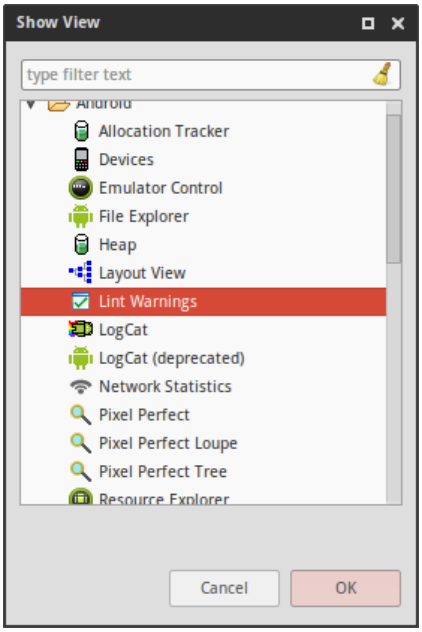


FIGURE 4.11 – Accéder à Android Lint dans Eclipse

L'annexe A montre un exemple de test effectué avec l'outil *Monkey* sur notre application.

Conclusion Générale

L'intégration des technologies au sein des établissements médicaux est, malgré les divers obstacles, une tendance établie et représente un marché juteux pour les sociétés désirant le conquérir, et justifiant la judicieuse idée derrière ce projet. Il en demeure que l'application en elle-même reste limitée. Et particulièrement, le processus de déploiement suggère un minimum d'infrastructures requises. Donc pour offrir l'expérience désirée, une solution alternative de support développée par TUNAV est de rigueur pour combler le manque dans les équipements de l'établissement client ou, dans les cas extrêmes les supplanter. Une stratégie de commercialisation est un besoin évidant. Ce projet peut être qualifié de type « proof of concept », il vise à explorer une idée et vérifier son applicabilité. Une aubaine pour l'application produite qui, en toute honnêteté, n'est pas encore au point et souffre de plusieurs lacunes de conceptions et d'implémentation. Si un produit sérieux dans le même thème est à offrir par TUNAV, des efforts de recherche et de développement sont de mise. En particulier l'intégration de médecins pratiquants dans des hôpitaux au processus de conception et de tests serait critique pour la compétitivité du produit. Cependant, les problèmes techniques pour le développement de cette application ne sont pas les seuls à freiner son adoption. Outre le problème de coûts et l'effort de persuasion requis, c'est un problème d'ordre psychologique auquel il faut faire face. En effet, avec tout concept qui change radicalement des procédures bien établies, une réticence de la part des utilisateurs ciblés, en l'occurrence les médecins, et le staff médical dans un contexte plus large, risque de saboter les tests d´intégrations. Des campagnes de sensibilisation sont à prévoir.

Annexe A

UI/Application Exerciser Monkey

Listing A.1– Utilisation de l'UI/Application Exerciser Monkey

```
$ adb shell monkey -v -p com.tunav.tunavmedi 300
:Monkey: seed=1371512317847 count=300
:AllowPackage: com.tunav.tunavmedi
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: 25.0%
//   6: 15.0%
//   7: 2.0%
//   8: 2.0%
//   9: 1.0%
//   10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=
    android.intent.category.LAUNCHER;launchFlags=0x10200000;
    component=com.tunav.tunavmedi/.activity.MainActivity;end
     // Allowing start of Intent { act=android.intent.action.
        MAIN cat=[android.intent.category.LAUNCHER] cmp=com.
        tunav.tunavmedi/.activity.MainActivity } in package
        com.tunav.tunavmedi
:Sending Touch (ACTION_DOWN): 0:(190.0,120.0)
:Sending Touch (ACTION_UP): 0:(191.60419,128.55092)
:Sending Trackball (ACTION_MOVE): 0:(3.0,-3.0)
```

```
:Sending Touch (ACTION_DOWN): 0:(473.0,23.0)
:Sending Touch (ACTION_UP): 0:(473.1426,23.805832)
:Sending Trackball (ACTION_MOVE): 0:(-5.0,3.0)
:Sending Trackball (ACTION_MOVE): 0:(3.0,0.0)
:Sending Trackball (ACTION_MOVE): 0:(-3.0,-5.0)
:Sending Touch (ACTION_DOWN): 0:(5.0,341.0)
:Sending Touch (ACTION_UP): 0:(4.349031,340.68045)
:Sending Trackball (ACTION_MOVE): 0:(2.0,-3.0)
:Sending Touch (ACTION_DOWN): 0:(611.0,766.0)
    //[calendar_time:2013-06-05 10:14:32.168  system_uptime
       :1098585565]
    // Sending event #100
:Sending Touch (ACTION_UP): 0:(678.4827,701.28955)
:Sending Touch (ACTION_DOWN): 0:(680.0,240.0)
:Sending Touch (ACTION_UP): 0:(669.64984,250.41994)
:Sending Trackball (ACTION_MOVE): 0:(1.0,2.0)
:Sending Trackball (ACTION_MOVE): 0:(-3.0,4.0)
:Sending Touch (ACTION_DOWN): 0:(33.0,339.0)
:Sending Touch (ACTION_UP): 0:(20.433603,303.77527)
:Sending Trackball (ACTION_MOVE): 0:(-4.0,-2.0)
:Sending Touch (ACTION_DOWN): 0:(556.0,636.0)
:Sending Touch (ACTION_UP): 0:(592.86835,561.529)
:Sending Trackball (ACTION_MOVE): 0:(2.0,2.0)
:Sending Touch (ACTION_DOWN): 0:(233.0,837.0)
:Sending Touch (ACTION_UP): 0:(226.95929,825.0325)
:Sending Touch (ACTION_DOWN): 0:(71.0,554.0)
:Sending Touch (ACTION_UP): 0:(73.91967,528.69226)
:Sending Touch (ACTION_DOWN): 0:(30.0,341.0)
:Sending Touch (ACTION_UP): 0:(84.22933,308.51373)
    //[calendar_time:2013-06-05 10:14:32.873  system_uptime
       :1098586237]
    // Sending event #200
    //[calendar_time:2013-06-05 10:14:32.874  system_uptime
       :1098586238]
    // Sending event #200
:Sending Trackball (ACTION_MOVE): 0:(1.0,-2.0)
:Sending Trackball (ACTION_MOVE): 0:(-3.0,0.0)
:Sending Trackball (ACTION_UP): 0:(0.0,0.0)
:Sending Touch (ACTION_DOWN): 0:(782.0,261.0)
:Sending Touch (ACTION_UP): 0:(789.2555,259.95465)
:Sending Touch (ACTION_DOWN): 0:(480.0,1180.0)
:Sending Touch (ACTION_UP): 0:(517.4512,1113.8969)
:Sending Touch (ACTION_DOWN): 0:(775.0,965.0)
:Sending Touch (ACTION_UP): 0:(762.51733,968.3565)
:Sending Trackball (ACTION_MOVE): 0:(4.0,-4.0)
:Sending Trackball (ACTION_MOVE): 0:(1.0,-2.0)
:Sending Trackball (ACTION_MOVE): 0:(-3.0,1.0)
:Sending Touch (ACTION_DOWN): 0:(89.0,1185.0)
:Sending Touch (ACTION_UP): 0:(109.65848,1195.4576)
```

```
:Sending Trackball (ACTION_MOVE): 0:(-4.0,-5.0)
:Sending Touch (ACTION_DOWN): 0:(339.0,280.0)
:Sending Touch (ACTION_UP): 0:(351.69287,274.5476)
:Sending Trackball (ACTION_MOVE): 0:(0.0,-1.0)
Events injected: 300
:Sending rotation degree=0, persist=false
:Dropped: keys=0 pointers=0 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=1986ms (0ms mobile, 1986ms
   wifi, 0ms not connected)
```

Annexe B

Logiciel de gestion de versions Git

Git [20] est un système de gestion de versions et un gestionnaire de code source connu pour sa rapidité. Conçu et développée initialement par Linus Torvalds pour le développement du *Kernel Linux*, Git a depuis été adopté par plusieurs autres projets.

Chaque répertoire de travail de Git est un dépôt complet avec un historique complet et des capacités de suivi de version, il est indépendant d'un accès réseau ou d'un serveur central.

Git est un logiciel libre distribué sous les termes de la licence *GNU General Public License* version 2.

L'usage de Git est très simple, un des workflow simplifié serai comme suit :

- La branche master contient la dernière version de l'application. Malheureusement, cette version contient un bug dans le *MenuItem*
- Pour éviter, pendant qu'on corrige le bug, d'altérer la version courante de l'application de façon nuisible, on crée une branche différente où on peut tester nos modifications tranquillement (listing B.1).
- On peut passer vers la nouvelle branche crée précédemment (listing

FIGURE B.1 – Logo du logiciel de gestion de version Git

B.2).

- On a maintenant réussi à corriger le bug en question, et aussi travailler un peu sur le rapport. On peut avoir un rapport sur les fichiers modifiés pendant ce processus (listing B.3).
- Vérification faite, on valide nos modifications pour les enregistrer dons l'historique des modifications, chaque validation est accompagné par un message détaillant les ajouts effectuées (listing B.4).
- Après on peut appliquer nos changements dans la branche master de notre répertoire Git (listing B.5).

Listing B.1– Git Branch

```
$  PFE git:(master)  git branch bug_MenuItem
```

Listing B.2– Git checkout

```
$  PFE git:(bug_MenuItem)  git checkout master

M   README.md
M   pfe.sublime-project
M   report/2_cadre_general.tex
M   report/4_travail.tex
M   report/acronyms.tex
M   report/appendix.tex
M   report/bibliography.bib
M   report/report.pdf
M   report/report.tex
M   res/ep/Scenarios.ep
M   res/ep/architecture.ep
M   source/tunavmedi/src/com/tunav/tunavmedi/fragment/
   PatientListFragment.java
```

Listing B.3– Git status

```
$  PFE git:(bug_MenuItem)  git status

# On branch bug_MenuItem
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed
   )
#   (use "git checkout -- <file>..." to discard changes in
   working directory)
#
```

```
#    modified:    README.md
#    modified:    pfe.sublime-project
#    modified:    report/2_cadre_general.tex
#    modified:    report/4_travail.tex
#    modified:    report/acronyms.tex
#    modified:    report/appendix.tex
#    modified:    report/bibliography.bib
#    modified:    report/report.pdf
#    modified:    report/report.tex
#    modified:    res/ep/Scenarios.ep
#    modified:    res/ep/architecture.ep
#    modified:    source/tunavmedi/src/com/tunav/tunavmedi/
    fragment/PatientListFragment.java
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
    committed)
#
#    res/architecture-interfaces.png
#    res/ep/scenarioactors.png
#    res/ep/scenarioalltunav.png
#    res/ep/scenariointeractions.png
#    res/ep/scenariootther.png
#    res/sc_display.png
#    res/sc_options.png
#    res/sc_urgent.png
no changes added to commit (use "git␣add" and/or "git␣commit␣
    -a")
```

Listing B.4– Git Commit

```
$  PFE git:(bug_MenuItem)  git commit -a

### TEXT EDITOR START ###
App:

* Fixed Menu Item Bug

Report:

* Worked on the git annexe chapter

# Please enter the commit message for your changes. Lines
    starting
# with '#' will be ignored, and an empty message aborts the
    commit.
# On branch bug_MenuItem
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README.md
#   modified:   pfe.sublime-project
#   modified:   report/2_cadre_general.tex
#   modified:   report/4_travail.tex
#   modified:   report/acronyms.tex
#   modified:   report/appendix.tex
#   modified:   report/bibliography.bib
#   modified:   report/report.pdf
#   modified:   report/report.tex
#   new file:   res/architecture-interfaces.png
#   modified:   res/ep/Scenarios.ep
#   modified:   res/ep/architecture.ep
#   new file:   res/ep/scenarioactors.png
#   new file:   res/ep/scenarioalltunav.png
#   new file:   res/ep/scenariointeractions.png
#   new file:   res/ep/scenariootther.png
#   new file:   res/sc_display.png
#   new file:   res/sc_options.png
#   new file:   res/sc_urgent.png
#   modified:   source/tunavmedi/src/com/tunav/tunavmedi/
    fragment/PatientListFragment.java
#
#### TEXT EDITOR END ###

[bug_MenuItem e951505] App:
 20 files changed, 743 insertions(+), 254 deletions(-)
 create mode 100644 res/architecture-interfaces.png
 create mode 100644 res/ep/scenarioactors.png
 create mode 100644 res/ep/scenarioalltunav.png
 create mode 100644 res/ep/scenariointeractions.png
 create mode 100644 res/ep/scenariootther.png
 create mode 100644 res/sc_display.png
 create mode 100644 res/sc_options.png
 create mode 100644 res/sc_urgent.png
```

Listing B.5– Git merge

```
$  PFE git:(master) git merge bug_MenuItem
Updating 330f830..e951505
Fast-forward
 README.md                                          |    4 +-
 pfe.sublime-project                                |    5 +
 report/2_cadre_general.tex                         |   27 +-
 report/4_travail.tex                               |   94
    +++-
 report/acronyms.tex                                |    3 +-
```

```
report/appendix.tex                                |  159
   ++++++-
report/bibliography.bib                            |   35 +-
report/report.pdf                                  |  Bin
   2495326 -> 3213273 bytes
report/report.tex                                  |    8 +-
res/architecture-interfaces.png                    |  Bin 0
   -> 202897 bytes
res/ep/Scenarios.ep                                |  462
   ++++++++++++++++++
res/ep/architecture.ep                             |  188
   ++++++--
res/ep/scenarioactors.png                          |  Bin 0
   -> 126059 bytes
res/ep/scenarioalltunav.png                        |  Bin 0
   -> 183287 bytes
res/ep/scenariointeractions.png                    |  Bin 0
   -> 155178 bytes
res/ep/scenarioother.png                           |  Bin 0
   -> 186564 bytes
res/sc_display.png                                 |  Bin 0
   -> 136024 bytes
res/sc_options.png                                 |  Bin 0
   -> 84087 bytes
res/sc_urgent.png                                  |  Bin 0
   -> 109422 bytes
.../tunavmedi/fragment/PatientListFragment.java    |   12 +-
20 files changed, 743 insertions(+), 254 deletions(-)
create mode 100644 res/architecture-interfaces.png
create mode 100644 res/ep/scenarioactors.png
create mode 100644 res/ep/scenarioalltunav.png
create mode 100644 res/ep/scenariointeractions.png
create mode 100644 res/ep/scenarioother.png
create mode 100644 res/sc_display.png
create mode 100644 res/sc_options.png
create mode 100644 res/sc_urgent.png
```

Bibliographie

[1] Wikipedia. Observateur (patron de conception), 2013. [accessed May-2013].

[2] Martin Fowler. Passive view, 18 Jul 06. [accessed May-2013].

[3] IDC Worldwide Mobile Phone Tracker.

[4] John Koetsier.

[5] Fiche société.

[6] Brian T. Horowitz. Cisco cius android tablets go to work on san diego hospital private cloud.

[7] David Raths. Real-time healthcare : How one hospital uses cisco's cius to improve patient care.

[8] Palomar Pomerado Health (Press Release). Palomar pomerado health unveils wireless healthcare application for mobile devices.

[9] Cerner. The patient visit...revisited. flayer.

[10] Benjamin Zores. The growth of android in embedded systems. Technical report, The Linux Foundation, 2013.

[11] Reto Meier. *Professional Android 4 Application Development*, chapter 1. Wrox, May 2012.

[12] Wikipedia. Geocoding - Wikipedia, the free encyclopedia, 2013. [accessed Feb-2013].

[13] Emna Hajlawi. Cours de communication spatiale, 2012.

[14] Reto Meier. *Professional Android 4 Application Development*, chapter 13. Wrox, May 2012.

[15] Android API Guides. Location strategies, 2013.

[16] James W. Cooper. *Java Design Patterns : A Tutorial*.

[17] Android API Documentation. Service, 2013.

[18] Android Tools. Android debug bridge, accessed May 2013.

[19] Android Tools. Ui/application exerciser monkey, accessed May 2013.

[20] Wikipedia. Git (software). [accessed June-2013].