République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Seddik Benyahia de Jijel
Faculté des Sciences Exactes et informatique
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme Master en Informatique

Option : ***Réseaux et sécurité***

Thème

Développement d'une application mobile pour le suivi des opérations de secours

Présenté par :
ALI ZOUAGHI Yousra.
HAMIMED Meriem.

Encadré par :
Mr BOUBAKIR Mohammed

Promotion : 2019.

Résumé

Ce mémoire porte sur le développement d'une application visant à faciliter les opérations de secours. L'application développée offre un ensemble de fonctionnalités permettant une intervention rapide et efficace en cas d'urgence. Elle permet principalement à ses utilisateurs de signaler les cas d'urgences et aux secouristes bénévoles géolocalisés dans les environs d'intervenir rapidement avant l'arrivée des secours publics. L'application comporte deux parties, une partie mobile offrant les fonctionnalités principales de secours et une partie web permettant d'effectuer les tâches administratives. L'application a été développée suivant l'approche dénommée processus simplifié en utilisant le langage UML et réalisée en utilisant un ensemble de plateformes et d'outils tels que JEE, Android Studio, Eclipse, etc.

Mots clés : application de secours, application mobile, application web, Android.

Abstract

This dissertation presents the process of developing an application to facilitate rescue operations. This application packs a bunch of features that help fast and reliable intervention in case of emergencies. It mainly allows users to send emergency alerts and geo-located volunteered rescuers in proximity to intervene quickly before the arrival of public emergency. The app consists of two parts, one mobile that offers main rescue features and a web part for administrative features. The app was developed using the named approach process simplified using UML language and executed using a set of platforms and tools such as JEE, Android Studio, Eclipse, etc.

Key words : *rescue application, mobile application, web application, Android.*

⁕ *Remerciements* ⁕

Je tiens à remercier toutes les personnes qui ont contribués au succès de mes études et qui m'ont aidé à l'élaboration de mon mémoire.

Je voudrais dans un premier temps remercier monsieur Mohamed.B mon directeur de mémoire pour sa patience, sa disponibilité et surtout ses judicieux conseils qui ont alimenté ma réflexion.

Je remercie également toute l'équipe pédagogique de l'université de Jijel et les responsables de ma formation qui ont assuré la partie théorique de celle-ci.

Aussi mes parents pour leurs soutiens constants et leurs encouragements.

⁎ Dédicaces ⁎

Je dédie ce modeste travail à :

⁎ *Mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études ;*

⁎ *Mon frère Ziad, mes sœurs Rayane et Loubna et à toute ma famille ;*

⁎ *Tous mes chères amies et collègues Nerimane.A, Nihad.A, Amina.A, Amel.R, Sihem.H, Zineb.L,Salim.K, Salah.F, Youcef.A, Said.H, Lamine.F ;*

⁎ *Mon binôme Meriem.H avec qui j'ai réalisé ce travail ;*

⁎ *Toute la promotion 2019 spécialement "Réseaux et sécurité" ;*

⁎ *Tous mes amis qui me connaissent de près ou de loin.*

Yousra

⁎ Dédicaces ⁎

Je dédie ce modeste travail à :

⁎ *Mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études ;*

⁎ *Mon frère Abderrahime, ma sœurs Wissem et à toute ma famille Hamimed et Bouchebbat ;*

⁎ *Tous mes chères amies et collègues Sihem.H, Amel.R, Ilhem.R, Sofia.A, Imen.B, Rania.B, Sama.C, Amel.K, Nerimene.A, Nihad.A, Amina.A ;*

⁎ *Mon binôme Yousra.A avec qui j'ai réalisé ce travail ;*

⁎ *Toute la promotion 2019 spécialement "Résaeux et sécurité" ;*

⁎ *Tous mes amis qui me connaissent de près ou de loin.*

Meriem

Table des matières

Table des matières i

Liste des tableaux iii

Table des figures v

Liste des abréviations vi

Introduction générale 1

1 Généralités 2

1.1 Introduction . . . 2
1.2 Applications mobiles . . . 2
1.2.1 Définition . . . 2
1.2.2 Type d'application mobiles . . . 2
1.2.3 Les différentes plateformes mobiles . . . 3
1.2.4 Comparaison entre les différents systèmes d'exploitation mobiles . . . 5
1.3 Applications web . . . 7
1.3.1 Définitions . . . 7
1.3.2 Avantages des applications web . . . 8
1.3.3 Architecture client/serveur . . . 8
1.3.4 Langages, protocoles et technologies web . . . 9
1.4 Langage UML . . . 10
1.5 Processus simplifié . . . 11
1.5.1 Processus unifié . . . 11
1.5.2 Méthode agile . . . 12
1.5.3 Les pratiques d'eXtreme Programming et les bases de Scrum . . . 13
1.5.4 La démarche suivie . . . 13
1.6 MVC . . . 14
1.7 Les applications de secours . . . 16
1.8 Conclusion . . . 18

2 Spécification des besoins 19
2.1 Introduction . . . 19
2.2 Description des fonctionnalités . . . 19
2.2.1 Objectif . . . 19
2.2.2 Fonctionnement . . . 19
2.3 Démarche . . . 21
2.4 Spécification des besoins d'après les cas d'utilisations et des IHMs . . . 21
2.4.1 Identification des acteurs . . . 22
2.4.2 Identification des cas d'utilisations . . . 22
2.4.3 Diagramme de cas d'utilisation global . . . 27
2.4.4 Structuration des cas d'utilisation en package . . . 30
2.4.5 Présentation des IHMs . . . 31
2.5 Spécification détaillée . . . 41
2.5.1 Les fiches types et les diagrammes de séquence système . . . 41
2.6 Concluion . . . 47

3 Analyse et conception 48
3.1 Introduction . . . 48
3.2 Diagramme de classes participantes . . . 48
3.2.1 Principe et démarche . . . 49
3.2.2 DCP des cas d'utilisation de notre système . . . 50
3.3 Diagramme de classe global . . . 60
3.4 Diagramme d'état . . . 61
3.5 Diagramme de navigation . . . 62
3.5.1 Diagrammes de navigation de notre système . . . 63
3.6 Conclusion . . . 68

4 Réalisation 69
4.1 Introduction . . . 69
4.2 Choix techniques . . . 69
4.3 Architecture du système . . . 76
4.4 Difficultés rencontrées . . . 77
4.5 Conclusion . . . 77

Conclusion générale 78

Bibliographie 79

Liste des tableaux

1.1 Comparaison entre les systèmes d'exploitation mobiles [8]. 6

2.1 FT de l'UC *Créer compte* 42
2.2 FT de l'UC *S'authentifier* 43
2.3 FT de l'UC *Consulter les types d'alertes* 44
2.4 FT de l'UC *Traiter une demande d'intervention* 44
2.5 FT de l'UC *Afficher notification* 45
2.6 FT de l'UC *Afficher les alertes en cours* 46

Table des figures

1.1 Part du marché des systèmes d'exploitation mobiles en 2017 [9]. 6
1.2 Echange dynamique client/serveur [10]. 8
1.3 Echange dynamique HTTP client/serveur [10]. 9
1.4 Schéma complet du processus simplifié pour la modélisation d'une application web [12]. 14
1.5 Modèle MVC [10]. 15

2.1 Les cas d'utilisation et leurs prolongements dans la démarche [12]. 21
2.2 UCs du *Visiteur*. 23
2.3 UCs du *Membre*. 24
2.4 UCs de *Secouriste*. 25
2.5 UCs de l'*Administrateur*. 26
2.6 UCs de *Webmaster*. 27
2.7 UCs de l'*Utilisateur*. 28
2.8 UCs du *Personnel Administratif*. 29
2.9 UCs de second rang. 30
2.10 Organisation des cas d'utilisation et des acteurs en packages (avec l'outil Enterprise Architect [31]). 31
2.11 IHM principale de *Visiteur*. 32
2.12 IHM d'inscription. 33
2.13 IHM de validation d'email. 33
2.14 IHM d'authentification. 33
2.15 IHM pour signaler l'alerte. 34
2.16 IHM dialogue de confirmation. 34
2.17 IHM du dossier médical. 35
2.18 IHM type groupage. 35
2.19 IHM coodonnées médecin 35
2.20 IHM types maladies 35
2.21 IHM de saisie du poids 35
2.22 IHM annuaire. 36
2.23 IHM d'aide personnelle. 36
2.24 IHM principale du *Membre*. 37

2.25 IHM de notification de l'alerte. 37
2.26 IHM d'intervention 38
2.27 IHM principale du *Membre*. 39
2.28 IHM principale de l'*Administrateur*. 39
2.29 IHM activation/désactivation des comptes. 40
2.30 IHM supression consulation de la liste des utilisateurs. 41
2.31 DSS de l'UC *Créer compte* 42
2.32 DSS de l'UC *S'authentifier* 43
2.33 DSS de l'UC *Consulter les types d'alertes*. 44
2.34 DSS de l'UC *Traiter une demande d'intervention*. 45
2.35 DSS de l'UC *Afficher notification*. 46
2.36 DSS de l'UC *Afficher les alertes en cours*. 47

3.1 Positionnement des diagrammes de classes participantes dans la démarche [12]. 49
3.2 DCP de l'UC *Créer un compte* 51
3.3 DCP de l'UC *Gérer dossier médical* 53
3.4 DCP des UCs *Consulter les types d'alertes*, *Signaler une alerte par type* et *Appeler* 54
3.5 DCP de l'UC *Demander aide personnelle* 55
3.6 DCP de l'UC *Traiter une demande d'intervention* 56
3.7 DCP de l'UC *Visualiser les positions sur la carte* 57
3.8 DCP de l'UC *Consulter l'annuaire* 58
3.9 DCP de l'UC *Afficher alerte* 59
3.10 DCP de l'UC *Afficher notification* 60
3.11 Diagramme de classe global. 61
3.12 Diagramme d'état de l'alerte 62
3.13 Le positionnement de diagramme de navigation dans la démarche [12]. 63
3.14 Début de diagramme de navigation du *Personnel Administratif*. 64
3.15 Diagramme de navigation de *l'Administrateur*. 65
3.16 Diagramme de navigation du *Webmaster*. 66
3.17 Début de diagramme de navigation du *Membre*. 67
3.18 Début de diagramme de navigation du *Secouriste*. 68

4.1 MVC avec JEE [10]. 70
4.2 Inscription à FCM [32]. 75
4.3 Envoie de message FCM [32]. 75
4.4 Architecture global de notre application. 76

Liste des abréviations

OHA : Open Handset Alliance
OS : Operating System
UP : Processus Unifié
PS : Processus simplifié
AM : Agile Modeling
XP : eXtreme Programming
UML : Unified Modeling Language
RUP : Rational Unified Process
URL : Uniform Resource Locator
MVC : Model View Controller
IHM : Interface Homme-Machine
UC : Use Case
DSS : Diagramme de Séquence Système
FT : fiches types
DCP : Diagramme de Classe Participante
JSON : JavaScript Object Notation
HTML : Hyper Text Markup Language
XHTML : Extensible HyperText Markup Language
CSS : Cascading Style Sheets
JS : JavaScript
PHP : Hypertext Preprocessor
ASP : Active Server Pages
HTTP : Hypertext Transfer Protocol
FCM : FireBase Cloud Messaging
AJAX : Asynchronous JavaScript and XML
JEE : Java Entreprise Edition
MYSQL : My Structured Query Language
JSP : Java Server Page
IDE : Integrated Development Environment
RPC : Remote Procedure Call
AVD : Appareil Virtuel Android
API : Application Programming Interface

SAMU : Service d'Aide Médicale Urgente

AFP : Agence France-Presse

Introduction générale

Ces dernières années, les appareils mobiles sont entrés au cœur de la vie quotidienne et devenus les moyens technologiques les plus utilisés avec des fonctionnalités très variées. Cette prédominance du mobile s'accompagne d'une évolution accrue de la consommation des applications mobiles.

Les applications mobiles peuvent contribuer d'une manière considérable à sauver des vies en facilitant les opérations de secours. Il est très fréquent que dans les cas d'urgences tels que les accidents et les urgences médicales, les gens perdent leurs vies en attendant l'arrivée des secours, alors que des personnes capables de les aider se trouvent à proximité. Il existe un nombre d'applications mobiles de secours. Cependant, la plupart de ces applications ne couvrent pas toutes les fonctionnalités de secours. Par exemple, certaines, applications sont uniquement destinées aux secouristes professionnels, alors que d'autres se focalisent sur les urgences médicales telles que les problèmes cardiaques.

Dans ce contexte, nous proposons de développer notre application de secours baptisée *BeSafe*. Cette application vise à faciliter les différentes opérations de secours et permettre une intervention rapide et efficace en cas d'urgences. Cela se fait en intégrant le public et les secouristes bénévoles dans les opérations de secours. *BeSafe* offre à ses utilisateurs un ensemble de fonctionnalités. Elle permet principalement aux public de signaler les cas d'urgences et aux secouristes bénévoles géolocalisés dans les environs d'intervenir avant l'arrivée des secours publics. Notre application comporte deux parties. Une partie mobile pour les fonctionnalités de secours principales et une partie web pour effectuer les tâches administratives. Pour le développement de notre application, nous avons suivi une approche nommée processus simplifié et utilisé le langage UML. Pour sa réalisation, nous avons employé un ensemble de plateformes et d'outils tels que JEE, Android Studio, Eclipse, etc.

Ce manuscrit est divisé en quatre chapitres. Dans le premier chapitre, nous présentons les éléments qui constituent le cadre théorique et métrologique de notre travail. Le deuxième chapitre est consacré à la partie spécification des besoins de notre application. Ces besoins sont spécifiés en utilisant des diagrammes de cas d'utilisation, des maquettes, des diagrammes de séquences système, ainsi que des descriptions textuelles. Le troisième chapitre présente la partie analyse et conception de l'application en utilisant des diagrammes de classes participantes, des diagrammes d'états et des diagrammes de navigations. Enfin, le quatrième et le dernier chapitre est consacré à la présentation des choix techniques de réalisation et l'architecture de l'application.

Chapitre 1

Généralités

1.1 Introduction

Ce chapitre est consacré aux généralités. Nous présentons les applications web et les applications mobiles, leurs avantages et inconvénients. Ensuite, nous présentons le langage de modélisation UML, l'approche de développement suivie ainsi que le modèle MVC. Enfin, nous présentons quelques exemples d'applications de secours.

1.2 Applications mobiles

1.2.1 Définition

Une application mobile est un logiciel applicatif développé pour être installé sur un appareil électronique mobile, comme un smartphone, une tablette ou un baladeur numérique. Elle peut être soit installée directement sur l'appareil dès sa fabrication en usine soit téléchargée depuis un magasin d'applications appelé *application store* telle que *Google Play*, l'*App Store* ou encore le *Windows PhoneStore*. Une partie des applications disponibles sont gratuites tandis que d'autres sont payantes [1].

1.2.2 Type d'application mobiles

Il existe trois types d'applications mobiles : les applications natives, les applications web et les applications hybrides.

1. **Les applications natives**
   Une application native est une application mobile qui est développée spécifiquement pour une seule plateforme, grâce aux outils conçus pour celle-ci. Elle est développée avec un langage spécifique à son système d'exploitation et est distribuée uniquement par l'intermédiaire de son store (*AppStore IOS, PlayStore pour Android*, etc).
   Le développement natif permet d'offrir une ergonomie adaptée à chaque modèle de smartphone et à chaque système d'exploitation. Les applications sont donc plus riches en termes de fonctionnalités, de qualité, de performances et d'ergonomie que les applications web ou les applications hybrides. Cependant, pour développer une application

native pour différent OS (Android et iOS par exemple), il convient de développer deux applications distinctes. Entraînant une perte de temps et une augmentation de coût. De plus, avec la sortie de nouveaux systèmes d'exploitation chaque année, des problèmes de rétrocompatibilité peuvent se poser. L'application doit être maintenue, ce qui augmente les coûts de développement [2].
A titre d'exemples d'applications natives, on peut citer : l'application *Google Play* pour les appareils Android et l'application *App Store* pour les appareils Apple.

2. **Les applications web**
En opposition à une application native, une application web (web application, en anglais) est une application mobile développée avec les outils de développement web tels que : HTML, CSS et JavaScript. L'application web est accessible et exécutable sur tous les smartphones via leur navigateur web. L'avantage de ces applications, c'est le gain du temps et d'argent réalisé grâce à leur développement unique et leur déploiement multiplateformes. On peut tout de même leur reprocher d'être moins performantes que les applications natives. En réalité, tout dépend de la complexité et des fonctionnalités de l'application [2].
A titre d'exemples d'applications web, on peut citer : l'application *Google Drive* et l'application *Google Maps*.

3. **Les applications hybrides**
Les applications hybrides sont des applications qui combinent les éléments d'une application web et les éléments d'une application native. Elles reposent essentiellement sur la solution Cordova/PhoneGap, qui sert de passerelle entre le langage web et le natif. Cette solution nous permet d'utiliser un seul et même outil pour le développement et les langages issus du développement web pour tous les mobiles (iOS, Android et Windows Phone).
Les applications hybrides offre le gain du temps et du coût. Si on prend en compte que la résolution, les performances, et la qualité de ces applications restent légèrement inférieures à celles des applications natives. En effet, l'application hybride peut mal s'adapter au système d'exploitation utilisé par le smartphone de l'utilisateur. De plus, les applications hybrides ne sont accessibles que sur iPhone et Android, et sont parfois refusées sur certains stores [2].
A titre d'exemples d'applications hybrides, on peut citer : l'application *LinkedIn* et *Instagram*.

1.2.3 Les différentes plateformes mobiles

Le marché des téléphones mobiles est aujourd'hui dominé par trois grandes entreprises de technologie smartphone qui sont *Apple*, *Google* et *Microsoft*. Ces entreprises développent respectivement les systèmes d'exploitation *IOS*, *Android* et *Windows Phone*.

Un système d'exploitation mobile est une plateforme logicielle sur laquelle les autres programmes appelés *programmes d'application* peuvent s'exécuter sur des appareils mobiles [3]. Nous allons présenter brièvement les systèmes d'exploitation mobiles les plus connus.

1. **iOS**
   iOS (iPhone OS) est le système d'exploitation développé et utilisé par Apple pour ses terminaux mobiles (iPhone, iPad, etc). Il est dérivé de Mac OS X dont il partage les fondations (le noyau hybride XNU basé sur le micronoyau Mach, les services Unix et Cocoa, etc). IOS comporte quatre couches d'abstraction, similaires à celles de Mac OS X [4] :

   - Une couche *Core OS* est une couche basse du système avec notamment la gestion de la sécurité.
   - Une couche *CoreServices* est une couche haute du système comme le support du format XML, l'accès à une base SQLite, etc.
   - Une couche *Media* est une couche pour la gestion multimédia et les aspects graphiques.
   - Une couche *Cocoa* est une couche de plus haut niveau permettant la gestion des interfaces et gérer le multitâches.

2. **Android**
   Android est un système d'exploitation qui a été à l'origine développé par OHA(Open Handset Alliance), une alliance internationale de compagnies. Aujourd'hui Android appartient à Google qui l'a racheté en 2005. Il est devenu une plateforme open source, basée sur le noyau Linux. En termes d'application, Android a été conçu pour intégrer plusieurs services Google comme *Gmail*, *Google Maps*, *Google Agenda* et *YouTube* [5].

3. **Windows Phone**
   Windows Phone est un système d'exploitation mobile développé par Microsoft pour succéder à Windows Mobile, il est lancé en 2010. Microsoft propose une interface utilisateur dénommée *Modern UI* avec un système de tuiles dynamiques, très différente de ce que l'on peut avoir l'habitude avec iOS ou Android. En mai 2013, Windows Phone devient le troisième système d'exploitation mobile. L'un de ses avantages principaux est qu'il possède la capacité de faire fonctionner des logiciels Microsoft sur son téléphone tels que *Microsoft office* ou *Windows Live* [6].

4. **BlackBerry**
   BlackBerry OS est un système d'exploitation propriétaire pour téléphone mobile de la gamme BlackBerry, conçu par la société canadienne Research In Motion (RIM), son nom a changé pour devenir Blackberry. Il s'agit d'un système multitâche. Le système est surtout connu pour son support natif des courriels, Il comporte aussi une technologie

qui supporte divers types de pièces jointes telles que les fichiers d'extensions .zip, .html, .doc, .dot, .ppt, .pdf, etc [7].

5. **Autres systèmes d'exploitation**
   Il existe d'autres systèmes d'exploitation qui ne sont pas largement connus, et d'autres abandonnés ou leurs développement ont été interrompus, comme par exemple les systèmes *Bada* de Samsung Electronics, *Firefox OS* de Mozilla et *CyanogenMod*, etc.

1.2.4 Comparaison entre les différents systèmes d'exploitation mobiles

Dans cette section nous, allons présenter deux études comparatives entre les systèmes d'exploitation mobiles les plus connus. La première étude à été réalisée par *SocialCompare* [8] et considère les critères suivants : appareils compatibles, dernières versions, mises à jour, dates de sortie, plateformes disponibles et parts de marché, le résultat de cette étude est présenté dans la Table 1.1. D'après cette table, on constate que les dernières mises à jour des deux principaux systèmes *Android* et *iOS* ont été effectuée en 2018 et 2017. Par contre, les deux autres systèmes, *Windows phone* et *BlackBerry*, ont eu leurs développements stoppés et seules des mises à jour de sécurité sont données en 2017. De plus, on signale que *Android* est le seul OS open-source et libre qui est compatible avec les trois plateforme contrairement aux autres systèmes , Il est distribué sous la licence Apache 2.0. La grande force d'Android est donc son ouverture. Par exemple, n'importe quel constructeur ou opérateur de téléphonie mobile peut modifier Android puis l'installer sur un de ses smartphones. Chacun pourra alors, adopter son propre style, installer ses propres applications etc. Cela explique son adoption par un grand nombre de constructeurs et opérateurs mobiles, où le nombre des applications est de plus de 800 000 applications selon les données du tableau.

	Android	IOS	Windows Phone	BlackBerry
Appareils compatibles	HTC, Samsung Galaxy, Motorola.	iPhone, iPad iPod.	Windows Phone, Nokia Lumia 710.	BlackBerry torch, Bold, curve, ...
Dernière version	7.1	10.0.2	10.0.14393.32	10
Date de sortie	4 oct 2016	23 sept 2016	11 oct 2016	30 jav 2013
Open source	Oui	Non	Non	Non
Mis à jour	17 août 2018	28 avr 2017	28 avr 2017	28 avr 2017
Plateformes disponibles				
Windows	Oui	/	Oui	Oui
Mac Os	Oui	Oui	/	/
Linux	Oui	Non	Non	Non
Marché				
Place de marché	Google Play	App Store	Windows Phone Marketplace	BlackBerry Apps
Nombre d'application	800 000+	1 000 000+	>9 000	70 000+

TABLE 1.1 – Comparaison entre les systèmes d'exploitation mobiles [8].

La deuxième étude, réalisée par *ZDNet* [9], s'est focalisée sur l'évolution des parts du marché occupés par un ensemble de systèmes d'exploitation et considère le nombre de téléphones mobiles vendus. Cette étude dont le résultat est présenté dans la Figure 1.1 a été menée en 2017 et comporte des prévisions jusqu'a l'ans 2021.

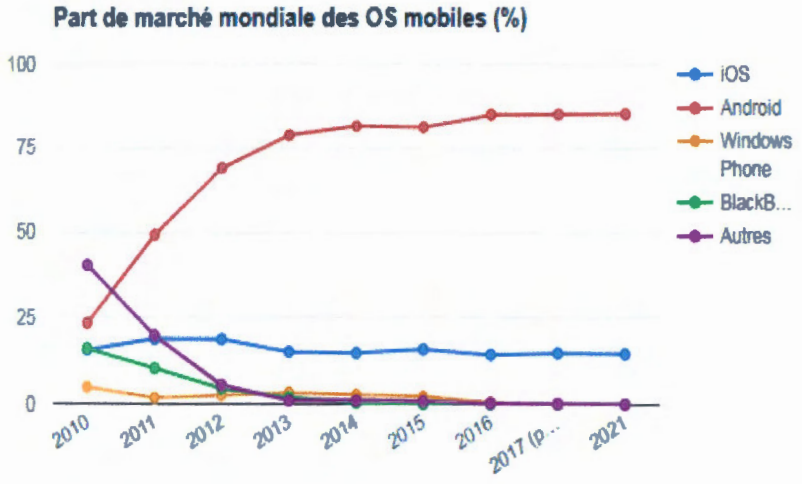


FIGURE 1.1 – Part du marché des systèmes d'exploitation mobiles en 2017 [9].

À partir de ce diagramme, on constate qu'il existe une grande différence entre la part de marché d'*Android* et celle des autres systèmes.
D'autre part, on observe qu'en 2010, *Android* était classé deuxième au monde par rapport aux autres systèmes.
À partir de 2011 à nos jours, on note que la part d'*Android* a augmenté considérablement et qu'elle occupe une place dominante sur le marché mondial d'environ 80%, ce qui est incomparable aux autres systèmes qui ne dépassent pas 20%, d'autre part iOS reste presque stable. Actuellement, le système *Android* bénéfice d'environ 85% des livraisons mondiales de Smartphones. Pour ces raisons, *Android* est largement utilisé comme système de base pour le développement des applications.

1.3 Applications web

1.3.1 Définitions

Avant tout, il ne faut pas confondre entre l'internet et le web. L'internet est le réseau, le support physique de l'information. Pour faire simple, c'est un ensemble de machines, de câbles et d'éléments réseau en tout genre éparpillés sur la surface du globe [10].
Le web (ou world wide web) est un système hypertexte public fonctionnant sur Internet. Il est constitué des pages accessibles par un navigateur web.

1. **Site web** : est un ensemble constitué de pages web (elles-mêmes faites de fichiers HTML, CSS, JavaScript, etc). Lorsqu'on développe puis publie un site web, on met en réalité en ligne du contenu sur internet. On distingue deux types de sites :

   - **Les sites web statiques** : sont des sites dont le contenu est *fixe*, il n'est modifiable que par le propriétaire du site. Ils sont réalisés à l'aide des technologies HTML, CSS et JavaScript uniquement [10].
   - **Les sites web dynamiques** : sont des sites dont le contenu est *dynamique*, ils contiennent des pages web générées au moment de l'arrivée de la requette du client. Lors de l'accès à une page dynamique, le code de la page est analysé sur le serveur web ,en même temp le code HTML résultant est envoyé au navigateur web du client. En plus des langages cités précédemment, ils font intervenir d'autres technologies telles que PHP, ASP et Java EE [10].

2. **Application web** : désigne un logiciel applicatif hébergé sur un serveur et accessible via un navigateur web. Contrairement à un logiciel traditionnel, l'utilisateur d'une application web n'a pas besoin de l'installer sur son ordinateur. Il lui suffit de se connecter à l'application à l'aide de son navigateur [11].

1.3.2 Avantages des applications web

Les applications web offrent les avantages suivants :

- Accès universel depuis n'importe quel type de poste : PC, portables, téléphone mobile et tablette.
- Compatibilité multiplateforme.
- Travailler depuis n'importe quel endroit de la planète.
- Les données sont centralisées.
- Les données sont disponibles 24h sur 24 et 7j sur 7.
- Sécurité des contenus.

1.3.3 Architecture client/serveur

L'architecture client/serveur (présentée dans la Figure 1.2) désigne un mode de communication entre plusieurs composants d'un réseau. Chaque entité est considérée comme un client ou un serveur.

- **Le client :** est un programme qui utilise le service offert par un serveur, il envoie une requête et reçoit la réponse.
- **Le serveur :** est un programme qui offre un service sur le réseau, il accepte des requêtes (un appel de fonction), les traite et renvoie la réponse au demandeur.

FIGURE 1.2 – Echange dynamique client/serveur [10].

La communication qui s'effectue entre le client et le serveur web est régie par des règles bien définies : le protocole *HTTP* (Figure 1.3). Entrons donc un peu plus dans le détail, et regardons de quoi est constitué un échange simple :

1. l'utilisateur saisit une URL dans la barre d'adresses de son navigateur

2. le navigateur envoie alors une requête HTTP au serveur pour lui demander la page correspondante.

3. le serveur reçoit cette requête, l'interprète et génère alors une page web qu'il va renvoyer au client par le biais d'une réponse http.

4. le navigateur reçoit, via cette réponse, la page web finale, qu'il affiche alors à l'utilisateur [10].

FIGURE 1.3 – Echange dynamique HTTP client/serveur [10].

1.3.4 Langages, protocoles et technologies web

Nous présentons dans cette section l'ensemble des protocoles, langages et technologies permettent d'implémenter de l'architecture client/serveur dans le web.

HTTP : (pour HyperText Transfer Protocol) est le protocole de communication communément utilisé pour transférer les ressources du web. HTTPS est la variante sécurisée de ce protocole.

URL : une URL (pour Uniform Resource Locator) pointe sur une ressource. C'est une chaîne de caractères permettant d'indiquer un protocole de communication et un emplacement pour toute ressource du web.

Hyperlien : un hyperlien (ou lien) est un élément dans une ressource associé à une URL. Les hyperliens du web sont orientés : ils permettent d'aller d'une source à une destination.

HTML et **XHTML** : HTML (pour HyperText Markup Language) et XHTML (Extensible HyperText Markup Language) sont les langages informatiques permettant de décrire le contenu d'un document (titres, paragraphes, disposition des images, etc.) et d'y inclure des hyperliens. Un document HTML est un document décrit avec le langage HTML. Les documents HTML sont les ressources les plus consultées du web.

CSS : (pour Cascading Style Sheets) est un mécanisme simple permettant d'ajouter du style (par exemple, des polices, des couleurs, un espacement) à des documents web [17].

JavaScript : (qui est souvent abrégé en *JS*) est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web [18].

PHP : (Hypertext Preprocessor), est un langage de programmation utilisé pour produire des pages web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme

n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet [19].

ASP : (Active Server Pages) est une technologie logicielle destinée à créer des sites web dynamiques. Elle est composée d'une structure d'objets accessibles par deux langages principaux : VBScript et JScript .Cette technologie utilise des langages interprétés.

JEE : (Java Enterprise Edition) est une technologie qui facilite le développement d'applications d'entreprise distribuées en java, JEE repose sur JSE (*Java Standard Edition* et est à destination des plateformes web) et plusieurs autres APIs. JEE propose deux technologies web principale qui sont : *JSP* et *Servlet* .

JSP : (Java Server Pages) est une technologie basée sur Java permettant de créer des pages web dynamiques à l'aide du langage java en l'intégrant au contenu HTML de la page à l'aide de scriptlets.

Servlet : est une classe JAVA côté serveur qui reçoit des requêtes HTTP, et retourne des réponses.

1.4 Langage UML

UML (Unified Modeling Language) est un langage pseudo-formel se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.
UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage [12].
Les diagrammes UML sont répartis en deux grands groupes chacun d'eux est dédié à la représentation des concepts particuliers d'un système logiciel.

1. **Diagrammes structurels**

   - **Diagramme de classes** : Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
   - **Diagramme d'objets** : Il montre les instances des éléments structurels et leurs liens à l'exécution.
   - **Diagramme de packages** : Il montre l'organisation logique du modèle et les relations entre packages.
   - **Diagramme de structure composite** : Il montre l'organisation interne d'un élément statique complexe.
   - **Diagramme de composants** : Il montre des structures complexes, avec leurs interfaces fournies et requises.

- **Diagramme de déploiement** : Il montre le déploiement physique des *artefacts* sur les ressources matérielles.

2. **Diagrammes comportementaux**

- **Diagramme de cas d'utilisation** : Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
- **Diagramme de vue l'ensemble des interactions** : Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
- **Diagramme de séquence** : Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
- **Diagramme de communication** : Il montre la communication entre objets dans le plan au sein d'une interaction.
- **Diagramme de temps** : Il fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
- **Diagramme d'activité** : Il montre l'enchaînement des actions et décisions au sein d'une activité.
- **Diagramme d'états** : Il montre les différents états et transitions possibles des objets d'une classe.

1.5 Processus simplifié

L'approche que nous avons suivie pour le développement de notre application s'inspire du *processus simplifié (PS)* [12], ce processus se situe à mi-chemin entre UP(Unified Process), un cadre général très complet de processus de développement, et les méthodes agiles en vogue actuellement, telles que XP (eXtreme Programming), et Scrum. Il s'inspire également des bonnes pratiques prônées par les tenants de la modélisation agile (Agile Modeling). Le PS propose une démarche suffisante pour construire efficacement des applications web. Cette démarche utilise un sous-ensemble de langage de modélisation UML dans le but de produire le plus rapidement possible une application qui satisfait au mieux ses utilisateurs [12].

1.5.1 Processus unifié

Le processus unifié est un processus de développement logiciel construit sur UML, il regroupe les activités à mener pour transformer les besoins d'un utilisateur en système logiciel. Les caractéristiques essentielles du processus unifié :

- **Le processus unifié utilise le langage UML** (ensemble d'outils et de diagramme).

- **Itératif et incrémental** : si on découpe le projet en itérations de courte durée, ils aident à mieux suivre l'avancement global. À la fin de chaque itération, une partie exécutable du système final est produite, de façon incrémentale.
- **Centré sur l'architecture** : tout système complexe doit être décomposé en parties modulaires afin de garantir une maintenance et une évolution facilitées. Cette architecture (fonctionnelle, logique, matérielle, etc.) doit être modélisée en UML et pas seulement documentée en texte.
- **Conduit par les cas d'utilisation** : le projet est mené en tenant compte des besoins et des exigences des utilisateurs. Les cas d'utilisation du futur système sont identifiés, décrits avec précision et priorisés.
- **Piloté par les risques** : les risques majeurs du projet doivent être identifiés au plus tôt, mais surtout levés le plus rapidement possible. Les mesures à prendre dans ce cadre déterminent l'ordre des itérations.

Le cycle de vie d'un processus unifié : le UP répète un certain nombre de fois une série de cycles. Tout cycle se conclut par la livraison d'une version du produit aux clients et s'articule en 4 phases :

- **Initialisation** : établir une vision globale du projet où on spécifie les besoins et on étudie la faisabilité du projet.
- **Elaboration** : on reprend les éléments de l'initialisation et on développe une architecture de référence, les risques et la plupart des besoins sont identifiés.
- **Construction** : finaliser l'analyse, la conception, l'implémentation et les tests puis transformer l'architecture de référence en produit exécutable tout en veillant à respecter son intégrité.
- **Transition** : livraison du produit au client afin d'effectuer des essais pour détecter d'éventuelles anomalies.

1.5.2 Méthode agile

Méthode agile(AM) est une méthodologie fondée sur la pratique pour la modélisation et la documentation efficaces de systèmes logiciels. À un niveau élevé, la méthode agile est un ensemble de pratiques fondamentales. À un niveau plus détaillé, AM est un ensemble de valeurs, de principes et de pratiques pour la modélisation de logiciels utilisant d'UML, pouvant être appliqués à un projet de développement logiciel de manière efficace et légère [12]..

Cette méthode prend quatre valeurs fondamentales :

- Personnes et interactions plutôt que processus et outils.

- Logiciel fonctionnel plutôt que documentation complète.
- Collaboration avec le client plutôt que négociation de contrat.
- Réagir au changement plutôt que suivre un plan.

1.5.3 Les pratiques d'eXtreme Programming et les bases de Scrum

- **L'eXtreme Programming** : est un ensemble de pratiques qui couvre une grande partie des activités de la réalisation d'un logiciel et qui s'appuie sur les valeurs suivantes : communication, simplicité et feedback, de la programmation proprement dite à la planification du projet, en passant par l'organisation de l'équipe de développement et les échanges avec le client. Elle est bien adaptée pour des projets de taille moyenne où le contexte (besoins des utilisateurs, technologies informatiques) évolue en permanence [12].
- **Les bases de Scrum** : est un processus agile s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but, Le principe de base de Scrum est de focaliser l'équipe de façon itérative sur un ensemble de fonctionnalités à réaliser, dans des itérations de 30 jours, appelées Sprints [12].

1.5.4 La démarche suivie

Les différentes étapes du processus simplifié sont présentées dans la Figure 1.4.
Premièrement, on modélise les besoins au moyen des *cas d'utilisation(UC)* UML. Puis, on les représente de façon plus concrète par une maquette *d'IHM* (Interface Homme-Machine) destinée à faire réagir les futurs utilisateurs. Ensuite, on décrit chaque *UC* textuellement de façon détaillée, ce qui donne lieu à des *diagrammes de séquence système(DSS)* qui représentent graphiquement la chronologie des interactions entre les acteurs et le système vu comme une boîte noire. Par le remplacement du système vu comme une boîte noire par un ensemble choisi d'objets de conception, on obtient *les diagrammes d'interaction*. Donc les liens importants entre *les cas d'utilisation* et les *diagrammes d'interaction* sont *les diagrammes de séquence système*. Après, on identifie *les concepts(entités)* du domaine à partir des *diagrammes de classes participantes (DCP)* qui permettent de faire la jonction entre les UCs, les maquettes et le diagramme de conception logicielle (diagrammes d'interaction et diagrammes de classes). Enfin pour compléter cette démarche, on détaille une exploitation supplémentaire de la maquette, en réalisant des diagrammes dynamiques représentant de manière formelle l'ensemble des chemins possibles entre les principaux écrans(IHMs) proposés à l'utilisateur, qui s'appellent *des diagrammes de navigation* [12].
Nous notons ici que nous avons suivie toutes ces étapes à l'exception du diagramme d'interaction qui présente des détails qui ne sont pas très nécessaires.

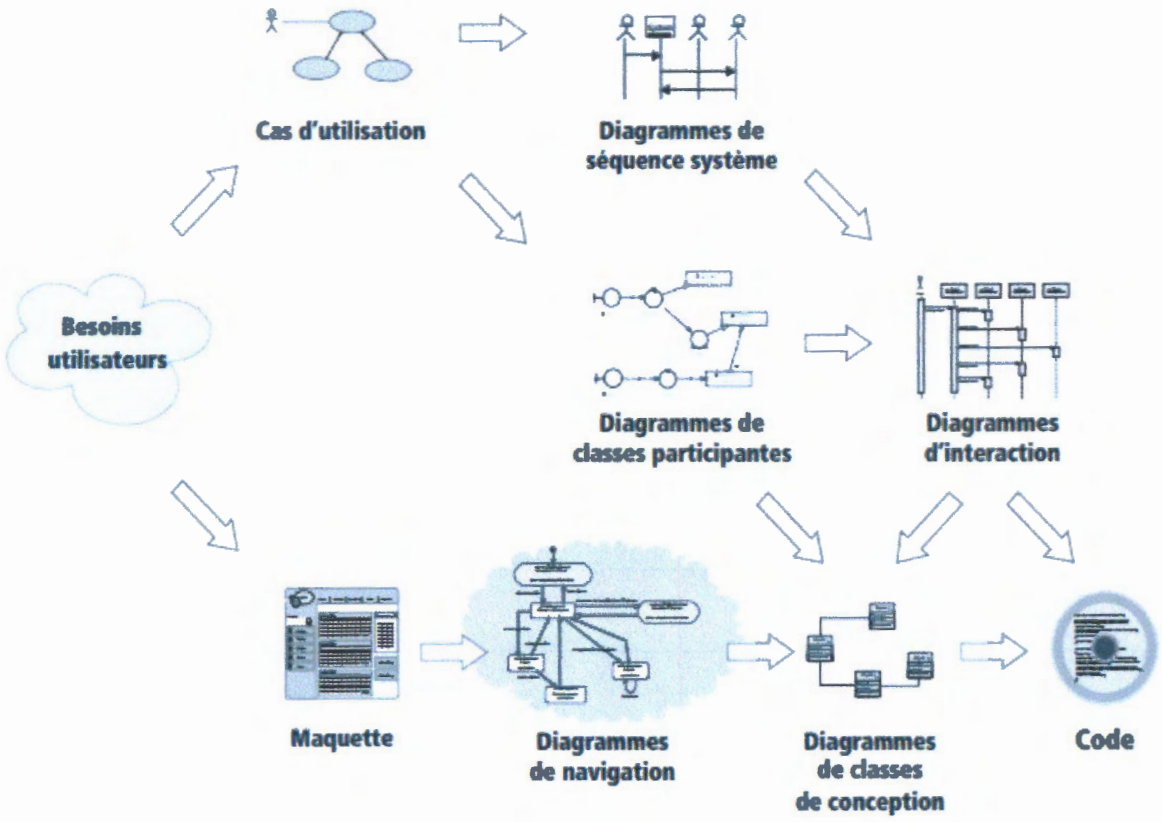


FIGURE 1.4 – Schéma complet du processus simplifié pour la modélisation d'une application web [12].

1.6 MVC

Le processus simplifié propose une démarche compatible avec le modèle d'architecture MVC (Model-View-Controller). En plus, le modèle d'implémentation (celui de la plateforme JEE) utilisé pour la réalisation de notre application respecte aussi ce modèle d'architecture. Pour cela, nous présentons dans cette section le modèle MVC qui consiste à séparer une application en trois groupes de composants principaux : Les modèles, les vues et les contrôleurs (Figure 1.5). Ce modèle permet d'effectuer la séparation des préoccupations. En utilisant ce modèle, les demandes de l'utilisateur sont acheminées vers un contrôleur qui a la responsabilité de fonctionner avec le modèle pour effectuer des actions de l'utilisateur et/ou de récupérer les résultats de requêtes. Le contrôleur choisit la vue à afficher à l'utilisateur et lui fournit toutes les données de modèle dont elle a besoin.

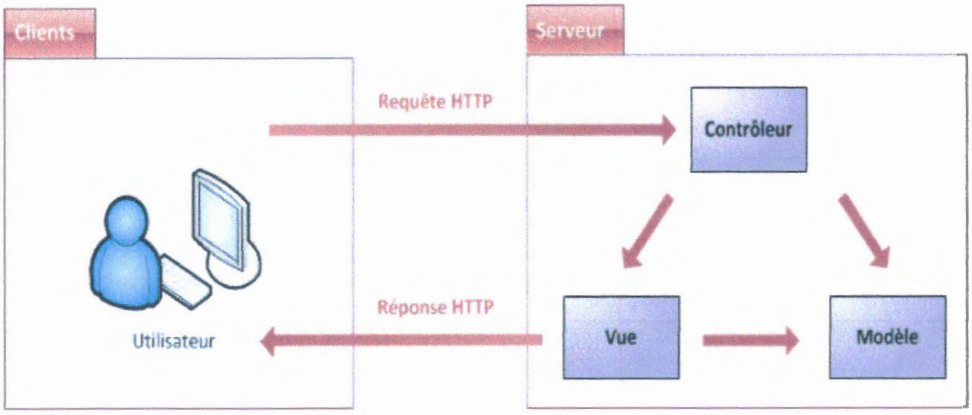


FIGURE 1.5 – Modèle MVC [10].

- **Modèle** : est tout ce qui concerne le traitement, le stockage et la mise à jour des données de l'application. Il représente l'état de l'application, ainsi que la logique métier ou les opérations à effectuer. La logique métier doit être encapsulée dans le modèle, ainsi que toute autre logique d'implémentation pour la persistance de l'état de l'application. En général, les vues fortement typées utilisent des types ViewModel conçus pour contenir les données à afficher sur cette vue. Le contrôleur crée et remplit ces instances de ViewModel à partir du modèle.

- **Vue** : est tout ce qui concerne l'interaction avec l'utilisateur et la présentation des données (mise en forme, affichage).

- **Contrôle** : est tout ce qui concerne le contrôle des actions de l'utilisateur et des données, les contrôleurs sont des composants qui gèrent l'interaction avec l'utilisateur, *fonctionnent avec le modèle et au final, sélectionnent une vue à afficher. Dans* une application MVC, la vue affiche uniquement des informations ; le contrôleur gère les entrées et interactions des utilisateurs, et y répond. Dans le modèle MVC, le contrôleur est le point d'entrée initial. Il est responsable de la sélection des types de modèle à utiliser et de la vue à afficher.

Les avantages du MVC : l'approche MVC apporte de réels avantages :

- Une conception claire et efficace grâce à la séparation des données de la vue et du contrôleur.

- Un gain de temps de maintenance et d'évolution du site.

- Une plus grande souplesse pour organiser le développement du site entre différent développeurs (indépendance des données, de l'affichage (webdesign) et des actions.

1.7 Les applications de secours

L'idée de l'application de secours est venue en 2015 après le décès d'une jeune fille qui s'est étouffée malgré l'intervention des collègues. «Je déjeunais juste à côté, si j'avais été averti plus tôt, j'aurais pu peut-être la sauver avec les premiers gestes», raconte à l'AFP(Agence France-Presse) Ganème Asloune, 34 ans, l'un des deux concepteurs de l'application *Permis de Sauver*[13].
Nous passons de plus en plus de temps sur les réseaux sociaux à communiquer avec des amis au bout du monde, nous parfois ignorons que nos voisins les plus proches peuvent être dans le besoin, un simple signal suffirait pour qu'on puisse se porter au secours d'une victime d'un malaise cardiaque ou de la vieille dame qui s'est évanouie dans l'escalier. L'arrivée des secours public jusqu'au lieu de la victime peut prendre beaucoup de temps ce qui diminue les chances des survies de celle-ci. C'est pour répondre à ces besoins que sont nées les applications de secours. Ces applications permettent d'alerter rapidement les services d'urgence, police et pompier, mais aussi des volontaires, localisés à proximité, qui pourront intervenir immédiatement.
Il existe plusieurs applications de secours, chacune offre un ensemble de fonctionnalités, parmi ces applications :

- **Permis de sauver** : a pour but d'augmenter les chances de survie des victimes grâce à une prise en charge précoce. En pratique; l'application permet aux secouristes de recevoir des notifications push transmises par les seuls services d'urgence possédant la plateforme *Permis de Sauver*. Les secouristes peuvent être sollicités en vue d'intervenir auprès d'une victime dans l'attente de l'arrivée des secours professionnels ou tout simplement en tant que témoin afin d'informer sur l'état de la situation. Les secouristes reçoivent une notification dans laquelle est indiqué le lieu où se trouve la victime, la catégorie de l'incident (arrêt cardiaque, étouffement,etc) et dans la mesure du possible un descriptif de son état physique [14].

- **SAUV Life** : est une application gratuite dédiée à l'urgence vitale, permettant aux citoyens volontaires à proximité d'aller aider une victime. Qu'il soit formé ou non, professionnel de santé ou non, elle est labélisée par les sociétés savantes de médecine d'urgence, par la Croix Rouge. SAUV Life a pour but de :

  - Faire intervenir les citoyens sur la demande des SAMU(Service d'Aide Médicale Urgente) en cas d'arrêt cardiaque ou d'hémorragie grave, l'un des points uniques de cette application est l'interaction entre les SAMU et les citoyens.
  - Permettre de contacter les SAMU rapidement de manière géolocalisée dans le cas d'êtres témoins d'un arrêt cardiaque afin qu'ils puissent faire intervenir les secours organisés mais aussi les autres citoyens à proximité [15].

- **AFPR-Premiers Répondants** : est une application d'urgence de proximité et de premiers secours qui met en lien les services de secours publics avec des secouristes professionnels et/ou bénévoles géolocalisés dans les environs d'un arrêt cardiaque. Un itinéraire minuté de navigation accompagne les secouristes jusqu'au lieu exact de l'intervention, en les guidant vers un défibrillateur si une unité se trouve à proximité.
  AFPR-Premiers Répondants permet deux types d'inscription : soit donneur d'alerte, soit secouriste ou professionnel de santé.
  Les secouristes inscrits sur AFPR-Premiers Répondants peuvent prétendre au statut de premiers répondants. Après validation de leurs renseignements par l'AFPR et acceptation de la charte éthique, les premiers répondants pourront être notifiés sur leur téléphone en cas d'urgence directement par le Centre 18 ou le Centre 15 de leur département.
  Lorsque plusieurs secouristes acceptent une notification d'alerte, ils peuvent être mis en rapport par voie téléphonique et/ou SMS afin, par exemple, de partager des données de localisation. L'application AFPR-Premiers Répondants offre aussi un important volet préventif : apprentissage illustré des Premiers Gestes de secours, offres de formation aux brevets de secourisme dans toute la France, géolocalisation des défibrillateurs, des pharmacies et des centres hospitaliers, numéros utiles.
  L'intervention immédiate d'un secouriste dans les quelques minutes qui séparent l'appel aux secours publics de leur arrivée effective peut se révéler vitale pour une personne en arrêt cardiaque. L'usage d'un défibrillateur sur une victime d'arrêt cardiaque double ses chances de survie [16].

Permis de sauver, *SAUV Life*, *AFPR-Premiers Répondants* sont des applications françaises dédiées seulement aux secouristes, elles sont installées dans les locaux des SAMU qui leurs permet de déclencher des alertes, elles ne fonctionnent pas en Algérie.
Sauf qu'*AFPR-Premiers Répondants* est dédiée aux secouristes et aux utilisateurs simples, elle permet d'appeler les services de secours. Toutes les applications de secours existantes sont faites à l'étranger et dédiées seulement pour eux et celles qui fonctionnent chez nous ne fonctionnent pas correctement, elles donnent des services très limités. Les applications de secours existantes permettent un seul moyen pour demander l'aide qui est l'appel, dans le cas où la personne qui demande l'aide n'a aucune information sur son adresse exacte (sa position), cela rend l'intervention difficile.
Dans ce contexte nous proposant notre application baptisée *BeSafe*, destinée aux services de pompiers. En plus des fonctionnalitées de base telle que : l'intégration des secouristes bénévoles dans la chaîne de secours, *BeSafe* permet de :

- Faciliter le déclenchement des alertes où l'utilisateur peut choisir directement le type d'alerte qu'il veut signaler, ce qui déclenche une alerte en précisant la position du déclencheur (la personne qui l'a envoyée).

- Permet aux utilisateurs de demander l'aide personnelle.
- Permet aux utilisateurs d'ajouter des contacts d'urgences pour les appeler facilement en cas d'urgence.
- Contrairement aux autres applications de secours, *BeSafe* est destinée pour fonctionner en algérie.

1.8 Conclusion

Nous avons consacré une partie de notre travail au cadre théorique et méthodologique de notre travail. Nous avons en premier temps exploré le domaine des applications mobiles et web. Cela, nous a permis de prendre certain choix tels que le choix d'Android comme plateforme mobile et le JEE come plateforme de développement web. Nous nous sommes aussi intéressés à l'approche de développent (processus simplifié), le langage de modélisation (UML) ainsi que le modèle architecturel utilisé (MVC). Enfin, nous avons aussi étudié un ensemble d'applications de secours.

Chapitre 2

Spécification des besoins

2.1 Introduction

La spécification des besoins constitue la phase de départ de toute application à développer. Dans ce chapitre nous présentons d'abord les fonctionnalités de notre système. Ensuite, en suivant le processus simplifié nous exprimons les besoins utilisateur sous forme de diagrammes de cas d'utilisation et présenter les IHMs de ces cas d'utilisation. Enfin, nous décrivons les différents cas d'utilisations de façon détaillée par des fiches types et des diagrammes de séquences système.

2.2 Description des fonctionnalités

2.2.1 Objectif

L'objectif de l'application BeSafe est de faciliter les différentes opérations de secours dirigées par les services de la protection civile. BeSafe offre à ses utilisateurs un ensemble de fonctionnalités permettant une intervention rapide et efficace en cas d'urgence. Elle permet, principalement, à des utilisateurs de signaler des accidents et à des personnes qui ayant des problèmes de santé d'appeler une aide médicale urgente. Les informations fournies par les utilisateurs, notamment la localisation géographique permettent aux services spécialisés d'intervenir de manière efficace. L'application offre aussi la possibilité d'informer les secouristes volontaires qualifiés qui se trouvent à proximité de l'accident pour une prise en charge rapide des victimes dans l'attente de l'arrivée des secours publics.

2.2.2 Fonctionnement

Dès qu'une alerte est signalée, elle sera envoyée à un administrateur, qui vas la propager aux secouristes. Le secouriste reçoit une notification sur son téléphone lui indiquant qu'un incident a eu lieu à proximité, cette notification spécifie un ensemble d'informations comme par exemple : le type de l'incident et sa localisation. Si le secouriste accepte d'intervenir, il sera guidé jusqu'au lieu de l'incident, une fois sur place il peut intervenir pour effectuer les premier secours recommandés avant l'arrivée des secours publics, dès l'arrivée de ces dernier

la mission de ce secouriste se termine.
Avant de présenter avec plus de détails la description des fonctionnalités de notre application, nous présentons quelques concepts de base.

- **Incident** : désigne tout type d'incidents nécessitant l'intervention des secours publics (accidents corporels, domestiques, de catastrophes naturelles, accident de route, incendie, attentats, agressions, etc).
- **Alerte** : désigne le message par lequel un utilisateur sollicite l'intervention des secouristes à proximité, les types d'alertes sont : *accident de route, crise et maladie, incendie, enlèvement et aide personnelle.*
- **Secours publics** : désigne les secours compétents en fonction des zones géographiques et de leurs domaines (police, pompiers).
- **Intervention** : prodiguer les premiers gestes de secours avant l'arrivée des secours publics.
- **Dossier médical** : contient des informations concernant l'utilisateur : groupe sanguin, maladies, poids, les coordonnées du médecin traitant.

BeSafe comporte deux parties : une partie web et une partie mobile :

1. L'application BeSafe mobile permet aux personnes en difficulté (problème de santé, accident, incendie, etc) de déclencher des alertes qui sont envoyées à tous ceux qui sont susceptibles de les secourir à proximité avant l'arrivée des secours publics, ce qui vise à augmenter les chances de survie des victimes en rendant l'intervention plus rapide. *Celui qui possède BeSafe peut :*
   - Signaler des alertes par type via le terminal mobile (pour soi-même ou un tiers accidenté) qui permet d'avertir des secouristes présents dans la zone géographique de l'alerte et de solliciter leurs interventions.
   - Localiser les hôpitaux, les pharmacies, les postes polices qui se situent à proximité.
   - Créer le dossier médical et ajouter des contacts d'urgence pour faciliter l'intervention en cas de demande d'aide personnelle.
   - Appeler les services de secours et les contacts d'urgence.

   En plus des services cités précédemment le secouriste peut :
   - Recevoir des notifications d'alertes catégorisées avec affichage de type et de lieu de l'incident.
   - Accepter ou ne pas accepter une alerte.

- Sera guidé jusqu'au lieu de l'incident par un itinéraire qui sera tracé sur la carte.
- Sera *notifié de la fin de traitement de l'alerte qu'il a accepté.*

2. L'application web BeSafe est une application sur PC dont l'objectif est d'offrir des tâches administratives permettant une gestion complète des alertes. L'application offre à l'administrateur les fonctionnalités suivantes :

   - Il sera averti par des nouvelles alertes.
   - Il décide soit de propager les alertes aux secouristes soit de les ignorer.
   - Après le traitement de l'évènement, il informe les secouristes que l'alerte en cours a été traitée.

2.3 Démarche

Dans un premier temps, les besoins vont être modélisés au moyen des cas d'utilisation et seront représentés de façon plus concrète par une maquette d'IHM. Ensuite à partir de la description détaillée des UCs ils seront représentés sous forme de DSS.
La Figure suivante montre les étapes suivies par cette démarche.

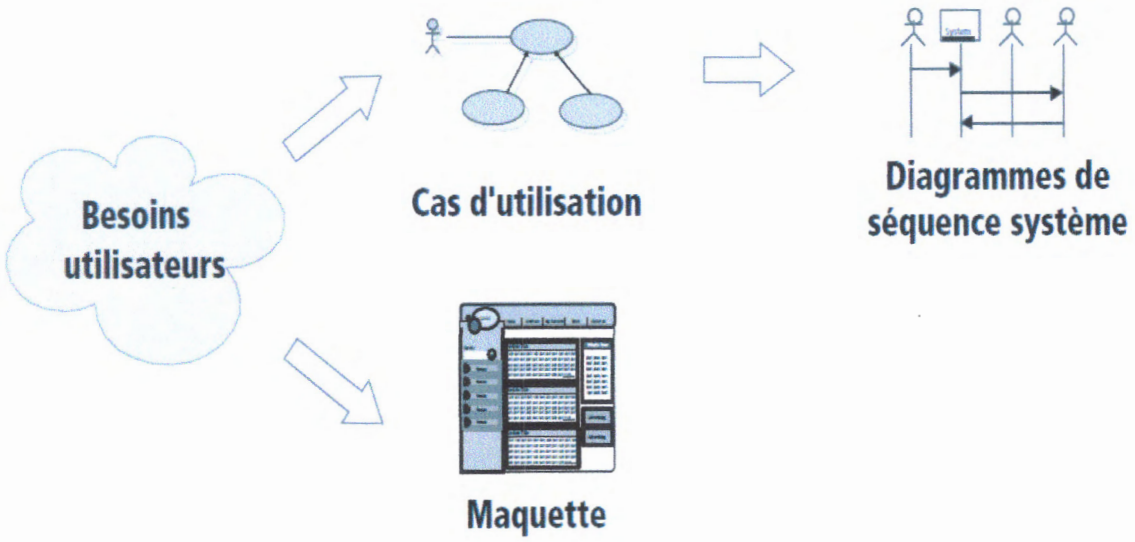


FIGURE 2.1 – Les cas d'utilisation et leurs prolongements dans la démarche [12].

2.4 Spécification des besoins d'après les cas d'utilisations et des IHMs

Dans cette section nous allons détailler les deux premières étapes de la démarche citée précédemment, qui se résume dans les étapes suivantes :

- Identification des différents acteurs, un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié [12].
- Identification des cas d'utilisation, qui représentent un ensemble de séquences d'actions réalisées par le système et produisent un résultat observable intéressant pour un acteur particulier [12].
- Ajouter des relations entre UC et finaliser les diagrammes de cas d'utilisation qui montrent les interactions fonctionnelles entre les acteurs et le système [12].
- Structurer les cas d'utilisation et les regrouper en package, qui est un mécanisme général de regroupement d'éléments en UML [12].
- Représenter les maquettes d'IHM qui nous donnent une vue sur la future interface de l'application [12].

2.4.1 Identification des acteurs

Les acteurs de notre système sont les suivants :

Visiteur : utilisateur qui n'est pas inscrit(ne possède pas de compte) et qui peut accéder à certaines fonctionnalités de l'application comme par exemple : effectuer un appel d'urgence, et créer un compte pour devenir *Membre*.

Membre : désigne tout utilisateur inscrit (qui possède un compte) qui peut accéder à l'application afin de signaler des alertes.

Secouriste : désigne des secouristes qualifiés inscrits sur l'application (pompier, professionnel de santé, sauveteur professionnel) qui peuvent intervenir pour une opération de secours. Tous les *Secouristes* sont des *Membres*.

Administrateur : *s'occupe du suivi des alertes et des opérations de secours.*

Webmaster : s'occupe des opérations de la maintenance et de la gestion technique.

2.4.2 Identification des cas d'utilisations

Pour chaque acteur identifié précédemment, nous cherchons les différentes intentions selon lesquelles il utilise le système.

1. **UCs du *Visiteur***
   La Figure ci-dessous présente les deux UCs du *Visiteur* :
   - *Effectuer un appel d'urgence* qui consiste à appeler les services de secours.

- *Créer un compte* permet au *Visiteur* de créer un compte pour devenir *Membre*.

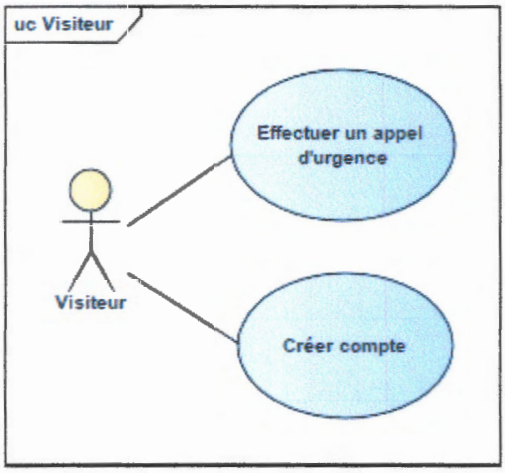


FIGURE 2.2 – UCs du *Visiteur*.

2. **UCs du *Membre***

La Figure ci-dessous présente les UCs du *Membre* :

- *Visualiser les services* qui permet de visualiser sur la carte les positions des hôpitaux, pharmacies et des postes polices localisés à proximité.
- *Demander l'aide personnelle* en signalant une alerte de type aide personnelle, ou en effectuant un appel direct.
- *Consulter les types d'alertes* qui lui permet de signaler une alerte par type ou d'effectuer un appel selon le type d'alerte.
- *Consulter l'annuaire* qui contient quatre contactes d'urgence (police, pompier, ambulance, gendarmerie) permettant de donner lieu à des appels.
- *Gérer l'espace personnel* permet la gestion du compte plus la gestion du dossier médical.

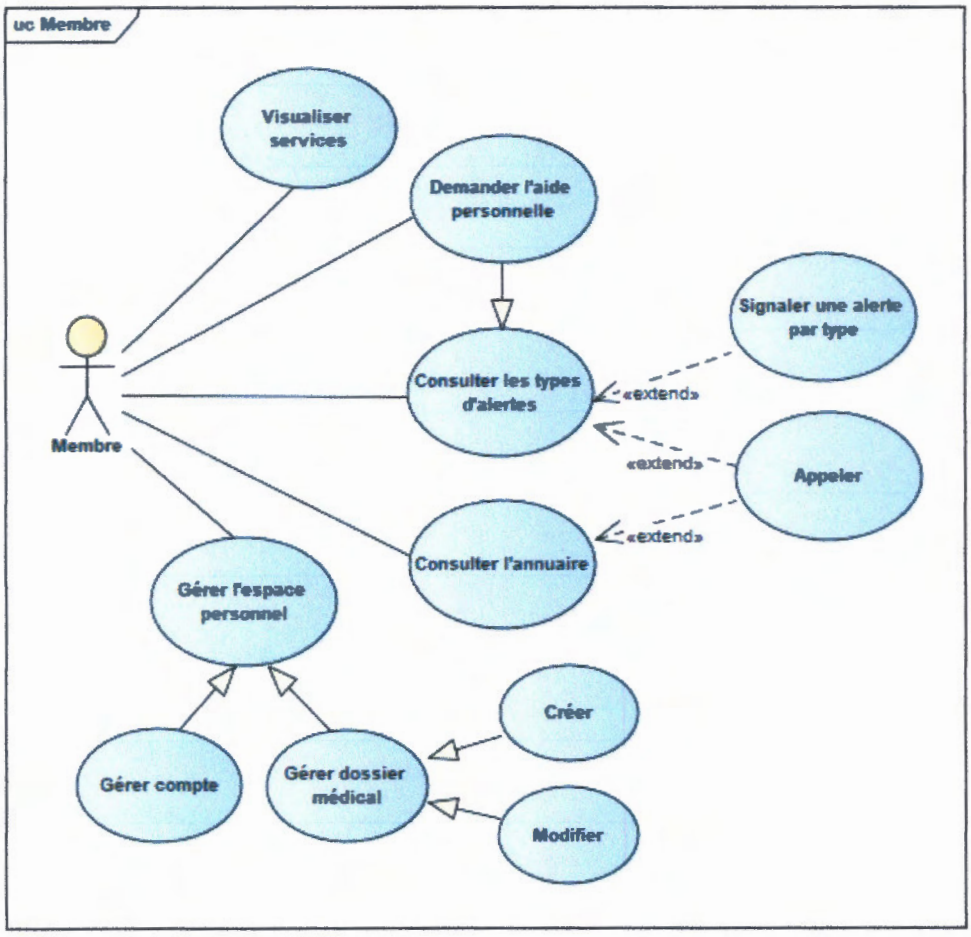


FIGURE 2.3 – UCs du *Membre*.

3. **UCs de *Secouriste***

   La Figure ci-dessous présente les UCs du *Secouriste*. En plus des UCs du *Membre* cités précédemment, le *Secouriste* peut aussi :

   - *Consulter la liste des alertes acceptées* lui permet d'afficher les alertes qu'il a acceptées.
   - *Répondre à l'alerte* lui permet *d'accepter* ou *d'ignorer* l'alerte qu'il a reçue.
   - *Traiter une demande d'intervention* après avoir accepté de donner l'aide, il peut *confirmer son arrivée* pour intervenir comme il peut effectuer un *appel*.

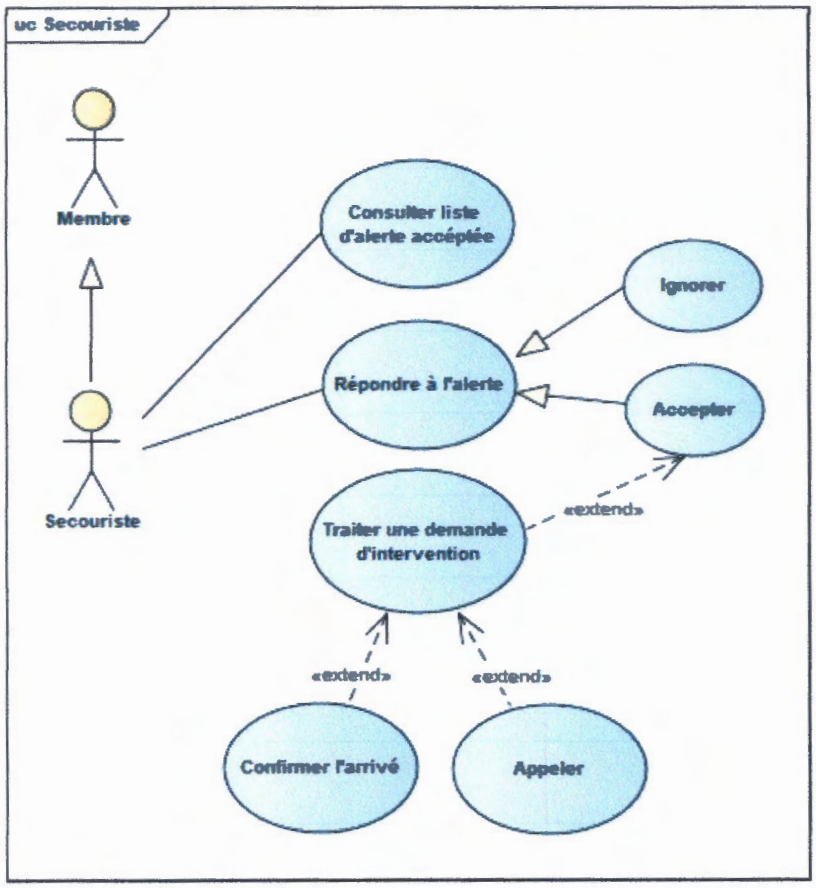


FIGURE 2.4 – UCs de *Secouriste*.

4. **UCs de l'*Administrateur***

Les UCs de l'*Administrateur* (présentés dans la Figure ci-dessous) sont :

- *Consulter la liste des Secouristes* selon leurs types (pompier, professionnel de santé, sauveteur professionnel).
- *Consulter les statistiques* telles que le nombre d'intervention des alertes.
- *Consulter l'historique* des actions.
- *Afficher les notifications*, lorsque l'*Administrateur* reçoit des alertes du *Membre*, il peut soit les *ignorer* ou les *valider* ou *afficher leurs détails*. La validation d'une alerte implique sa propagation aux *Secouristes*.
- *Afficher alerte* lui permet d'afficher les alertes selon leurs états : les alertes *ignorées*, les alertes *traitées*, les alertes *en cours* (qui sont validées et ne sont pas encore traitées), *afficher les détails* d'une alerte. Il peut aussi *modifier l'état* de l'alerte et le rend *traitée*.

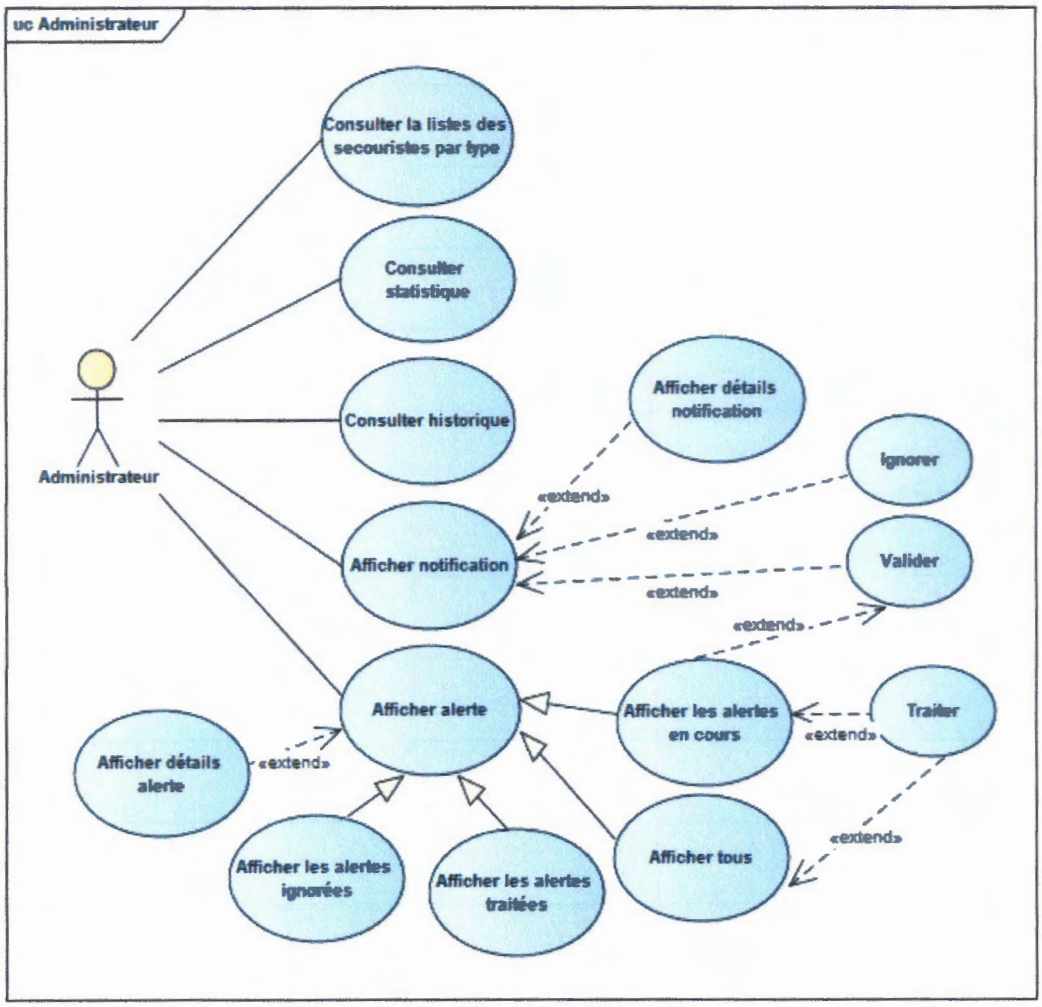


FIGURE 2.5 – UCs de l'*Administrateur*.

5. **UCs de *Webmaster***

   Les UCs de *Webmaster* sont illustrés dans la Figure ci-dessous. Le *Webmaster* peut :

   - *Gérer les comptes des utilisateurs* qui lui permet d'activer/désactiver, supprimer des comptes et consulter la liste des comptes.

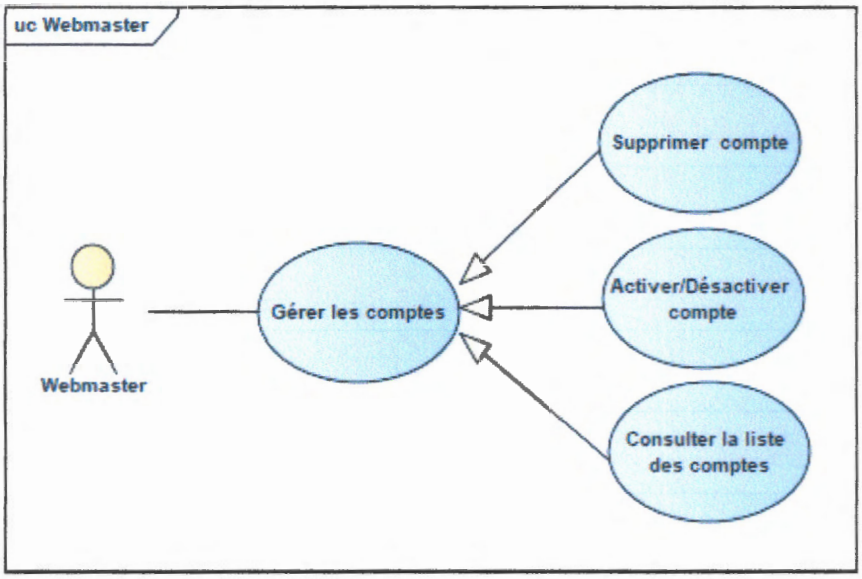


FIGURE 2.6 – UCs de *Webmaster*.

2.4.3 Diagramme de cas d'utilisation global

Nous allons maintenant ajouter les relations entre les UCs cités précédemment et les regrouper dans les trois diagrammes de cas d'utilisation suivants : *UCs des Utilisateurs*, *UCs du Personnel Administratif*, *UCs de second rang.*

1. **UCs des utilisateurs**

   Nous avons créé un acteur généralisé nommé *Utilisateur*, dont *Membre* et *Visiteur* seront des spécialisations. Après simplification et regroupement des UCs des utilisateurs nous obtiendrons le diagramme ci-dessous :

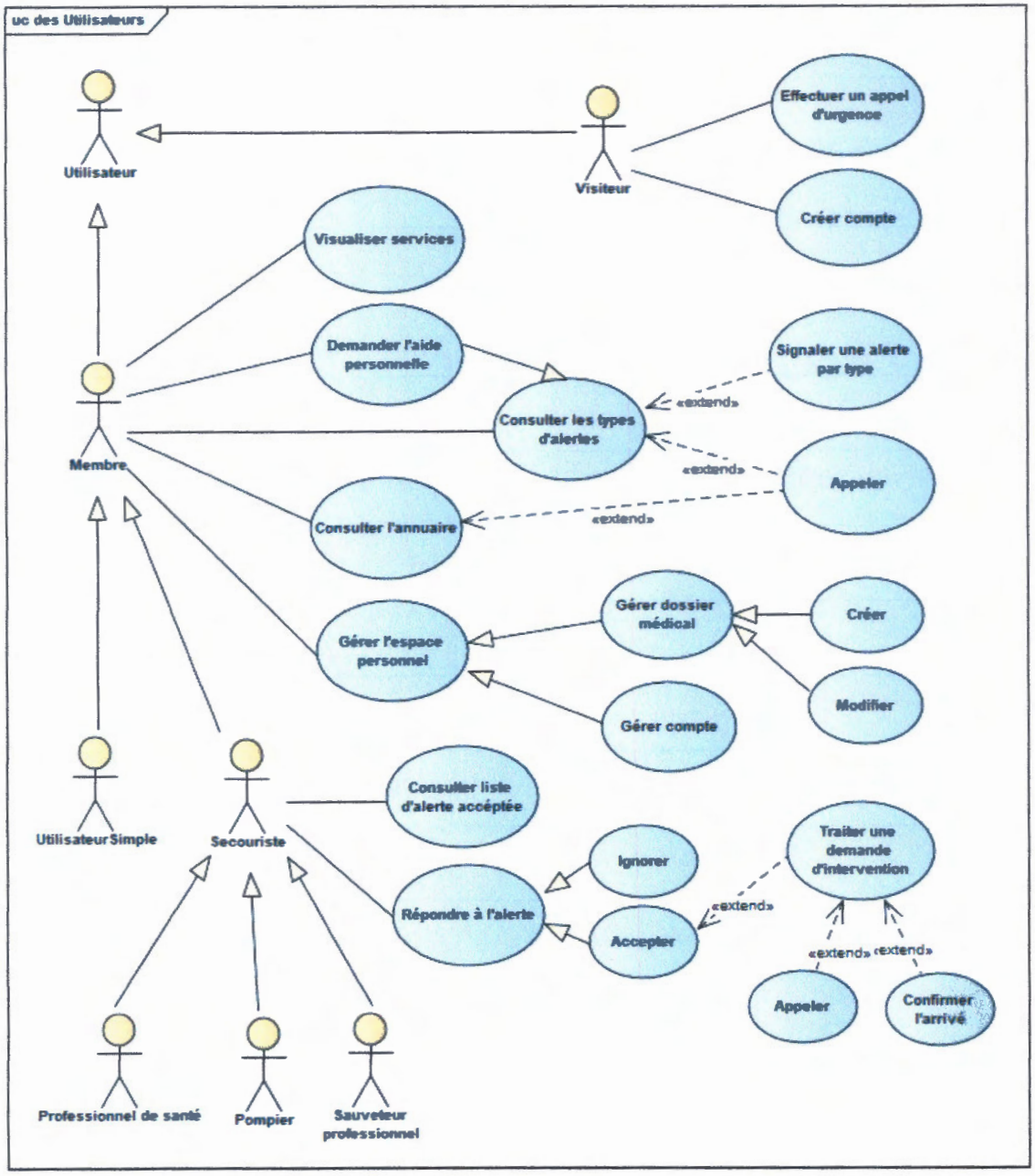


FIGURE 2.7 – UCs de l'*Utilisateur*.

2. **UCs du Personnel Administratif**

   Nous avons ajouté un acteur nommé *Personnel Administratif* qui est une généralisation des acteurs *Administrateur* et *Webmaster*, nous obtiendrons le diagramme suivant :

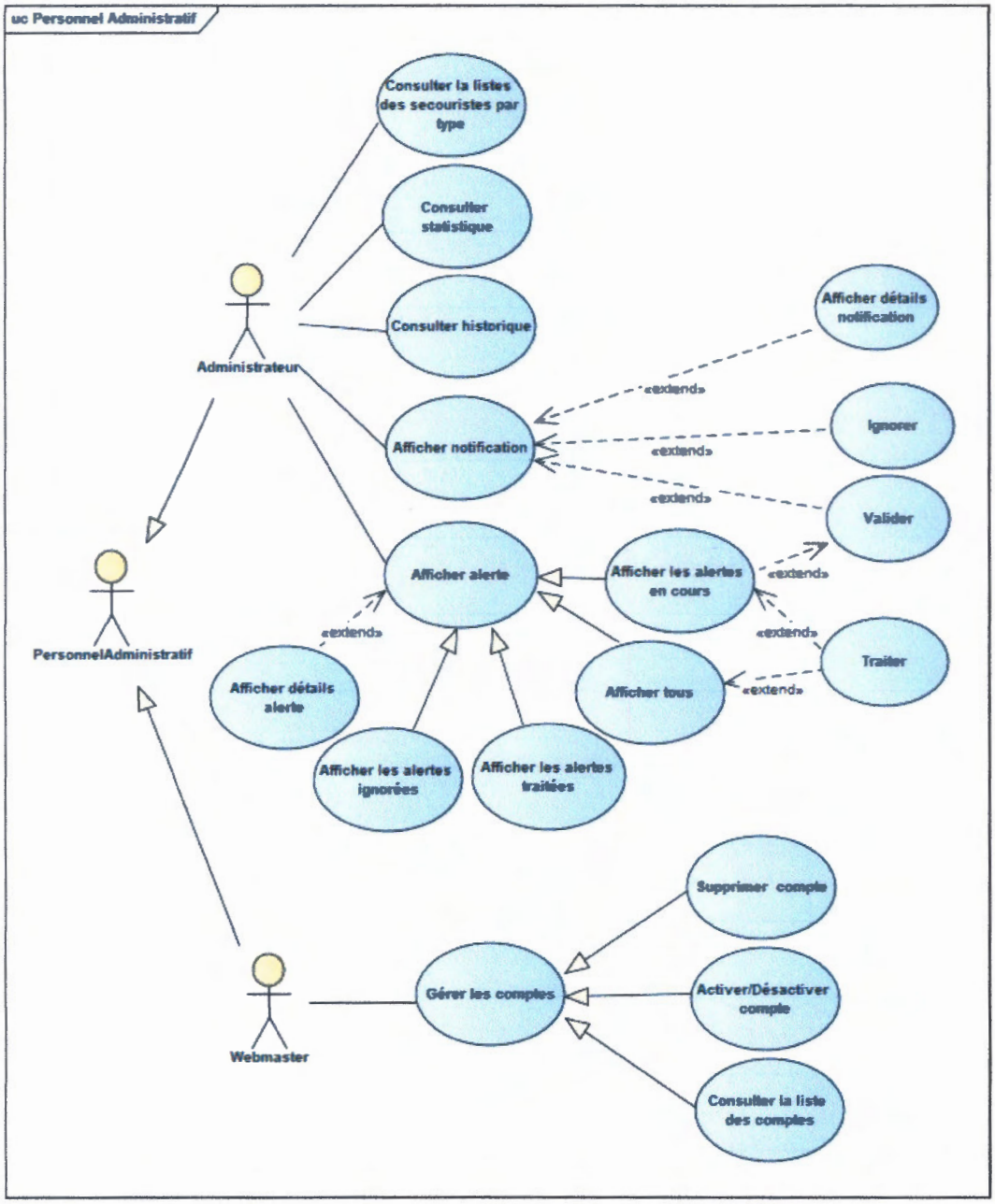


FIGURE 2.8 – UCs du *Personnel Administratif.*

3. **UCs de second rang**

   Le diagramme ci-dessous présente les UCs de second rang. Un UC de second rang est un UC qui ne représente pas l'objectif principal d'un acteur. Le cas d'utilisation *s'authentifier* devra être réalisé afin de permettre au *Membre* et au *Personnel Administratif* d'exécuter leurs propres cas d'utilisation majeurs. Nous qualifierons donc ce petit cas d'utilisation stéréotypé par le mot *fragment* pour indiquer qu'un cas d'utilisation n'est qu'un fragment factorisé d'autres cas d'utilisation, il ne représente pas un objectif à part entière de ces acteurs, mais plutôt un objectif de niveau intermédiaire [12].

Le cas d'utilisation *Consulter l'aide* ne s'agit pas d'un cas d'utilisation majeur. Nous le qualifions par le Stéréotype *secondaire* pour indiquer qu'un cas d'utilisation est moins important que les autres [12].

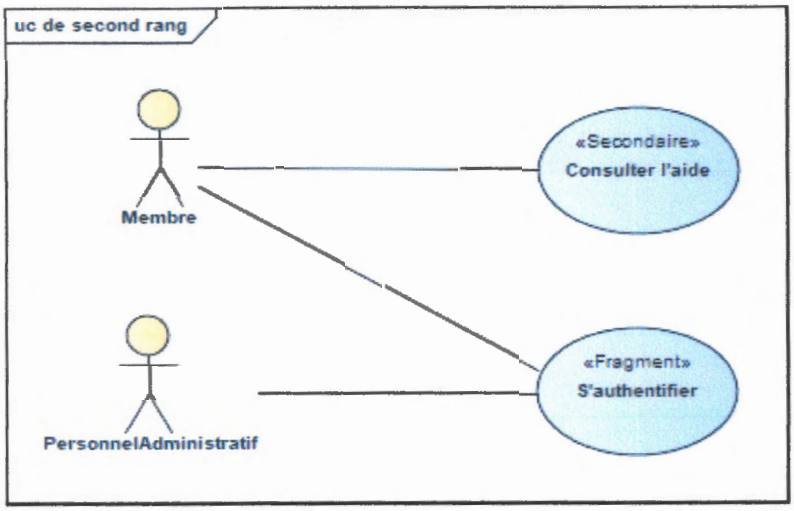


FIGURE 2.9 – UCs de second rang.

2.4.4 Structuration des cas d'utilisation en package

Pour améliorer notre modèle, nous allons organiser les cas d'utilisation et les regrouper en trois packages présentés dans la figure ci-dessous :

1. **Package des Utilisateurs** : contient les utilisateurs et leurs UCs.
2. **Package d'Administration** : contient les *Administrateurs* et leurs UCs.
3. **Package de second rang** : nous préférons d'isoler les UCs de second rang (*s'authentifier* et *consulter l'aide*) dans un package à part.

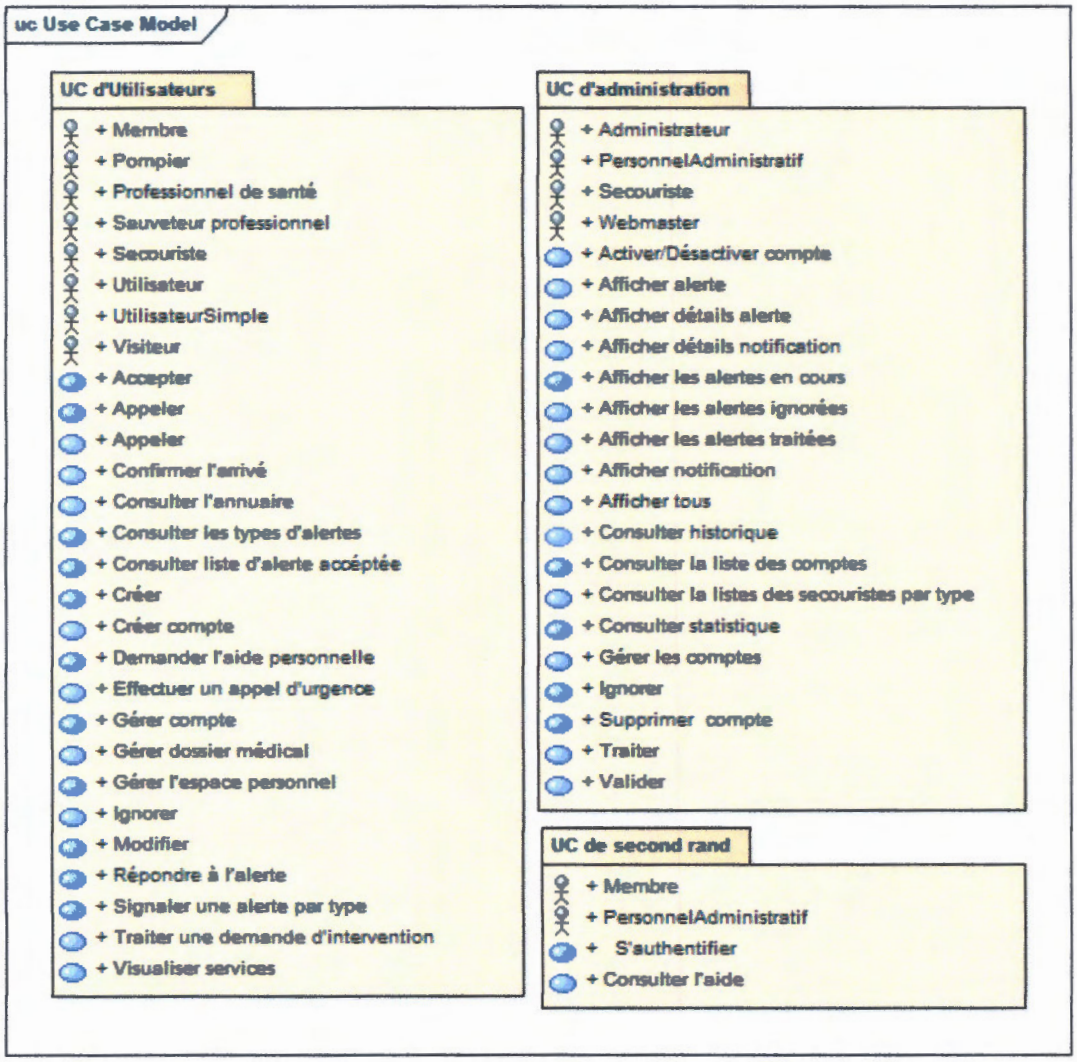


FIGURE 2.10 – Organisation des cas d'utilisation et des acteurs en packages (avec l'outil Enterprise Architect [31]).

2.4.5 Présentation des IHMs

Nous allons présenter les IHMs les plus importantes de chaque acteur.

1. **IHM de l'acteur *Visiteur***
   Le *Visiteur* possède deux IHMs : l'IHM principale et l'IHM de création de compte.

   (a) **IHM principale de *Visiteur*** : après l'installation de l'application le *Visiteur* se trouve sur l'IHM suivante à partir de laquelle il peut créer un compte et il peut aussi appeler les services d'urgence.

FIGURE 2.11 – IHM principale de *Visiteur*.

(b) **IHM de création de compte** : pour créer un compte, le *Visiteur* doit remplir le formulaire d'inscription présenté dans l'IHM d'inscription (Figure 2.12) qui contient les informations suivantes :

- Nom, prénom, date de naissance, Numéro de téléphone, etc. Le *Visiteur* peut s'inscrire comme utilisateur simple, ou comme *Secouriste* en spécifiant sa profession : pompier, sauveteur professionnel, professionnel de la santé.
- Adresse électronique valide : la validation du compte s'opère obligatoirement par l'obtention d'un email d'activation (contient un code) qui sera envoyé à l'adresse email saisie par le *Visiteur*, le code reçu sera utilisé dans l'IHM de validation d'émail (Figure 2.13).

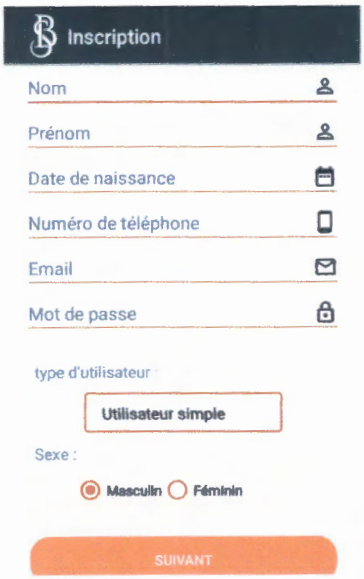


FIGURE 2.12 – IHM d'inscription.

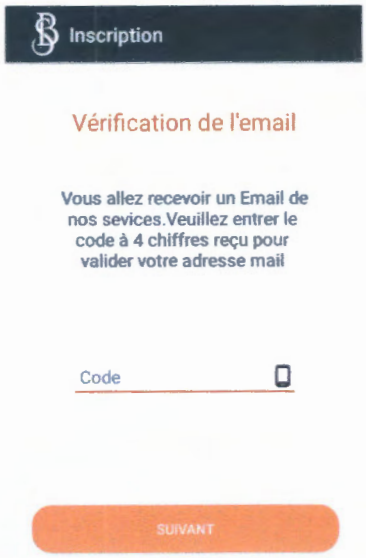


FIGURE 2.13 – IHM de validation d'email.

2. **IHM de l'acteur *Membre***

Un *Membre* possède les IHMs suivantes :

(a) **IHM d'authentification** : l'accès aux services s'effectue grâce à un nom d'utilisateur et un mot de passe (choisis lors de l'inscription) via cette IHM.

FIGURE 2.14 – IHM d'authentification.

(b) **IHM pour signaler l'alerte par type** : un *Membre* peut signaler une alerte par type ou effectuer un appel direct selon le type d'alerte (accident de route, incendie, crise et maladie, enlèvement). A partir de l'IHM présentée dans la Figure 2.15.

Quand le *Membre* choisit de signaler une alerte par type une boite de dialogue de confirmation (présentée dans la Figure 2.16) sera affichée pour valider l'alerte en précisant la position actuelle du *Membre* (son adresse).

FIGURE 2.15 – IHM pour signaler l'alerte.

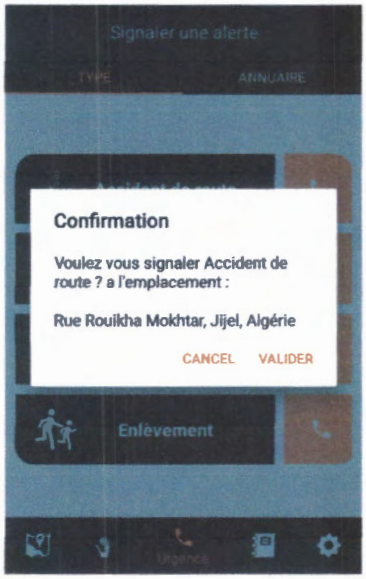


FIGURE 2.16 – IHM dialogue de confirmation.

(c) **IHM du dossier médical** : chaque *Membre* possède un dossier médical. Les IHMs du dossier médical (présentées dans les Figures ci-dessous) sont : L'IHM principale du dossier médical (Figure 2.17), l'IHM de choix du groupage sanguin (Figure 2.18), l'IHM des coordonnées du médecin traitant (Figure 2.19), l'IHM des antécédents médicaux (Figure 2.19) et l'IHM de saisit du poids (Figure 2.21).

FIGURE 2.17 – IHM du dossier médical.

FIGURE 2.18 – IHM type groupage.

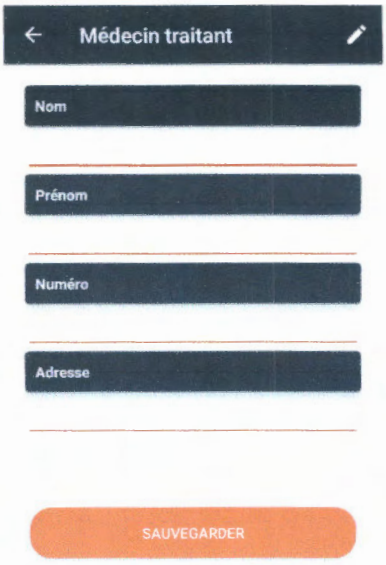


FIGURE 2.19 – IHM co-odonnées médecin

FIGURE 2.20 – IHM types maladies

FIGURE 2.21 – IHM de saisie du poids

(d) **IHM annuaire** : l'annuaire présenté dans la Figure suivante contient principalement trois numéros : le numéro des pompiers, le numéro de la police, et le numéro de gendarmerie pour les contacter facilement en cas d'urgence.

FIGURE 2.22 – IHM annuaire.

(e) **IHM d'aide personnelle** : à partir de l'IHM suivante un *Membre* peut ajouter des contacts d'urgence en plus des numéros cités précédemment. Il peut demander l'aide personnelle par appel téléphonique ou en signalant une alerte de type aide personnelle s'il a besoin d'aide.

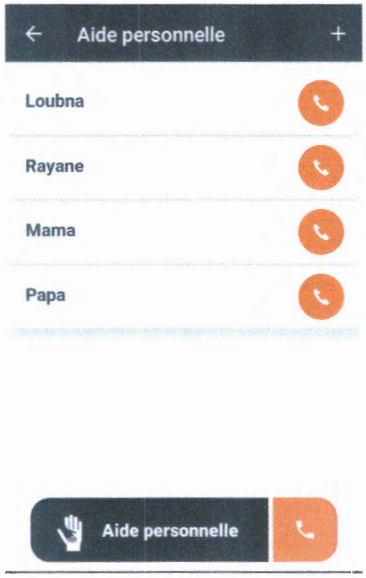


FIGURE 2.23 – IHM d'aide personnelle.

(f) **IHM principale du *Membre*** : l'IHM principale du *Membre* est présentée dans la figure suivante. Elle contient une carte géographique sur laquelle se situe la position actuelle du *Membre* avec la possibilité d'afficher les positions des services (hôpitaux, des pharmacies et des postes polices) qui se trouvent à proximité.

FIGURE 2.24 – IHM principale du *Membre*.

3. **IHM de l'acteur *Secouriste***

(a) **IHM de notification de l'alerte** : cette IHM est présentée dans la (Figure 2.25). Lors de la réception d'une notification d'alerte catégorisée (avec affichage du type d'alerte, la distance entre la position actuelle du *Secouriste* et la position de l'alerte) le *Secouriste* peut accepter d'aider ou ignorer l'alerte.

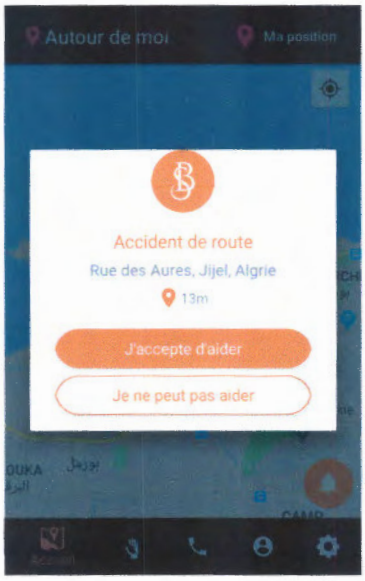


FIGURE 2.25 – IHM de notification de l'alerte.

(b) **IHM d'intervention** : quand le *Secouriste* accepte de donner l'aide, le système lui affiche des informations qui lui permettent d'intervenir.Ces informations sont

l'itinéraire à suivre pour aller jusqu'au lieu de l'incident, plus les détails de l'incident. Le *Secouriste* peut aussi indiquer qu'il est arrivé au lieu de l'incident, comme il peut contacter le service d'urgence par appel appel téléphonique.

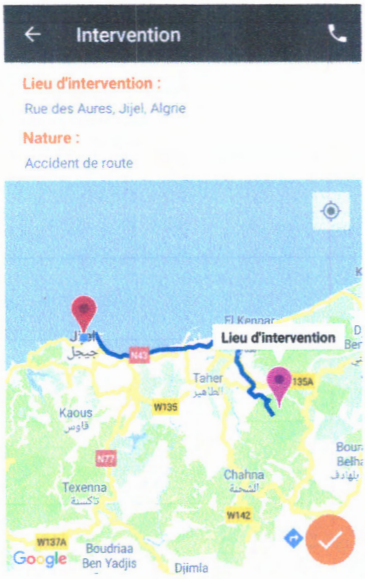


FIGURE 2.26 – IHM d'intervention

4. **IHM paramètre** : Cette IHM (Figure 2.27) permet au *Secouriste* de passer au mode *ne pas déranger* qui consiste à désactiver la réception des notifications d'alertes. Elle lui permet aussi de se déconnecter et d'éditer son compte. L'acteur *Membre* dispose aussi de cette IHM qui s'affiche sans montrer le bouton permettant l'activation/désactivation du mode *ne pas déranger*.

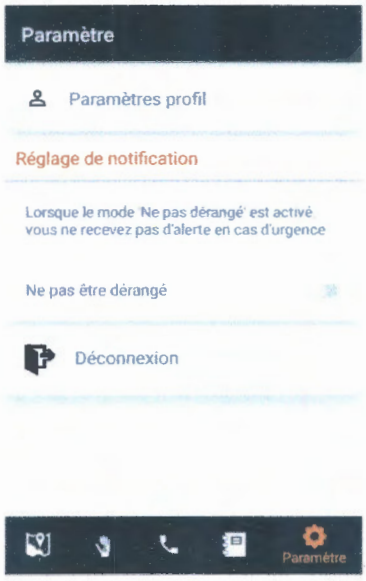


FIGURE 2.27 – IHM principale du *Membre*.

5. **IHM de l'acteur *Administrateur***

   L'*Administrateur* possède un tableau de bord représenté dans la Figure 2.28 qui lui permet la gestion des alertes. Lors de la réception d'une alerte par l'*Administrateur*, il peut choisir soit de la valider ou de l'ignorer. En validant l'alerte elle sera envoyée aux *Secouristes*, après le traitement de cet évènement l'*Administrateur* signale aux *Secouristes* qui avait accepté cette alerte quelle a été traitée.

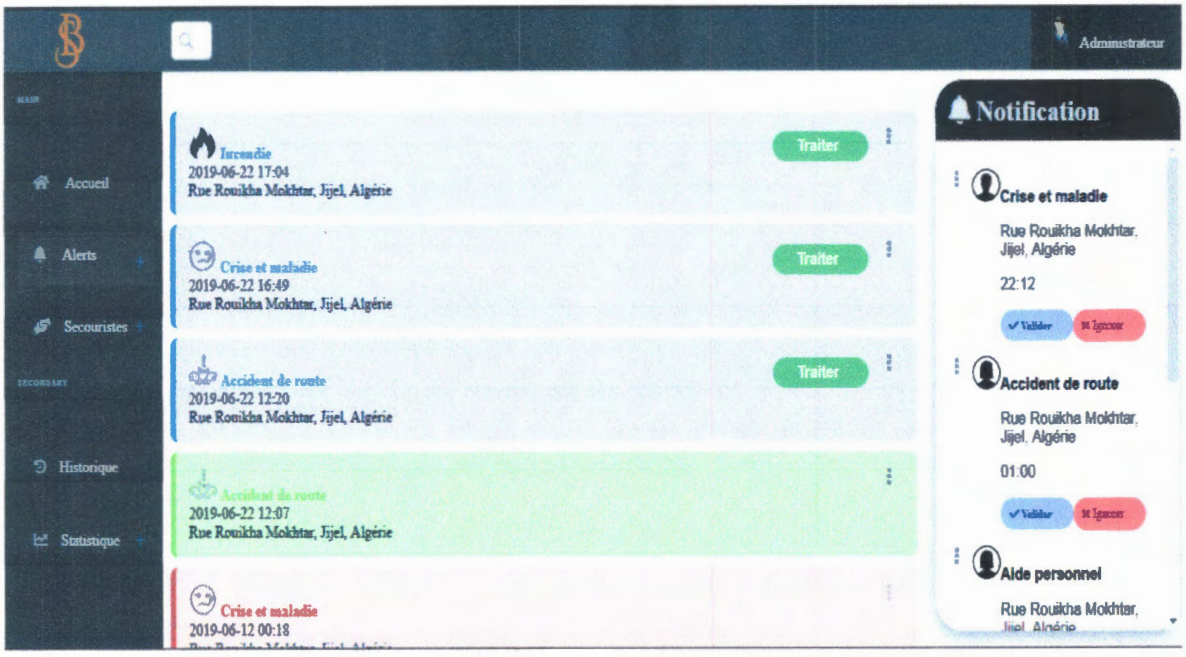


FIGURE 2.28 – IHM principale de l'*Administrateur*.

6. IHM de l'*Webmaster*

Le *Webmaster* gère les comptes des Membres, il peut activer/désactiver les comptes via l'IHM de la Figure 2.29. Il peut aussi supprimer et consulter la liste des comptes via l'IHM de la figure 2.30.

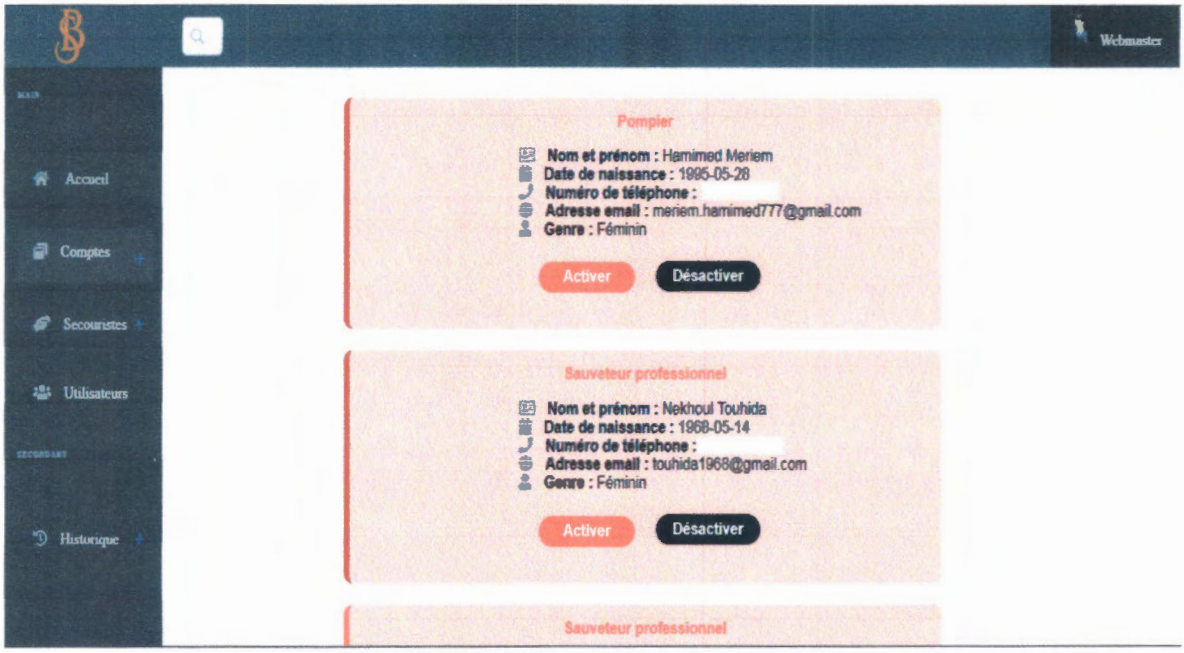


FIGURE 2.29 – IHM activation/désactivation des comptes.

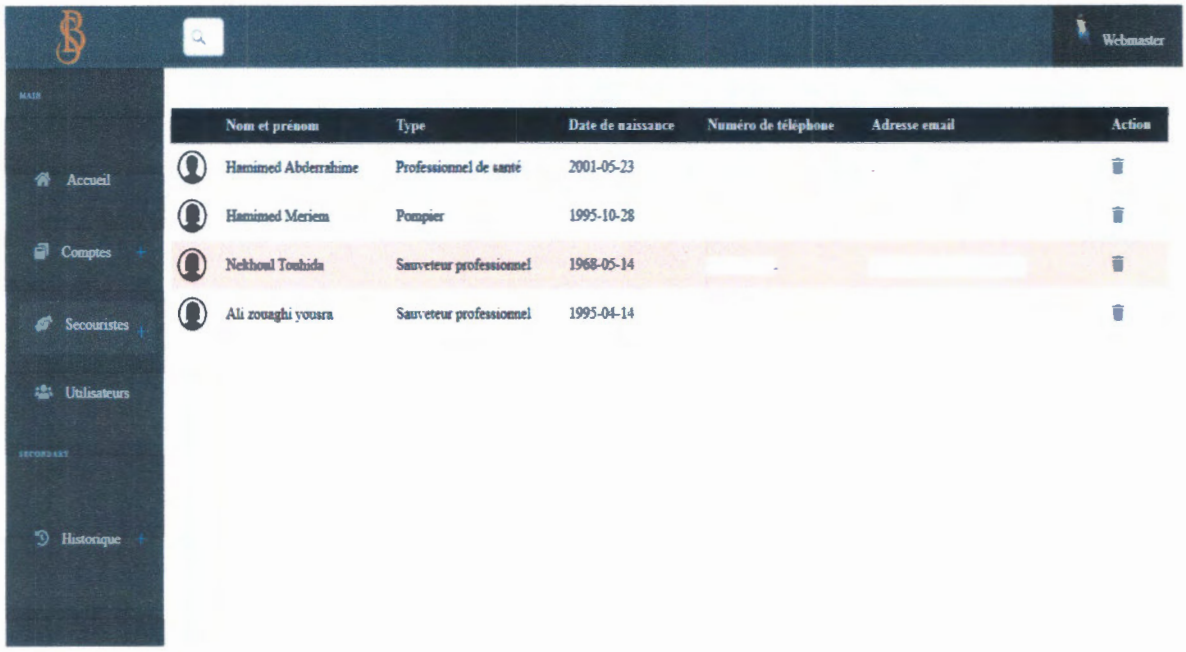


FIGURE 2.30 – IHM supression consulation de la liste des utilisateurs.

2.5 Spécification détaillée

Nous allons maintenant décrire textuellement les UCs de façon détaillée par des fiches types (FT) et des DSS. Les fiches types ne sont pas normalisées par UML, donc nous allons suivre le modèle de description proposé par Pascal Roques dans [12].
Pour donner une autre définition du cas d'utilisation, on peut dire que c'est une collection de scénarios de succès ou d'échec qui décrit la façon dont un acteur particulier utilise un système pour atteindre un objectif. Pour détailler la dynamique du cas d'utilisation, la procédure la plus évidente consiste à recenser de façon textuelle toutes les interactions entre les acteurs et le système. Le cas d'utilisation doit avoir un début et une fin clairement identifiés. Il faut aussi préciser les variantes possibles tout en essayant d'ordonner séquentiellement les descriptions afin d'améliorer leur lisibilité [12].
Les DSS qui décrivent les interactions fonctionnelles entre les acteurs et le système en le considérant comme une boite noire. Le comportement du système est décrit vu de l'extérieur, sans préjuger de comment il le réalisera [12].

2.5.1 Les fiches types et les diagrammes de séquence système

Nous choisissons les UCs les plus importants pour les détailler par des FT et les illustrer par les DSS.

1. FT et DSS de l'UC *Créer compte*

Cas d'utilisation	**Créer compte**
Acteur principal	*Visiteur*
Objectifs	Permet l'inscription des nouveaux utilisateurs.
Préconditions	
Postconditions	Un nouveau compte est créé.
Scénario nominal	**1** Le système affiche le formulaire d'inscription. **2** Le *Visiteur* remplit le formulaire et confirme. **3** Le système vérifie les informations. **4** Le système enregistre les informations du compte. **5** Le système affiche une notification de succès. **6** Le système dirige le *Visiteur* vers son espace personnel.
Alternative	**3a** les informations ne sont pas valides. 1. Le système affiche une notification d'échec. 2. Le *Visiteur* reprendre à partir de l'étape 1 du scénario nominal.

TABLE 2.1 – FT de l'UC *Créer compte*

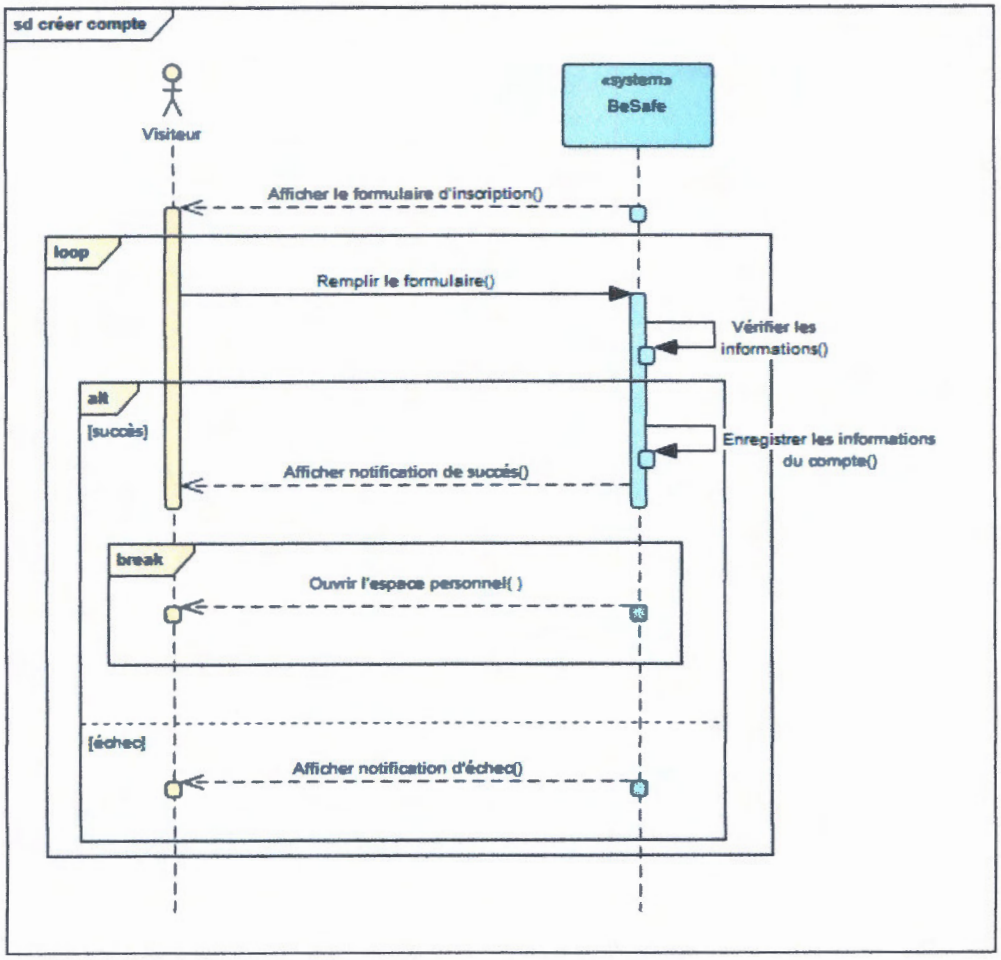


FIGURE 2.31 – DSS de l'UC *Créer compte*

2. FT et DSS de l'UC *S'authentifier*

Cas d'utilisation	**S'authentifier**
Acteur principal	*Membre*
Objectifs	Permettre au *Membre* d'accéder à son espace personnel.
Préconditions	
Postconditions	Ouverture de l'espace personnel
Scénario nominal	**1** Le système affiche le formulaire d'authentification. **2** Le *Membre* saisie son nom d'utilisation et son mot de passe. **3** Le système vérifie l'existence du compte. **4** Le système dirige le *Membre* vers son espace personnel.
Alternative	**3a** les informations d'authentification ne sont pas valides. 1. Le système affiche une notification d'échec. 2. Le *Membre* peut réessayer en revenant à l'étape 1 du scénario nominal s'il n'a pas dépassé 3 fois d'essai.

TABLE 2.2 – FT de l'UC *S'authentifier*

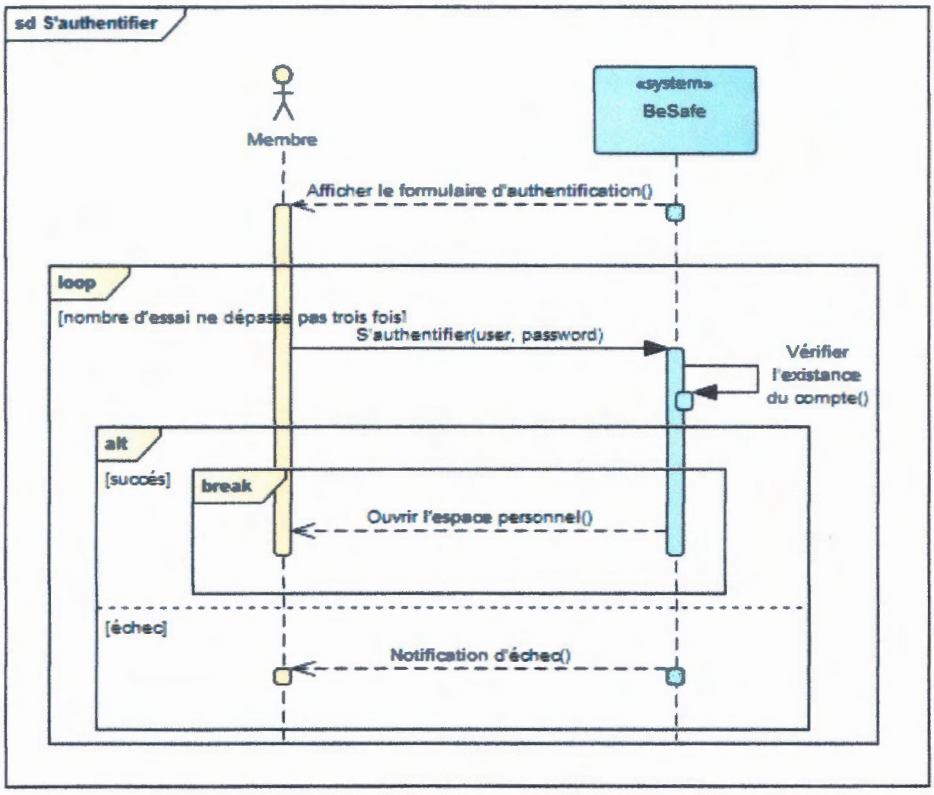


FIGURE 2.32 – DSS de l'UC *S'authentifier*

3. FT et DSS de l'UC *Consulter les types d'alertes*

Cas d'utilisation	**Consulter les types d'alertes**
Acteur principal	*Membre*
Objectifs	Permettre au *Membre* de signaler des alertes catégorisées.
Préconditions	*Membre* authentifié
Postconditions	Une nouvelle alerte est signalée
Scénario nominal	**1** Le *Membre* signale une alerte en spécifiant son type. **2** Le système affiche un accusé de réception.
Alternative	**1a** Le *Membre* signale une alerte en appelant le type correspondant .

TABLE 2.3 – FT de l'UC *Consulter les types d'alertes*

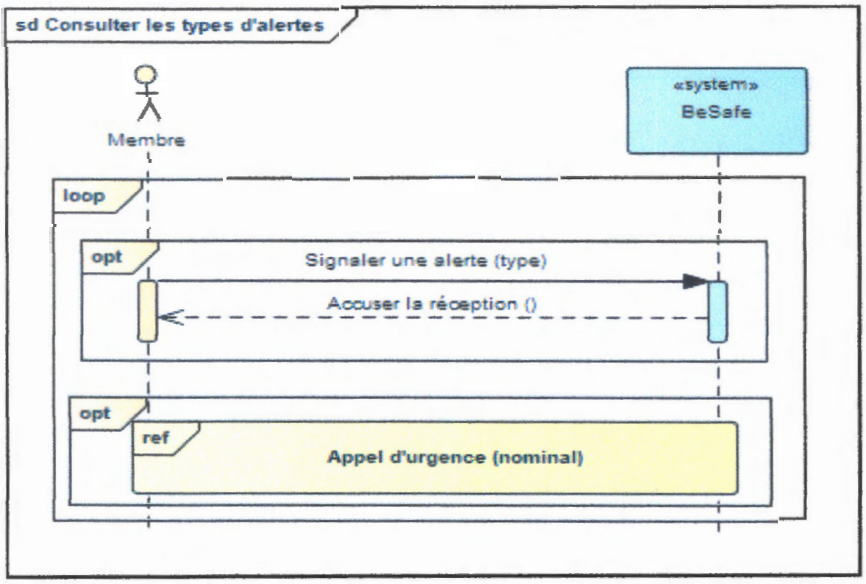


FIGURE 2.33 – DSS de l'UC *Consulter les types d'alertes.*

4. FT et DSS de l'UC *Traiter une demande d'intervention*

Cas d'utilisation	**Traiter une demande d'intervention**
Acteur principal	*Secouriste*
Objectifs	Permettre au *Secouriste* de traiter l'alerte.
Préconditions	*Secouriste* authentifié
Postconditions	Alerte acceptée
Scénario nominal	**1** Le *Secouriste* reçoit une notification d'alerte. **2** Le *Secouriste* peut accepter l'alerte s'il veut offrir l'aide. **3** Le système affiche l'itinéraire et les détails de l'alerte **4** Le *Secouriste* peut se mettre en contact avec le système s'il accepte l'alerte. **5** Le *Secouriste* peut confirmer son arrivée au lieu de l'incident pour donner les premiers secours (la confirmation se réalise une fois le *Secouriste* est arrivé).
Alternative	**2a** Le *Secouriste* peut ignorer l'alerte s'il ne peut pas offrir l'aide.

TABLE 2.4 – FT de l'UC *Traiter une demande d'intervention*

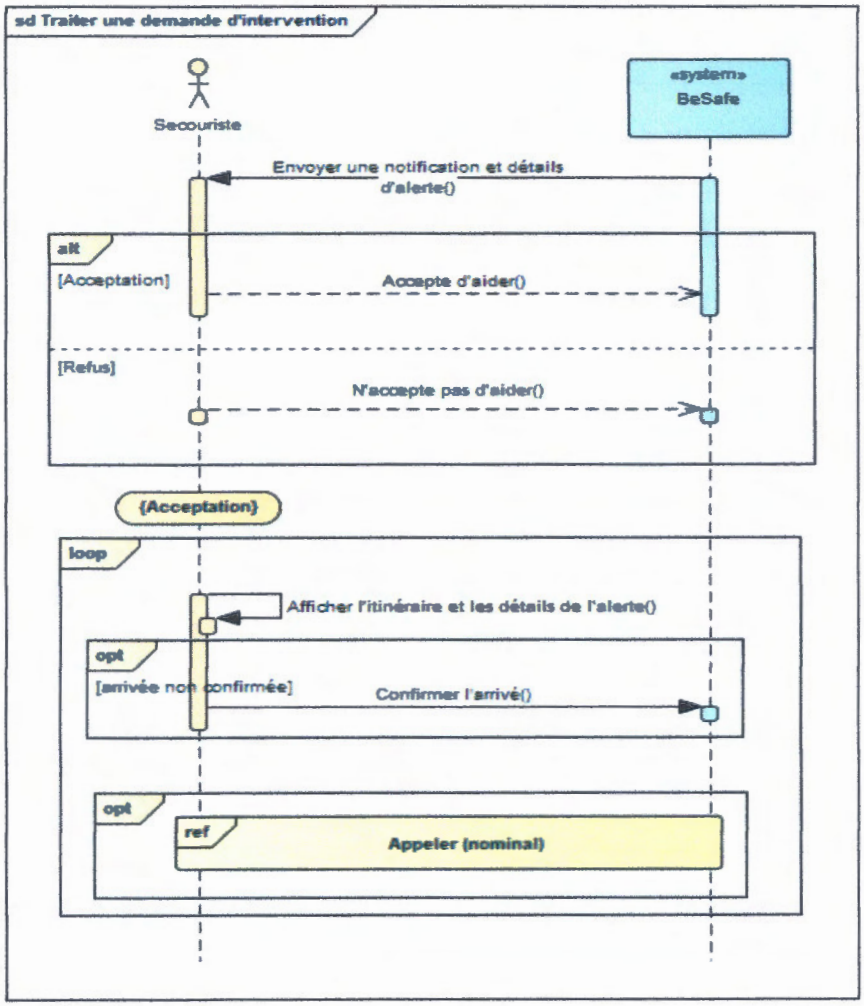


FIGURE 2.34 – DSS de l'UC *Traiter une demande d'intervention.*

5. FT et DSS de l'UC *Afficher notification*

Cas d'utilisation	**Afficher notification**
Acteur principal	*Administrateur*
Objectifs	Permet de valider ou d'ignorer l'alerte.
Préconditions	*Administrateur* authentifié
Postconditions	Alerte validée et propagée
Scénario nominal	**1** l'*Administrateur* reçoit une alerte envoyée par un *Membre.* **2** L'*Administrateur* peut valider l'alerte (changer son état vers validée) et la propager vers tous les *Secouristes.* **3** Les alertes en cours seront affichées.
Alternative	**2a** L'*Administrateur* peut ignorer l'alerte

TABLE 2.5 – FT de l'UC *Afficher notification*

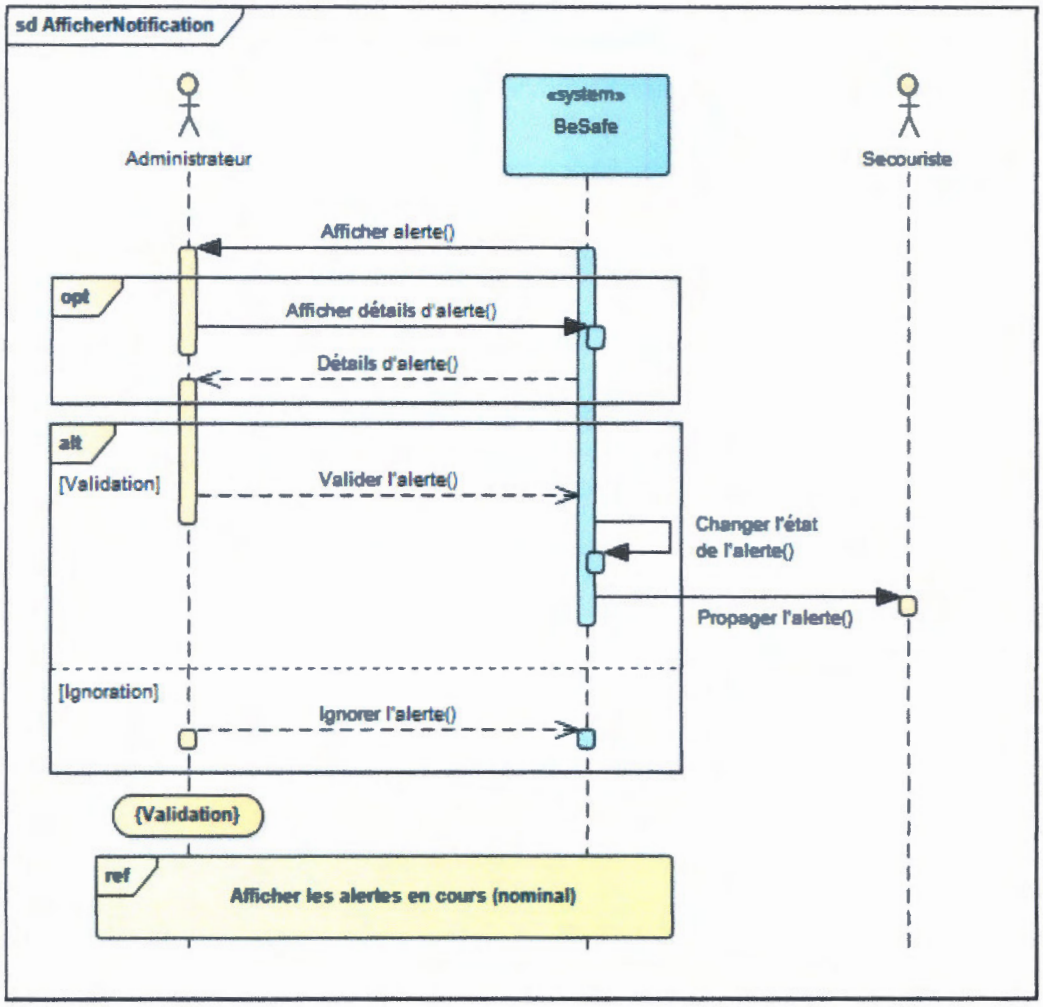


FIGURE 2.35 – DSS de l'UC *Afficher notification.*

6. FT et DSS de l'UC *Afficher les alertes en cours*

Cas d'utilisation	**Afficher les alertes en cours**
Acteur principal	*Administrateur*
Objectifs	Informer les *Secouristes* qu'une alerte a été traitée.
Préconditions	*Administrateur* authentifié
Postconditions	Alerte traitée
Scénario nominal	**1** l'*Administrateur* sélectionne une alerte. **2** L'*Administrateur* peut changer l'état de l'alerte et le rend traitée. **3** Après le changement d'état de l'alerte, les Secouristes qui ont accepté cette alerte seront informés . **1a** L'*Administrateur* affiche les détails de l'alerte sélectionnée.
Alternative	

TABLE 2.6 – FT de l'UC *Afficher les alertes en cours*

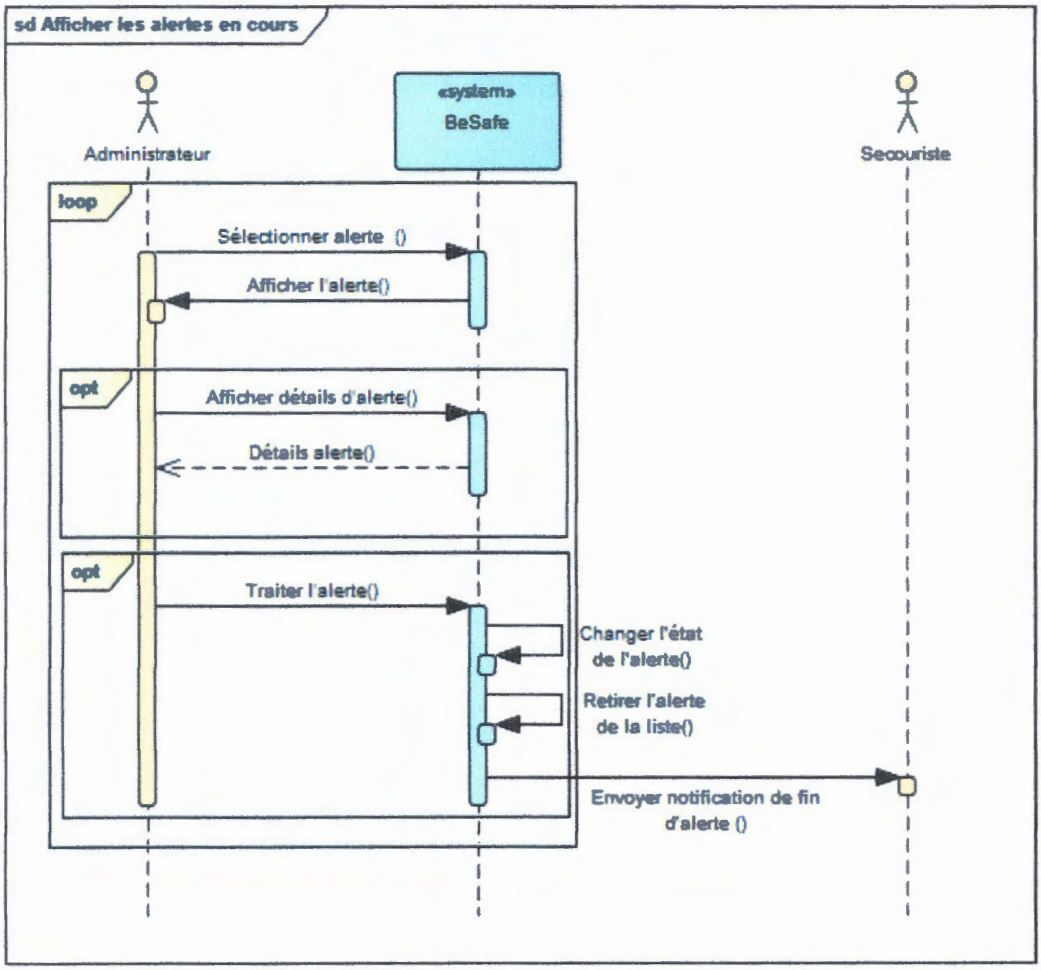


FIGURE 2.36 – DSS de l'UC *Afficher les alertes en cours.*

2.6 Concluion

La phase de spécification des besoins, présentée dans ce chapitre, nous a permet en premier temps d'exprimer les fonctionnalités du future système en utilisant les diagrammes de cas d'utilisations. Ensuite, nous avons présenté les IHMs de ces différents diagrammes et détaillé ces derniers en utilisant des diagrammes de séquences UML combinés avec des descriptions textuelles.

Chapitre 3

Analyse et conception

3.1 Introduction

Nous présentons dans ce chapitre les deux étapes fondamentales qui précèdent l'implémentation et qui sont l'analyse et la conception. Cela passe premièrement par l'élaboration des diagrammes de classes participantes des différents UCs en exploitant la maquette. Ces DCPs qui permettent principalement d'identifier les concepts du domaine sont regroupés et raffinés pour obtenir le diagramme de classes global. Nous présentons aussi la partie dynamique du système via le diagramme d'état et les diagrammes de navigation qui permettent de modéliser la navigation entre les différentes IHMs.

3.2 Diagramme de classes participantes

Commençons par l'élaboration des diagrammes de classes participantes qui vient de l'analyse des UCs et des maquettes comme le montre la Figure suivante.

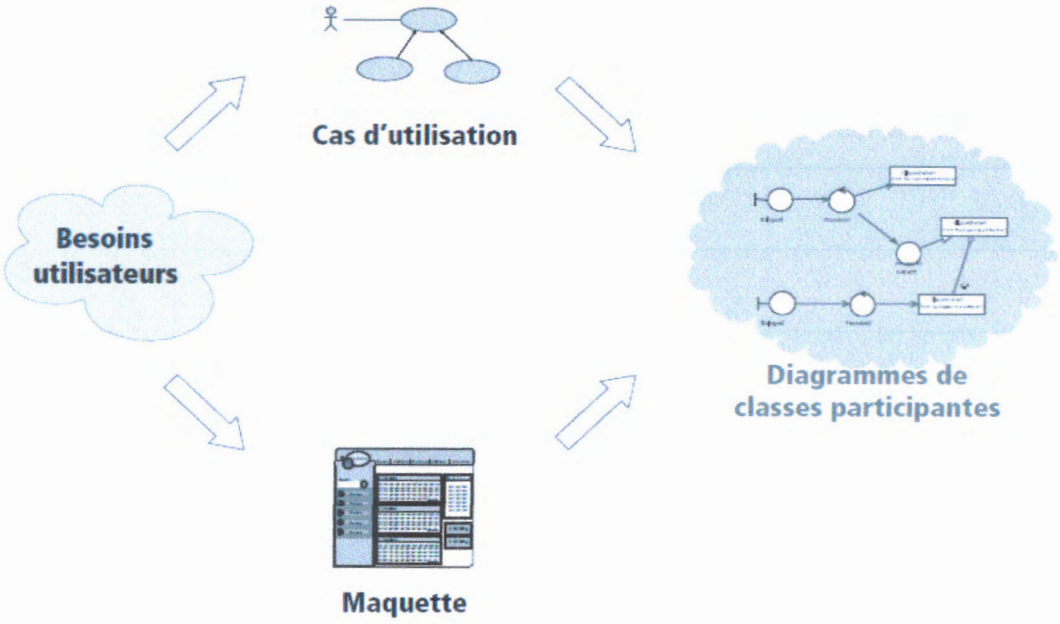


FIGURE 3.1 – Positionnement des diagrammes de classes participantes dans la démarche [12].

3.2.1 Principe et démarche

Les diagrammes de classes participantes sont des diagrammes de classes UML qui décrivent cas d'utilisation par cas d'utilisation, les trois principales classes d'analyse (Entité, Dialogue et Contrôle) et leurs relations [12].
Pour réaliser les DCPs, il faut identifier les concepts du domaine et les classes d'analyses puis ajouter les associations et les attributs.

1. **Identification des concepts du domaine et des classes d'analyses**
   Cela consiste à décomposer un domaine d'intérêt en classes conceptuelles représentant les entités significatives de ce domaine. Il s'agit simplement de créer une représentation visuelle des objets du monde réel dans un domaine donné. Il s'agit d'un ensemble de diagrammes de classes dans lesquels on fait figurer les éléments suivants :

   - Les classes conceptuelles ou les objets du domaine ;
   - Les associations entre classes conceptuelles ;
   - les attributs des classes conceptuelles.

   Utiliser la catégorisation des classes d'analyse qui a été proposée par *I. Jacobson* et popularisée ensuite par le RUP(Rational Unified Process).
   Les classes d'analyse se répartissent en trois catégories :

   (a) Les *dialogues* s'agissent des écrans (IHM) proposés à l'utilisateur. Ces classes proviennent directement de l'analyse de la maquette. Les dialogues vont posséder des

attributs et des opérations qui représenteront des champs de saisie ou des résultats, ces résultats seront distingués en utilisant la notation de l'attribut dérivé. Les opérations représenteront des actions de l'utilisateur sur l'IHM.

(b) Les *contrôles* contiennent la cinématique de l'application. Elles font la transition entre les dialogues et les concepts du domaine, en permettant aux écrans de manipuler des informations détenues par des objets métier. Elles contiennent les règles applicatives et les isolent à la fois des objets d'interface et des données persistantes. Les contrôles ne donnent pas forcément lieu à de vrais objets de conception, mais assurent que nous n'oublions pas de fonctionnalités ou de comportements requis par les cas d'utilisation. Les contrôles vont seulement posséder des opérations qui montrent la logique de l'application, des règles transverses à plusieurs entités. Il y a souvent un seul contrôle par cas d'utilisation, mais il peut également y en avoir plusieurs, en fonction du nombre et de la cohérence des comportements associés.

(c) Les *entités* représentent les concepts métier. Elles sont très souvent persistantes, c'est-à-dire qu'elles vont survivre à l'exécution d'un cas d'utilisation particulier et qu'elles permettront à des données et des relations d'être stockées dans des fichiers ou des bases de données. Les entités vont posséder seulement des attributs qui représentent en général des informations persistantes de l'application.

2. **Ajout des associations et des attributs**
   Une fois que l'on a identifié les concepts fondamentaux, il est utile d'ajouter :

   - les associations nécessaires pour prendre en compte les relations qu'il est fondamental de mémoriser ;
   - les attributs nécessaires pour répondre aux besoins d'information.

Les DCPs sont importants car ils font la jonction entre les cas d'utilisation, la maquette et les diagrammes de conception logicielle (diagrammes d'interaction et diagrammes de classes), ainsi qu'ils offrent pour le chef de projet la possibilité de découper le travail de son équipe d'analystes suivant les différents cas d'utilisation, plutôt que de vouloir tout traiter d'un bloc [12] .

3.2.2 DCP des cas d'utilisation de notre système

Nous allons présenter les DCPs des principaux cas d'utilisation que nous avons identifiés pour notre système.

1. **DCP de l'UC *Créer un compte***

   Le *Visiteur* peut créer un compte pour bénéficier des fonctionnalités de l'application. Lors de la création du compte, le *Visiteur* passe par deux écrans gérés par un contrôle unique. Le DCP de l'UC Créer un compte est présenté dans la Figure ci-dessous

- L'écran formulaire d'inscription (dialogue *Formulaire d'Inscription*) qui lui permet de saisir ses informations (nom, prénom, etc) avec le choix de type du compte (utilisateur simple, pompier, sauveteur professionnel, professionnel de la santé).
- L'écran de saisie du code de confirmation (dialogue *CodeConfirmation*) pour la validation de l'email.
- Le contrôle *(CtrDeValidité)* contient les opérations suivantes : la vérification des champs de saisies, la vérification de l'existence du compte, la vérification de la validité du code de confirmation, l'affichage de notification en cas d'échec, l'inscription en cas de validité des informations.

Cet UC nous permet d'identifier 7 classes entités : *Compte*, *Membre*, *Secouriste* et *Utilisateur Simple*, *Professionnel de santé*, *Pompier* et *Sauveteur Professionnel*.

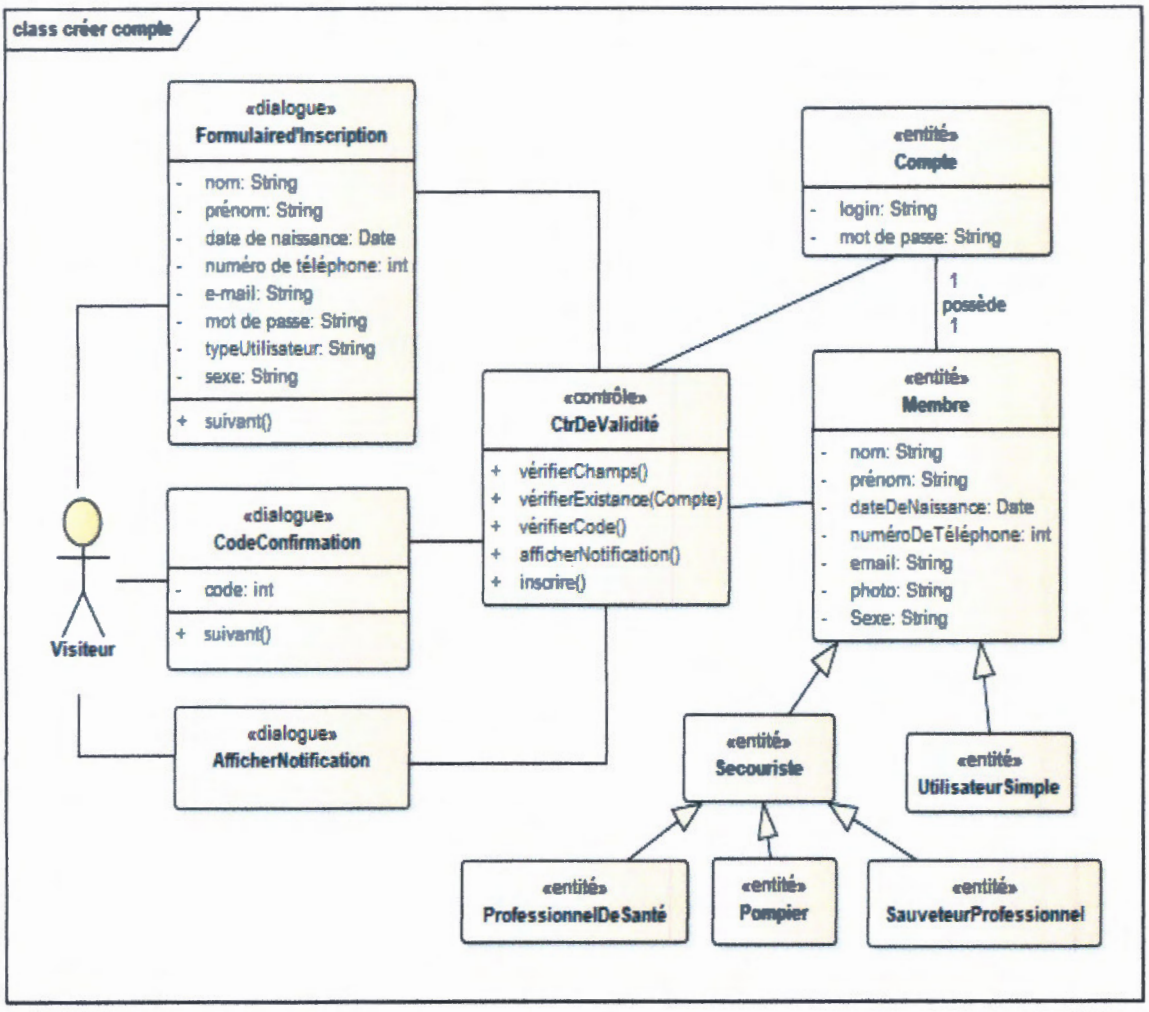


FIGURE 3.2 – DCP de l'UC *Créer un compte*

2. **DCP de l'UC *Gérer dossier médical***

Chaque *Membre* possède un dossier médical qui contient les informations suivantes : poids, maladies, médecin traitant et groupage sanguin.
Il y a quatre dialogues qui permettent au *Membre* de saisir ses informations qui sont gérées par un seul contrôle. Le DCP de l'UC Gérer dossier médical est présenté dans la Figure ci-dessous.

- L'écran *gérer dossier médical* qui est l'écran principal du dossier médical.
- L'écran de saisie des *coordonnées du médecin traitant.*
- L'écran de sélection des *maladies.*
- L'écran de sélection du *groupage* sanguin.
- L'écran de saisie du *poids.*
- Le *contrôle du dossier médical* qui contient l'opération d'affichage et l'opération d'enregistrement.

Cet UC nous permet d'identifier 6 classes entités qui sont : *Localisation*, *Dossier médical*, *Médecin traitant*, *Maladie*, *Infectieux*, *cardiovasculaire* et *Diabète.* De plus, il utilise la classe *Membre* précédemment identifiée.

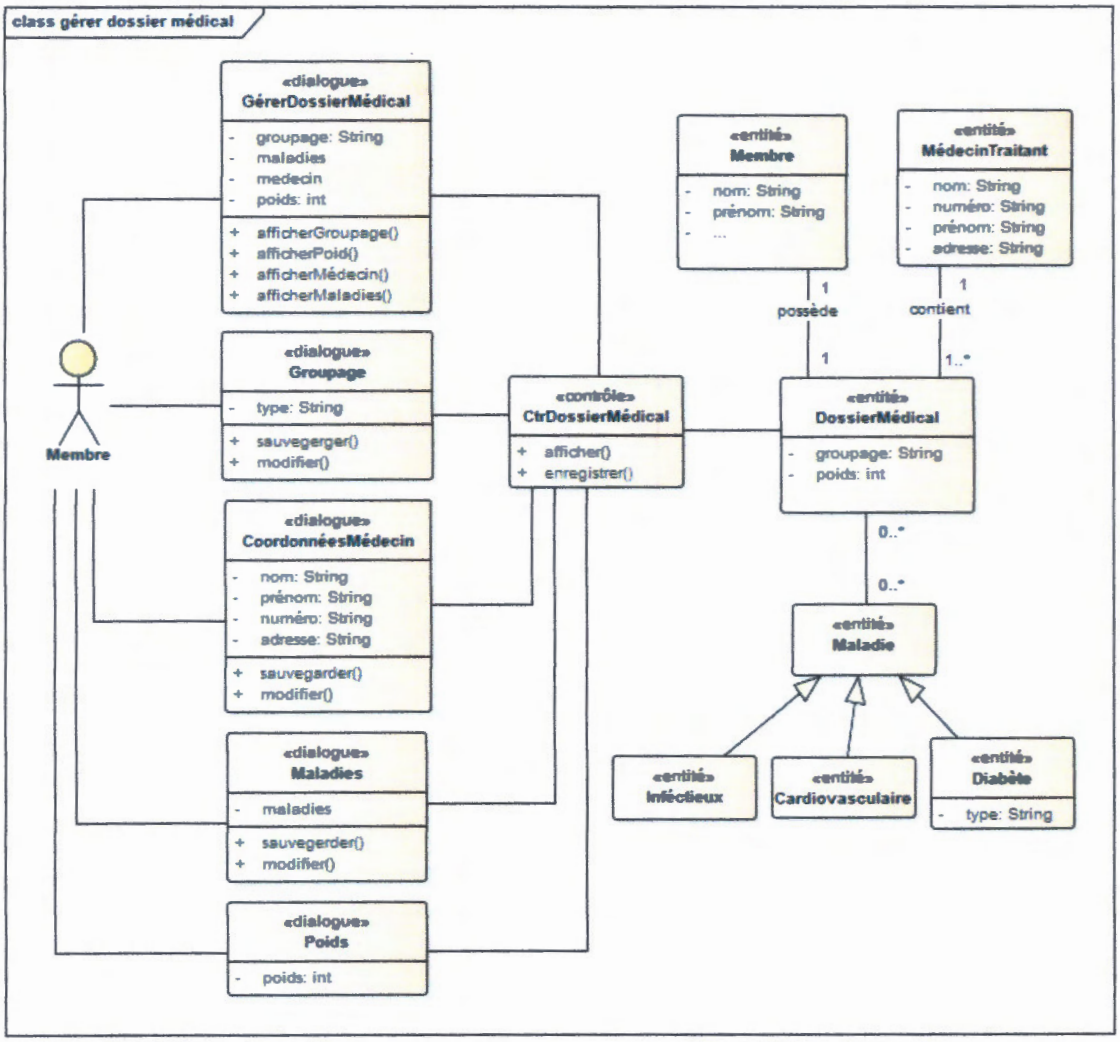


FIGURE 3.3 – DCP de l'UC *Gérer dossier médical*

3. **DCP des UCs *Consulter les types d'alertes*, *Signaler une alerte par type* et *Appeler***

   Le *Membre* peut consulter les types d'alertes afin de signaler une alerte en sélectionnant son type, il peut en outre effectuer un appel téléphonique. Le *Membre* a à sa disposition deux écrans représentés par deux dialogues, gérés par un contrôle comme le montre la Figure suivante.

   - L'écran signalé alerte par type qui contient les quatre catégories d'alertes.
   - Boite de dialogue de confirmation d'envoi de l'alerte.
   - Le contrôle d'alerte permet de signaler l'alerte on récupérant la localisation qui contient l'adresse et les coordonnées GPS, le temps actuel, le nom du *Membre*, afficher le dialogue de confirmation et effectuer un appel.

Cet UC utilise un ensemble de classes. En plus des classes précédemment identifiées : *Membre* et *Localisation*, nous avons identifié de nouvelle classes pour cet UC : *Alerte* qui contient quatres types *Accident de route*, *Crise et maladies*, *Incendie* et *Enlèvement*.

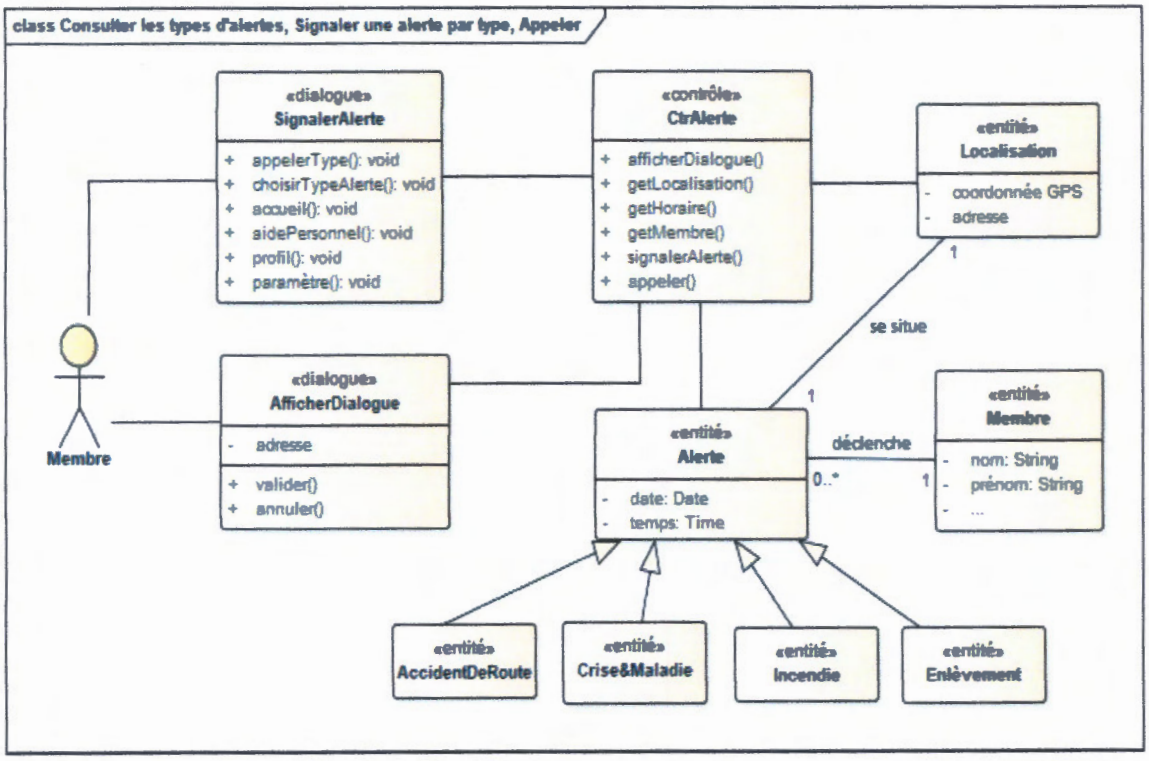


FIGURE 3.4 – DCP des UCs *Consulter les types d'alertes*, *Signaler une alerte par type* et *Appeler*

4. **DCP de l'UC *Demander aide personnelle***

Le DCP de l'UC *Demander aide personnelle* est présenté dans la Figure 3.5.

L'aide personnelle est basée sur le dossier médical du *Membre*, donc ce dernier peut demander une aide personnelle en envoyant une alerte de type *aide personnelle* avec sa localisation, il peut aussi ajouter une liste de contacts, il possède trois dialogue gérés par deux contrôles.

- L'écran principal de l'aide personnelle.
- L'écran qui contient les détails du contacte.
- Boite de dialogue de confirmation de demande d'aide personnelle.
- Le contrôle de contact contient les opérations suivantes : sauvegarder, modifier, supprimer et afficher contact.
- Le contrôle d'alerte (le même cité dans le DCP précèdent).

Cet UC nous permet d'identifier les classes entité suivantes : la classe *Aide personnelle* qui est une spécialisation de la classe *Alerte*, la classe *Dossier médical* et la classe

Contact. En plus, il utilise les classes entités *Membre*, *Localisation* et *Alerte* que nous avons identifiées dans le DCP précédent.

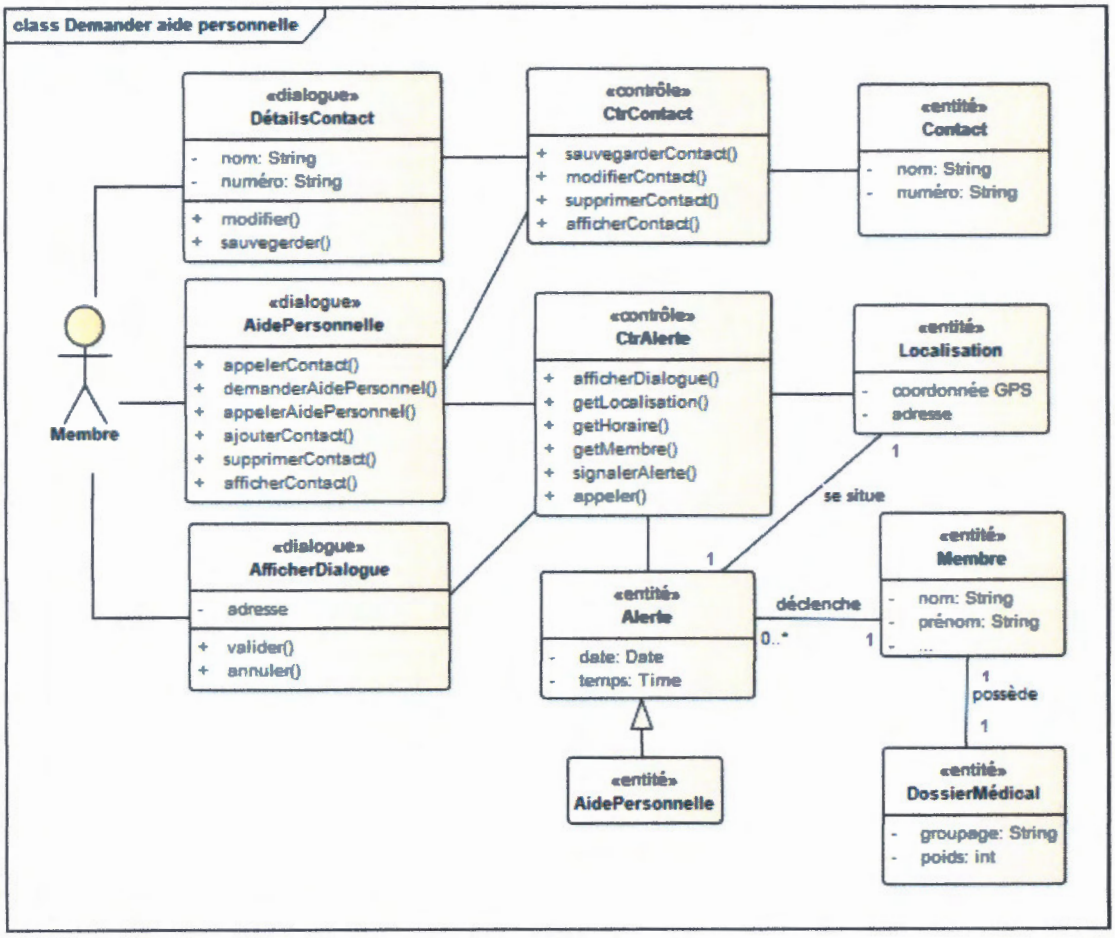


FIGURE 3.5 – DCP de l'UC *Demander aide personnelle*

5. **DCP de l'UC *Traiter une demande d'intervention***

Une fois le *Secouriste* reçoit une notification et l'accepte, il peut voir plus de détails sur l'alerte. Il peut en plus appeler le service de secours, comme il peut confirmer son arrivée au lieu de l'incident. Le *Secouriste* a à sa disposition les dialogues présentée dans la Figure 3.6.

- Notification d'alerte à partir de laquelle le *Secouriste* peut accepter ou refuser l'alerte.
- L'écran de détails de l'alerte contenant une carte géographique sur laquelle s'affiche la position de l'alerte et la position actuelle du *Secouriste*. Cet écran permet aussi au *Secouriste* de confirmer son arrivée au lieu de l'alerte et d'effectuer un appel.
- Boite de dialogue distance qui affiche la distance entre le lieu de l'alerte et la position de *Secouriste* lorsque le *Secouriste* veut confirmer son arrivée.

Ces dialogues sont gérés par les contrôles suivants :

- Le contrôle de notification contient les opérations suivantes : afficher la position de *Secouriste*, la position et le détail de l'alerte et envoyer la réponse du *Secouriste*.
- Le contrôle d'intervention permet d'afficher la notification de l'alerte, confirmer l'arrivée et effectuer un appel.

Cet UC nous permet d'identifier une nouvelles classe entité : *Réponse*.

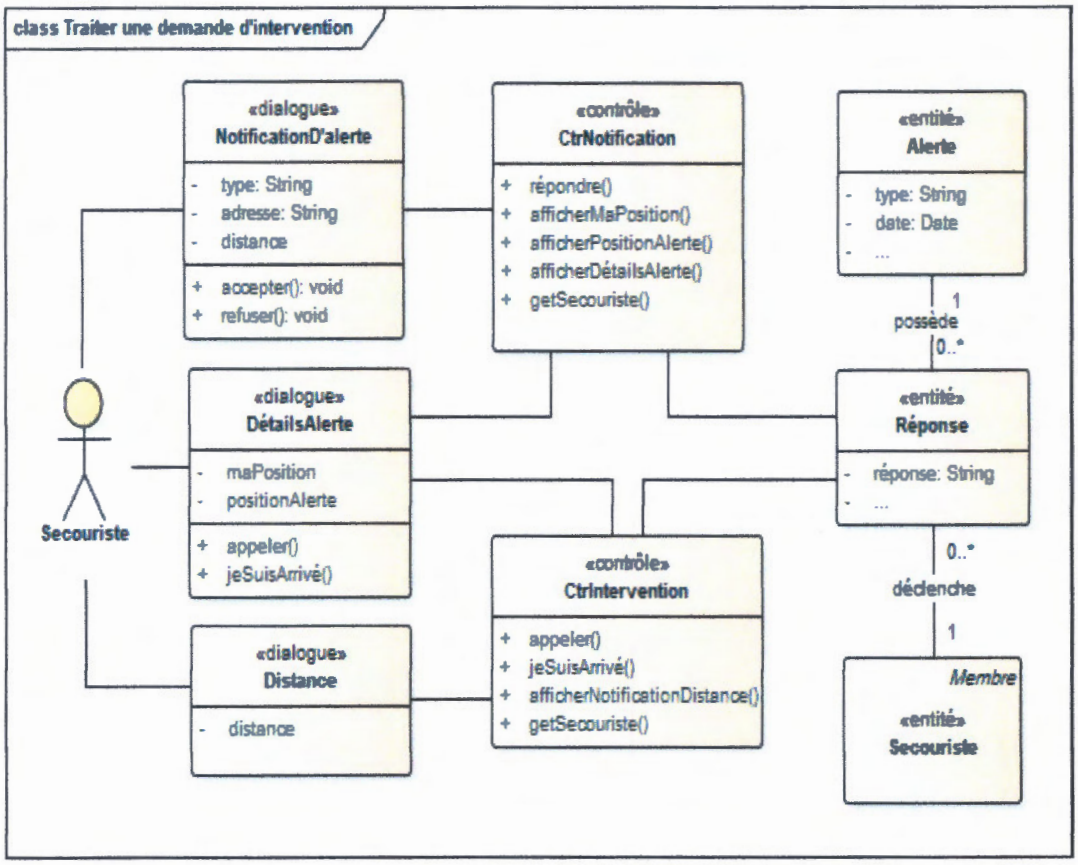


FIGURE 3.6 – DCP de l'UC *Traiter une demande d'intervention*

6. **DCP de l'UC *Visualiser les positions sur la carte***

Le *Membre* peut visualiser sur la carte les emplacements des hôpitaux, pharmacies, postes polices. Ce DCP est Présenté dans la Figure 3.7, il contient deux dialogues gérés par un seul contrôle :

- L'écran d'accueil de l'application contient la carte avec la position actuelle du *Membre*. Ce dernier peut visualiser les emplacements des hôpitaux, pharmacies et des postes polices.
- Le dialogue détails permet d'afficher les détails d'un service.

- Le contrôle s'occupe de l'affichage des emplacements selon le type choisi et l'affichage des détails d'un service.

Ce DCP permet d'identifier les classes entités suivantes : *Services*, *Police*, *Pharmacie* et *Hôpital*.La classe *Services* qui représent une généralisation des différents types de services représentés par les sous classes *Police*,*Pharmacie* et *Hôpital*.

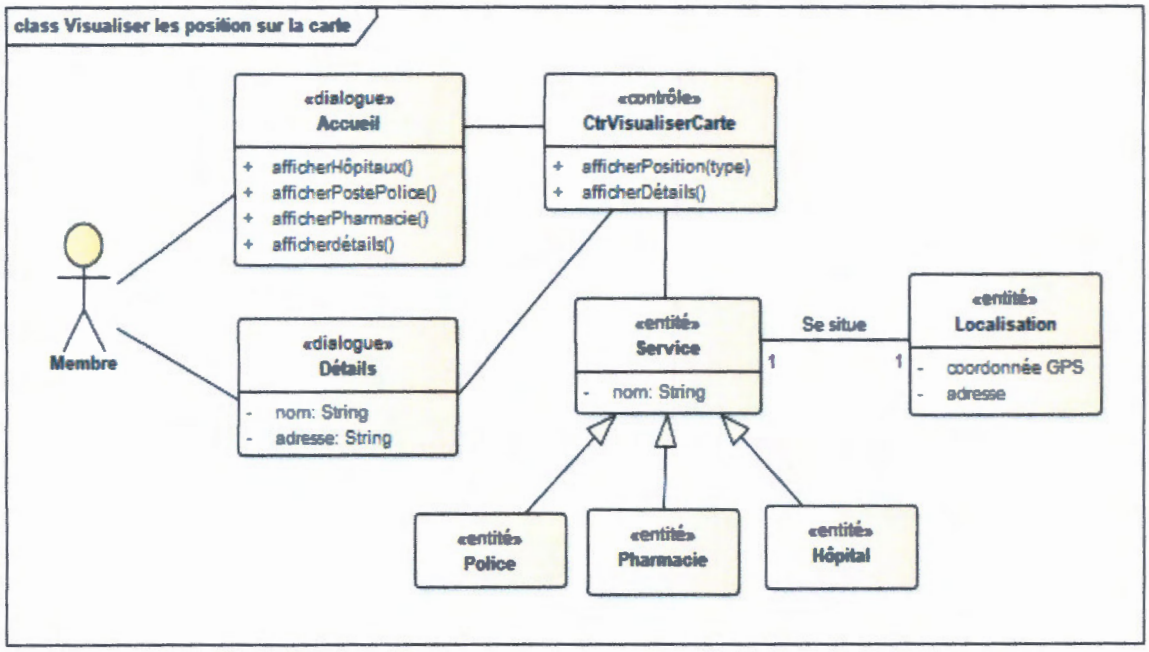


FIGURE 3.7 – DCP de l'UC *Visualiser les positions sur la carte*

7. **DCP de l'UC *Consulter l'annuaire***

Un *Membre* peut consulter l'annuaire et peut aussi effectuer un appel en cas d'urgence. Le DCP de l'UC *Consulter l'annuaire* est présenté dans la Figure 3.8. Il contient un seul dialogue et un seul contrôle. Cet UC permet d'ajouter les entités *Service pompier*, *Ambulance* et *Gendarmerie* qui héritent de la classe *Service*.

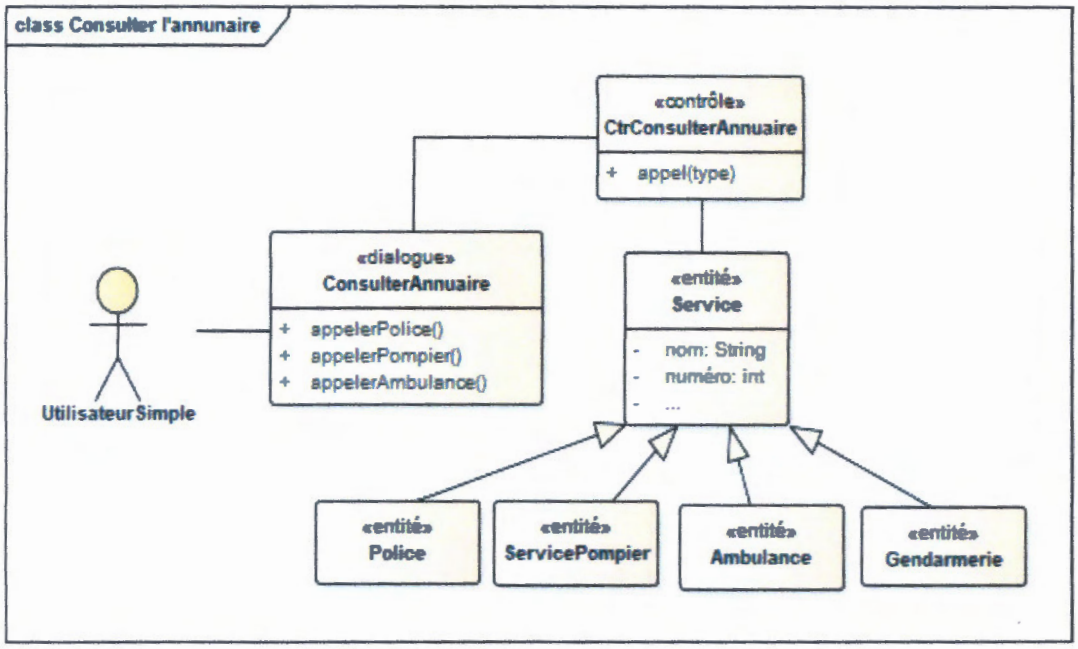


FIGURE 3.8 – DCP de l'UC *Consulter l'annuaire*

8. **DCP de l'UC *Afficher alerte***

L'*Administrateur* peut afficher les alertes selon leurs types comme il peut afficher les détails d'une alerte. Le DCP de l'UC *Afficher alerte* est présenté dans la Figure 3.9. Ce DCP contient trois dialogues et deux contrôles :

- Le dialogue alerte contient des informations sur une alerte.
- Le dialogues détails alerte permet d'afficher plus de détails d'une alerte.
- Les deux contrôles (CtrAlerte et CtrDétails) permet d'afficher les alertes avec leurs détails.

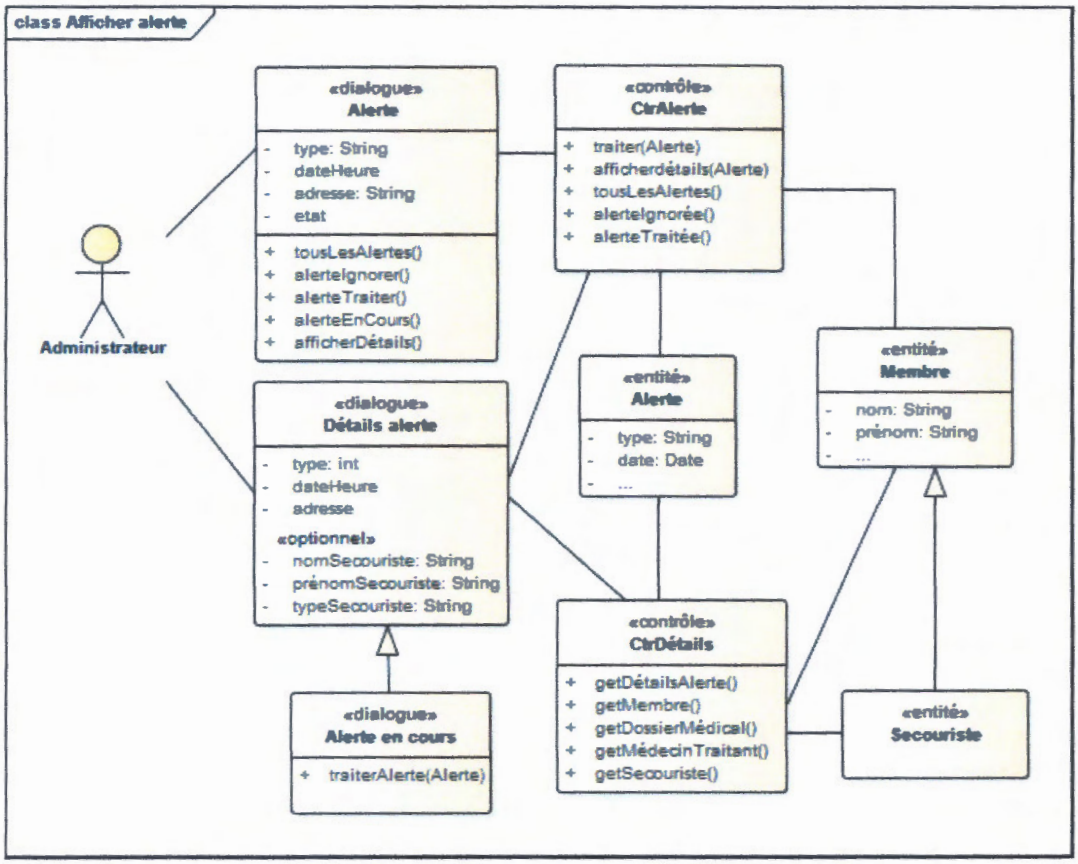


FIGURE 3.9 – DCP de l'UC *Afficher alerte*

9. **DCP de l'UC *Afficher notification***

L'*Administrateur* reçoit des alertes, qui s'affichent dans la barre de notification. Il peut les valider comme il peut les ignorer.

Le DCP l'UC *Afficher notification* est présenté dans la Figure 3.10. Il contient quatre dialogues gérés par deux contrôles :

- La barre de notification d'alerte permet de valider ou d'ignorer l'alerte.
- La boîte de dialogue de confirmation (pour valider ou ignorer l'alerte).
- Liste des alertes en cours.
- Détails de l'alerte.
- Le contrôle de notification contient les opérations suivantes : valider, ignorer, afficher détails d'une alerte, afficher les alertes en cours, confirmer la validation ou l'ignoration de l'alerte.
- Le contrôle de détails permet d'afficher plus de détails d'une alerte.

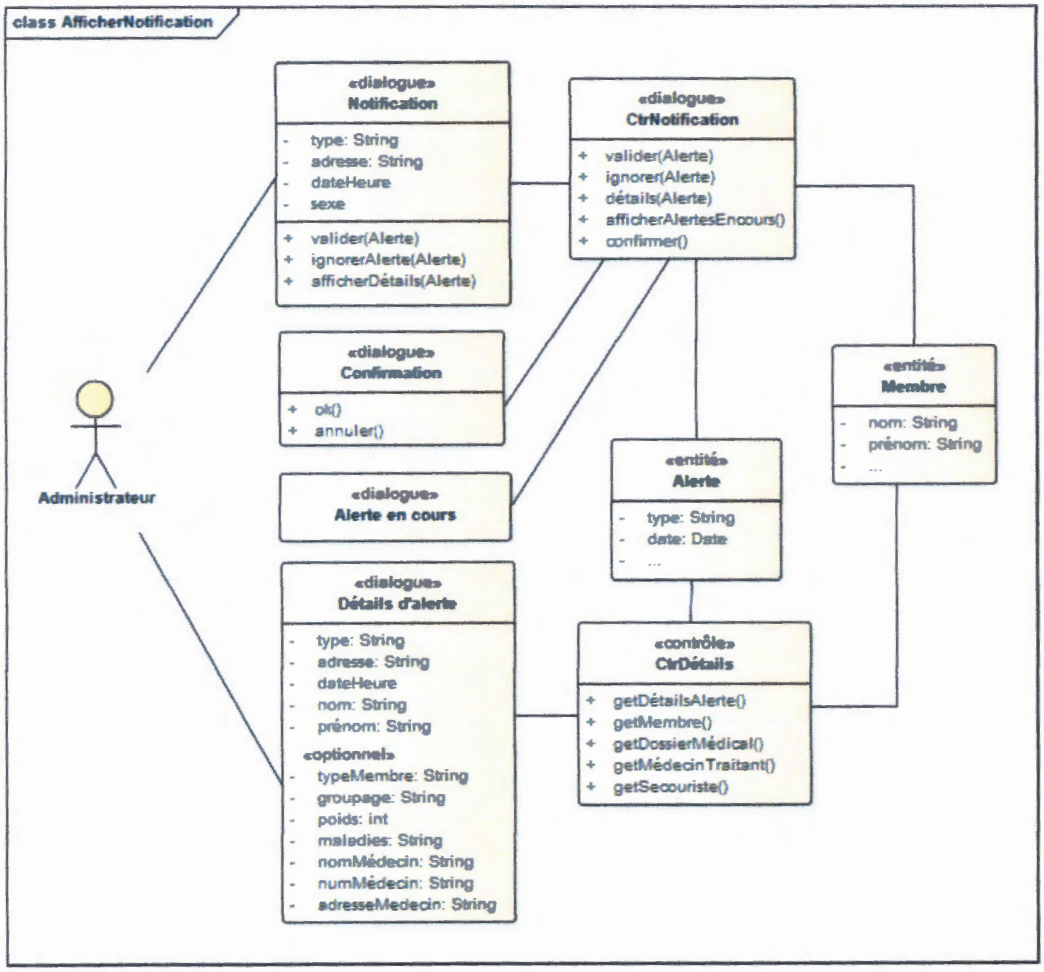


FIGURE 3.10 – DCP de l'UC *Afficher notification*

3.3 Diagramme de classe global

Nous passons maintenant au diagramme de classe global qui appartient à la phase de conception. Le diagramme de classe global est élaboré en regroupant et détaillant tous les concepts du domaine (entité) identifiés précédemment. Il est représenté dans la figure suivante.

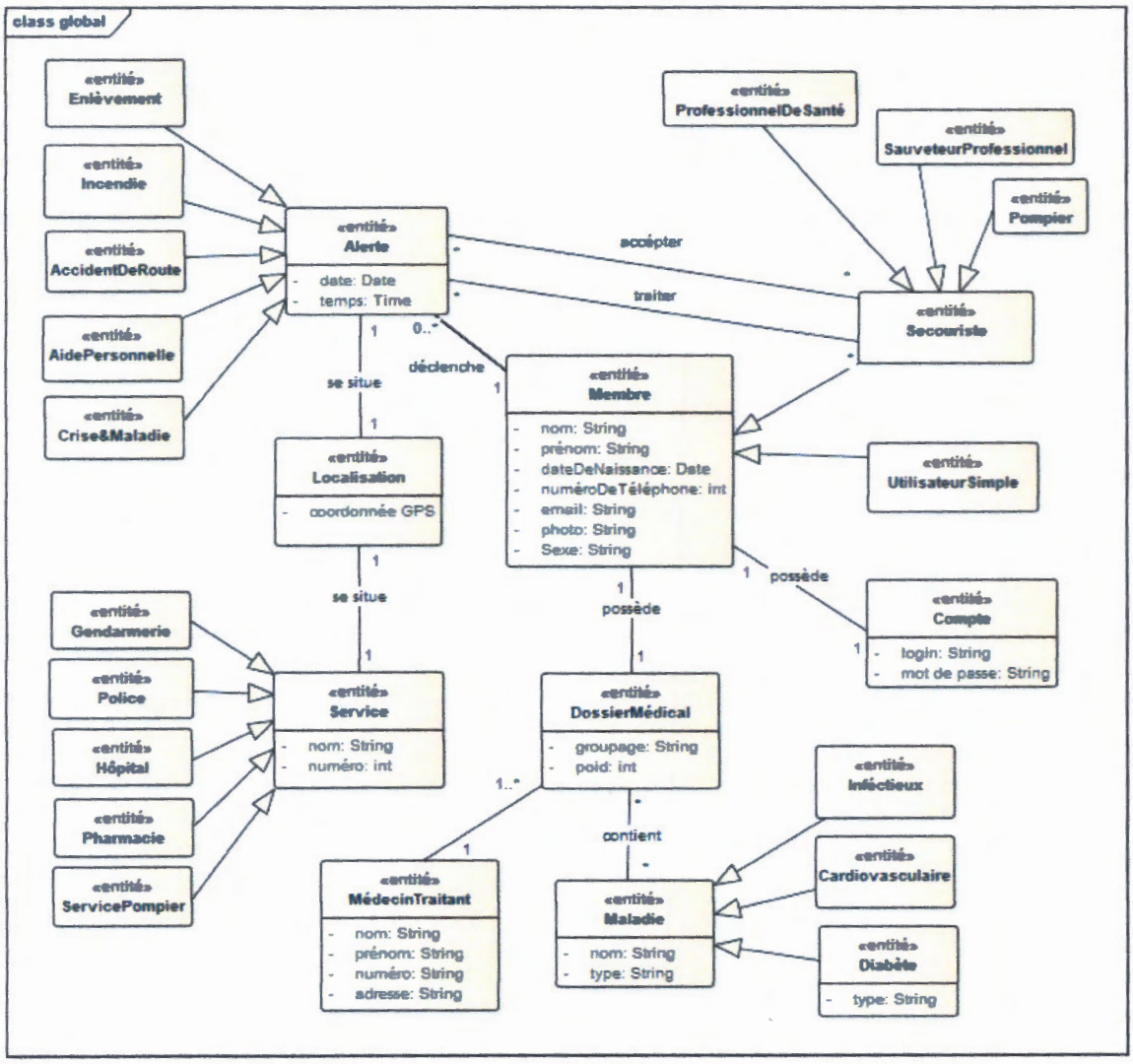


FIGURE 3.11 – Diagramme de classe global.

3.4 Diagramme d'état

Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets. Ils spécifient habituellement le comportement d'une instance de classeur (classe ou composant), mais parfois aussi le comportement interne d'autres éléments tels que les cas d'utilisation, les sous-systèmes et les méthodes [33].

Nous présentons par la suite un seul diagramme d'état qui est celui de la classe alerte. Comme le montre la Figure 3.12, une alerte est créée après avoir été signalé par un *Membre*. Lorsqu'une nouvelle alerte est reçue par l'*Administrateur*, il décide de la valider ou de l'ignorer. Si l'alerte est validée elle sera envoyée aux *Secouristes*. Ces derniers vont soit l'accepter (s'ils

veulent intervenir) soit l'ignorer. Si l'alerte est acceptée par au moins un *Secouriste* elle passe à l'état acceptée, sinon l'*Administrateur* change son état qui devient traitée après un certain temps.

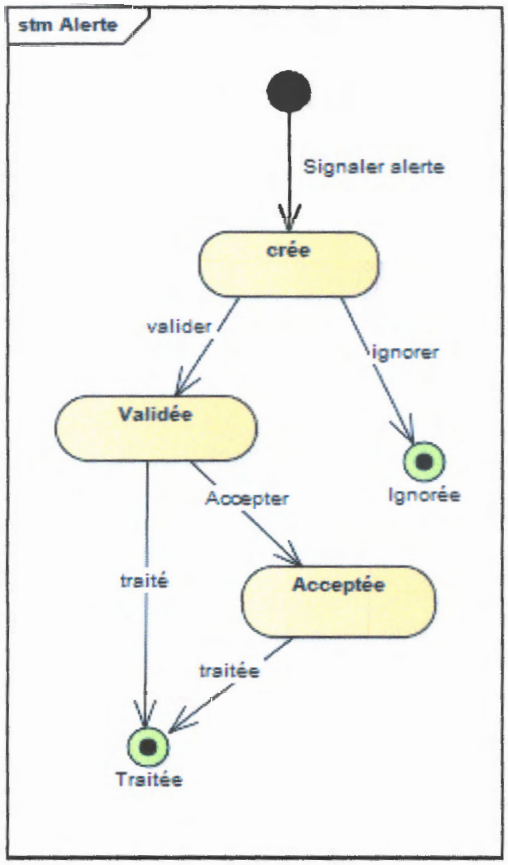


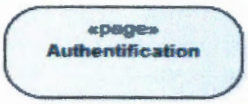
FIGURE 3.12 – Diagramme d'état de l'alerte

3.5 Diagramme de navigation

La Figure ci-dessous (figure 3.13) illustre le positionnement de diagramme de navigation dans la démarche. Les diagrammes de navigation sont des diagrammes dynamiques représentant de manière formelle l'ensemble des chemins possibles entre les principaux écrans proposés à l'utilisateur [12].
La modélisation de la navigation se réalise par des états et des transitions, en se basant sur les maquettes et les DCPs.

- Les *états* représentent les classes dialogues. Pour modéliser les différents concepts d'état, nous allons utiliser les quatre types suivants :

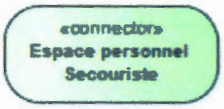
  – Une page complète du site ou de l'application (*page* avec la couleur gris).



– Un frame particulier à l'intérieur d'une page (*frame* avec la couleur bleu).

– Une erreur ou un comportement inattendu du système (*exception*).

– Une liaison vers un autre diagramme d'activité, pour des raisons de structuration et de lisibilité (*connector* avec la couleur vert).



- Les *transitions* entre états déclenchées par des événements et pouvant porter des conditions, pour représenter les actions IHM [12] .

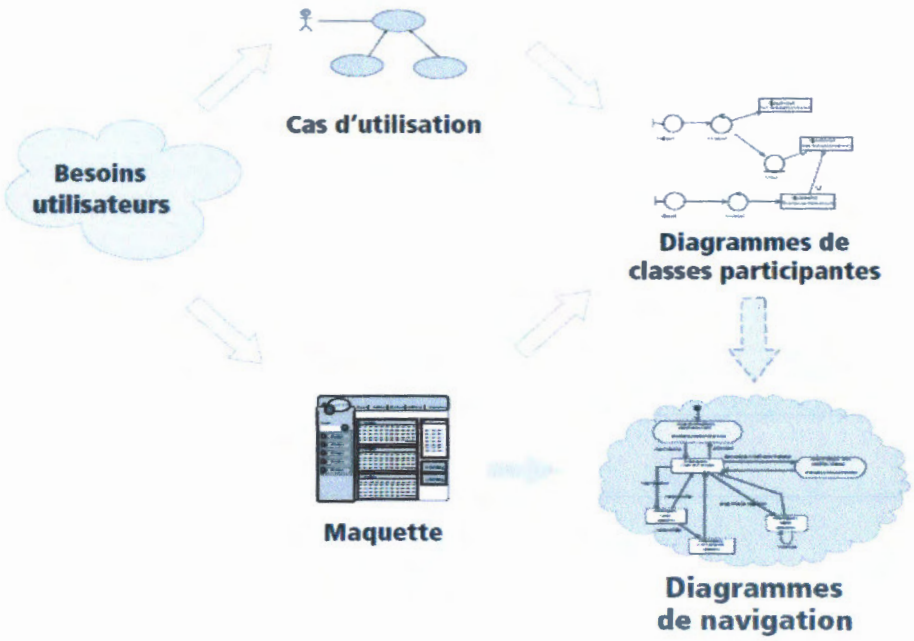


FIGURE 3.13 – Le positionnement de diagramme de navigation dans la démarche [12].

3.5.1 Diagrammes de navigation de notre système

Nous allons structurer la modélisation de la navigation par acteur. Premièrement, la navigation du *Membre* est complètement différente de celle du Personnel Administratif. Le

Membre utilise une application mobile et le Personnel Administratif utilise une application web.

1. **Diagramme de navigation du Personnel Administratif**

   Selon le diagramme suivant (Figure 3.14), le *Personnel Administratif* démarre par la page d'authentification du site web «http ://localhost :8080BeSafeServer/Login», il doit saisir son identifiant et son mot de passe correct. Suivant le résultat du contrôle effectué par le système, il sera dirigé soit vers la page d'accueil (soit en tant qu'*Administrateur* ou bien un *Webmaster*) (Figure 3.15), soit vers la page d'accueil de Webmaster (Figure 3.16).

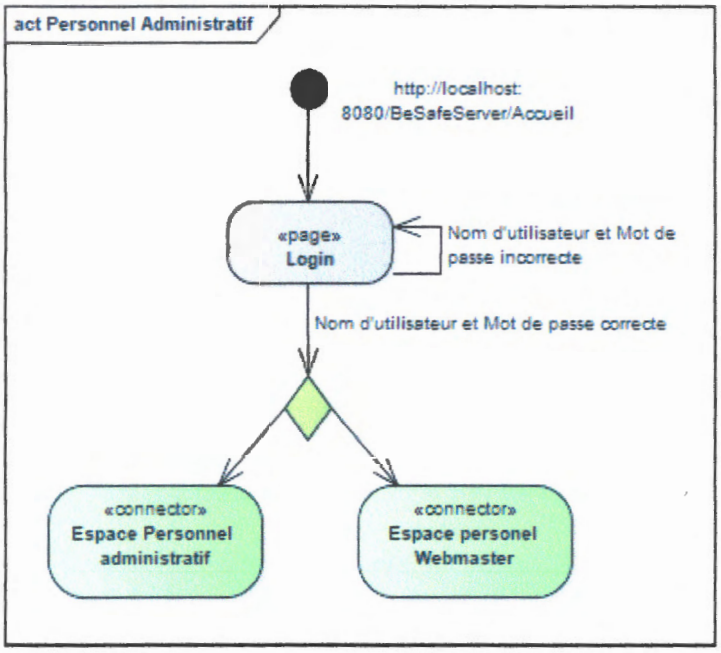


FIGURE 3.14 – Début de diagramme de navigation du *Personnel Administratif*.

(a) **Administrateur**

Comme le montre la Figure 3.15, à partir de *la page d'accueil*, l'*Administrateur* reçoit des notifications, il peut les valider ou les ignorer via le *frame* de confirmation, ou afficher leurs détails. En plus, il peut flâner dans un ensemble de *pages* : *alertes*, *secouristes*, *historique* et *statistique*.

La *notification* est un *frame* toujours présent dans *la page d'accueil*, *la page des alertes* et *la page de Secouriste*. Cette dernière est composée de 4 pages : *page d'alertes en cours*, *page d'alertes ignorées*, *page d'alertes traitées* et *la page de toutes les alertes* où l'*Administrateur* peut afficher tous les détails d'une alerte en cours ou déjà *traitée*, comme il peut changer l'état d'une alerte vers *traitée* en informant les *Secouristes* qui l'ont accepté.

La page secouristes est composée de 4 *pages* dont chacune affiche les détails d'un

type de secouriste : *page pompier*, *page professionnel de santé*, *page sauveteur professionnel* et *page de toutes les secouristes.*
L'*Administrateur* peut consulter toutes les anciennes alertes signalées en détail à partir de la *page d'historique.*

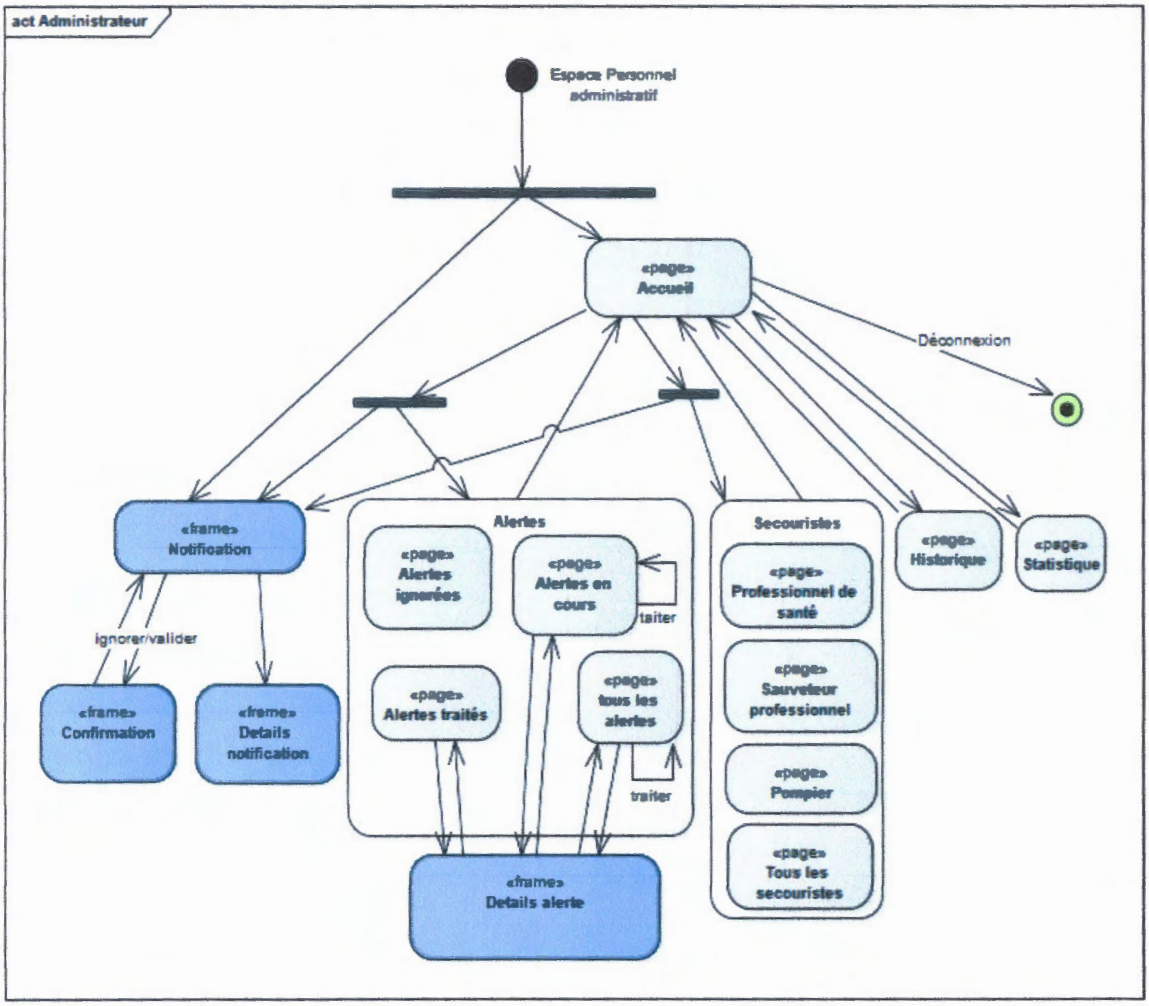


FIGURE 3.15 – Diagramme de navigation de *l'Administrateur.*

(b) **Webmaster**

Comme le montre la Figure 3.16, à partir de *la page d'accueil*, le Webmaster a la possibilité d'afficher tous les comptes des utilisateurs, en plus il peut naviguer entre les différentes autres pages.
La page d'activation permet au Webmaster de consulter les nouveaux comptes créés. Après la confirmation de validité des informations de l'utilisateur, il peut l'activer ou le désactiver.
la page secouristes est composée de 4 pages dont chacune affiche les détails d'un type de secouriste (*pompier*, *sauveteur professionnel*, *professionnel de santé*) et

la page utilisateur simple permet d'afficher les détails des utilisateurs simples. A partir des 4 pages composant la page secouriste el la page utilisateur simple, le Webmaster peut supprimer un compte d'utilisateur qui n'a pas respecté les conditions d'utilisation (si par exemple signaler des fausses alertes, etc). Ensuit le *frame de confirmation* s'affiche pour permettre au *Webmaster* la confirmation des opérations suivantes sur un compte : activer, désactiver et supprimer.

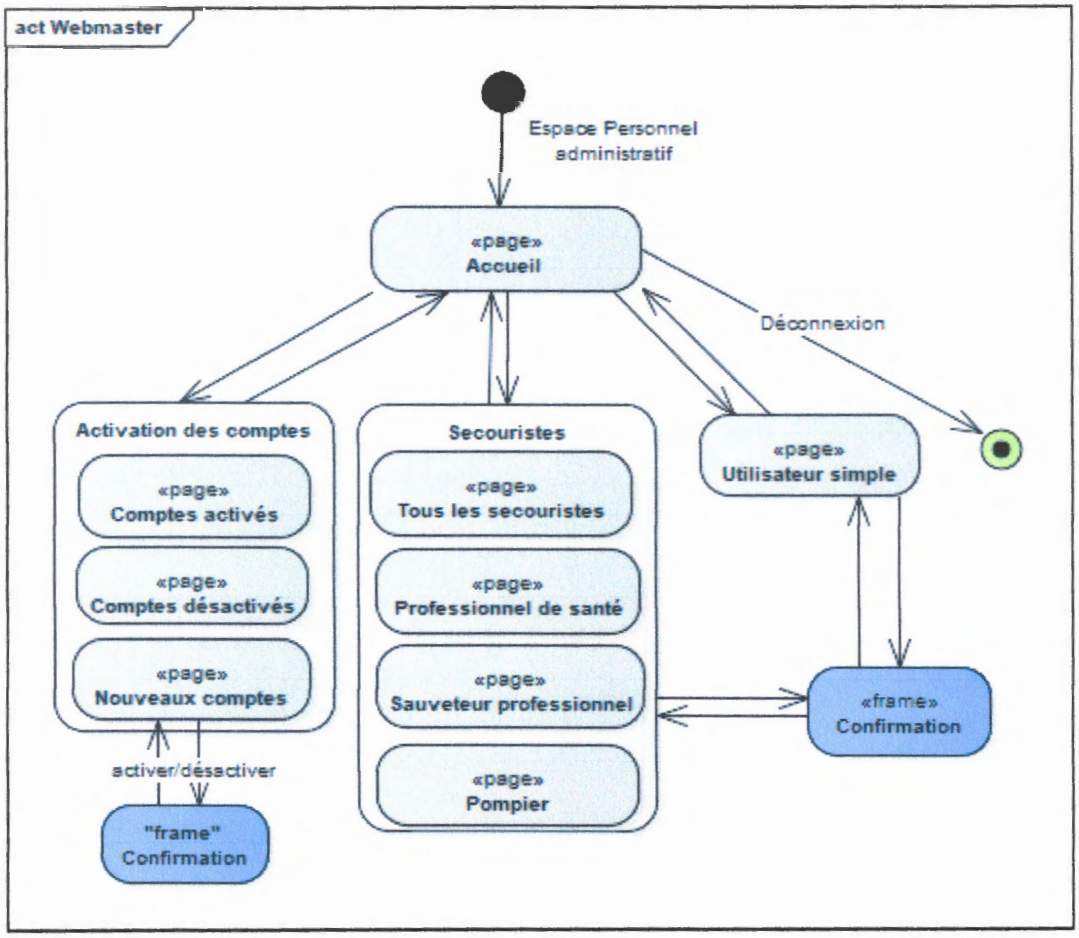


FIGURE 3.16 – Diagramme de navigation du *Webmaster*.

2. **Diagramme de navigation du Membre**

Comme le montre la Figure 3.17. Le *Membre* démarre par la page d'authentification de l'application BeSafe. Pour cela il doit d'abord saisir son identifiant et son mot de passe, suivant le résultat du contrôle effectué par le système, il sera dirigé soit vers sa *page d'accueil* (soit en tant que secouriste ou bien un utilisateur simple) soit se diriger de nouveau sur *la page d'authentification* avec un message d'erreur (login incorrecte).

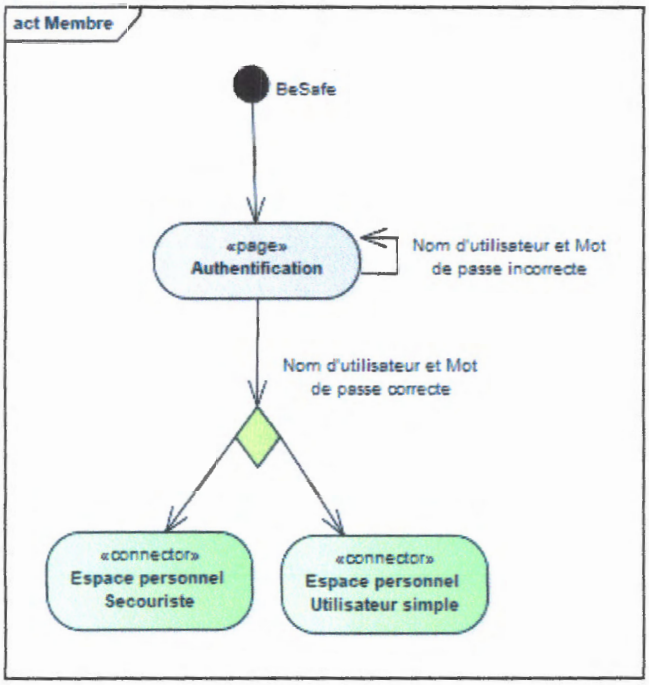


FIGURE 3.17 – Début de diagramme de navigation du *Membre*.

Le diagramme de navigation de l'utilisateur simple est le même que celui du secouriste sauf qu'il ne contient pas les éléments graphiques suivants : *page listes des alertes*, *frame notification d'alerte* et *page d'intervention*.
Comme le montre la Figure 3.18 qui présente le diagramme de navigation du secouriste. Une fois le secouriste est dirigé vers son espace personnel, il peut naviguer entre différentes pages. Par exemple, dans la *page d'accueil*, le secouriste a la possibilité d'afficher les services (hôpitaux, pharmacie, etc) sur la carte ou d'afficher la liste des alertes acceptées, comme il peut signaler des alertes selon le type à partir de la *page signaler une alerte* ou signaler un aide personnelle à partir de la *page aide personnelle*. Le secouriste reçoit des notifications d'alertes sur le *frame notification* d'alerte, s'il accepte une alerte il sera dirigé vers la *page d'intervention*. Dans la *page dossier médicale*, le secouriste à la possibilité de consulter ou modifier son dossier médicale. D'après la *page paramètre*, le secouriste peut activer/désactiver le mode *ne pas déranger*, il peut aller à *la page profil* pour consulter ou modifier ses informations personnelles comme il peut se déconnecter.

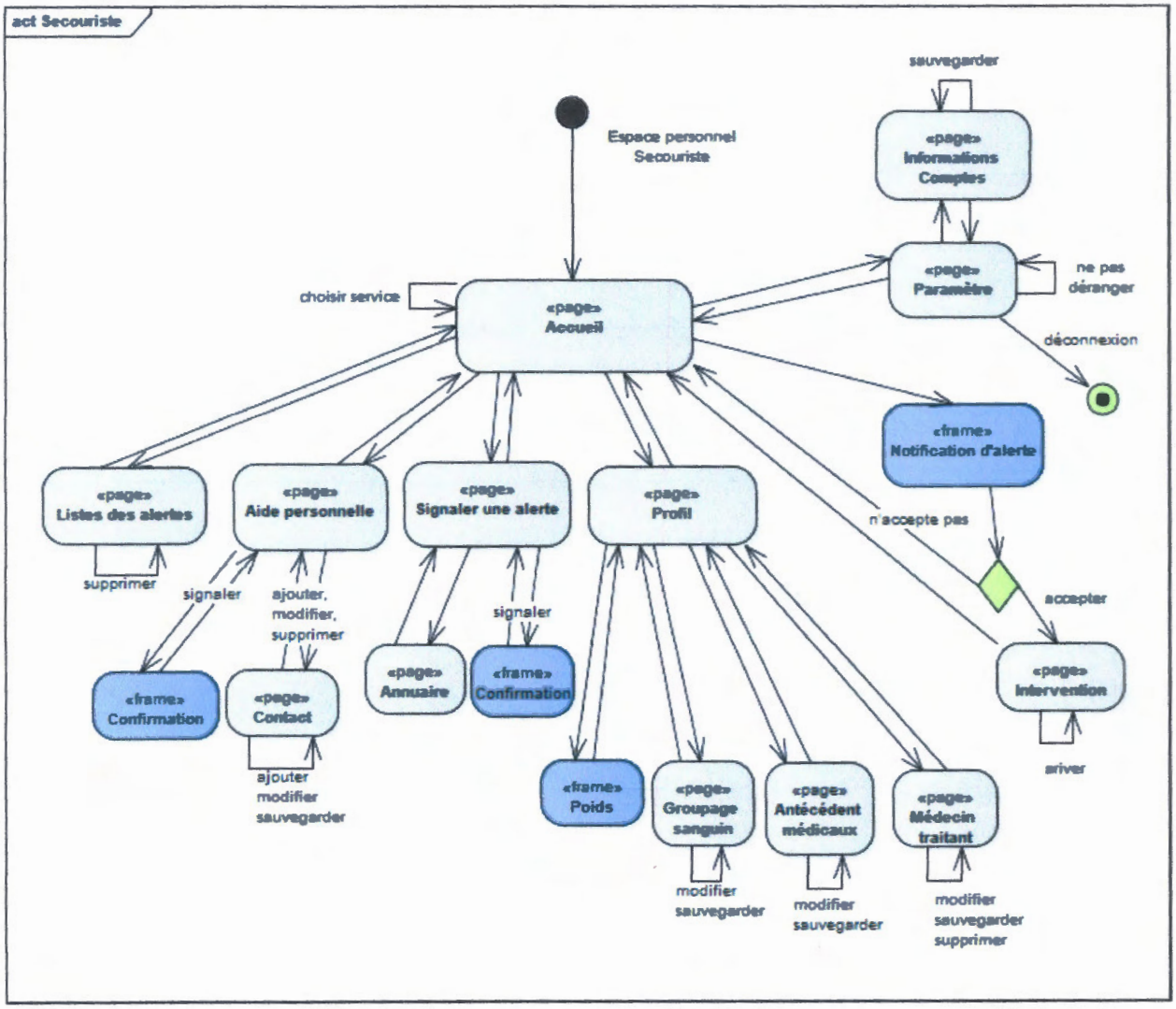


FIGURE 3.18 – Début de diagramme de navigation du *Secouriste*.

3.6 Conclusion

Dans ce chapitre, nous avons élaboré les diagrammes de classes participantes des cas d'utilisation de notre système. Cela a permis d'identifier les différents concepts du domaine : classes, attributs, opérations, etc, et aider à construire le diagramme de classe global. Ensuite, nous avons modélisé la partie dynamique du système à l'aide des diagrammes d'état et de navigation.

Chapitre 4

Réalisation

4.1 Introduction

Nous passons maintenant à la dernière partie qui est la réalisation de notre application, nous exposons les différents choix techniques : langages, environnements, et plateformes de développement utilisés. Ensuite, nous présentons l'architecture de notre application.

4.2 Choix techniques

Langage JAVA : Java est un langage de programmation orienté objet, développé par Sun Microsystems qui a été racheté par Oracle [25]. Outre son orientation objet, le langage Java a l'avantage d'être modulaire (on peut écrire des portions de code génériques, c'est-à-dire utilisables par plusieurs applications), rigoureux (la plupart des erreurs se produisent à la compilation et non à l'exécution) et portable (un même programme compilé peut s'exécuter sur différents environnements) [20].

HTML : (Hyper Text Markup Language) a fait son apparition dès 1991 lors du lancement du web, son rôle est de gérer et organiser le contenu. C'est donc en HTML qu'on peut écrire ce qui doit être affiché sur la page : du texte, des liens, des images [21].

CSS : (Cascading Style Sheets aussi appelées Feuilles de style) le concept feuille de style permet de définir l'apparence des textes (comme la police, la couleur, la taille, etc), ainsi que l'agencement de la page (comme les marges, l'arrière-plan, etc). CSS définit donc la présentation du document [22].

JavaScript : est un langage de script orienté objet principalement utilisé dans les pages HTML. A l'opposé des langages serveurs (qui s'exécutent sur le site), JavaScript est exécuté sur l'ordinateur de l'internaute par le navigateur lui-même. Ainsi, ce langage permet une interaction avec l'utilisateur en fonction de ses actions (lors du passage de la souris au-dessus d'un élément, du redimensionnement de la page, etc).

JSON : (JavaScript Object Notation - Notation Objet issue de JavaScript) est un format léger d'échange de données, facile à lire ou à écrire pour les humains. Il est généré ou analysé aisément par des machines et basé sur un sous-ensemble de langage de programmation JavaScript, c'est un format texte complètement indépendant de tout langage, mais utilisant

des conventions. Ces propriétés font de JSON un langage d'échange de données idéal [24].

Java EE : Le terme *Java EE* signifie Java Enterprise Edition, était anciennement raccourci en *J2EE*. Il fait quant à lui référence à une extension de la plate-forme standard, la plate-forme Java EE est construite sur le langage Java et la plate-forme Java SE, et elle y ajoute un grand nombre de bibliothèques remplissant tout un tas de fonctionnalités que la plate-forme standard ne remplit pas d'origine. L'objectif majeur de Java EE est de faciliter le développement d'applications web robustes et distribuées, déployées et exécutées sur un serveur d'applications [10].

Le Java Enterprise Edition, comme son nom l'indique, a été créé pour le développement d'applications d'entreprises, ses spécifications ont été pensées afin, notamment, de faciliter le travail en équipe sur un même projet : l'application est découpée en trois couches (selon le modèle MVC) et le serveur sur lequel tourne l'application est lui-même découpé en plusieurs niveaux [10].

Le serveur est composé d'un serveur web et d'un serveur d'application.

1. **Serveur HTTP**(Serveur web) : est un serveur qui gère exclusivement des requêtes HTTP. Il a pour rôle d'intercepter les requêtes Http, sur un port qui est par défaut 80, pour les traiter et générer ensuite des réponses Http [10].

2. **Serveur d'application** : est un environnement pour l'exécution du code java.

Le modèle MVC avec JEE est représenté dans la Figure 4.1 .

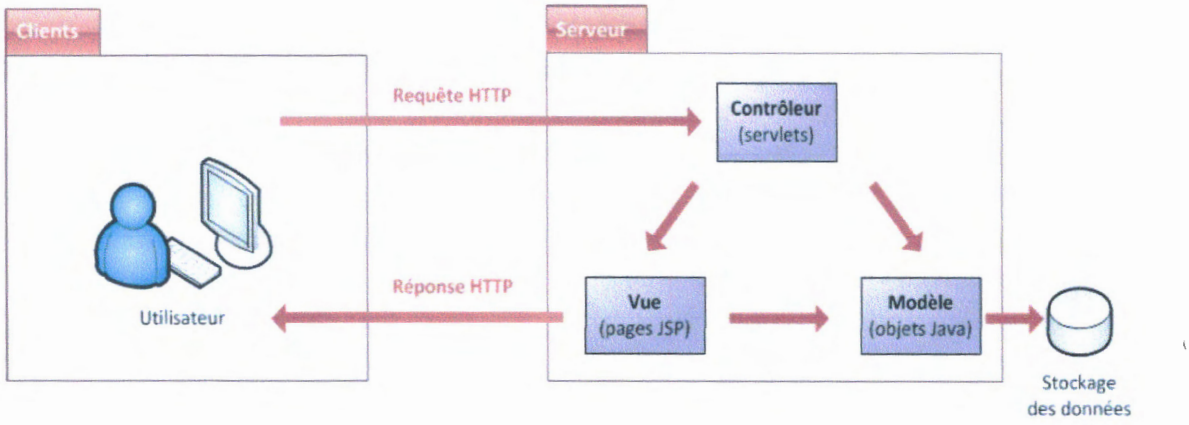


FIGURE 4.1 – MVC avec JEE [10].

Les couches composant une application web selon le modèle MVC sont :

1. **Modèle** : Dans le modèle, on trouve à la fois les données et les traitements à appliquer à ces données. Ce bloc contient donc des objets Java (JavaBean) d'une part, qui peuvent contenir des attributs (données) et des méthodes (traitements) qui leur sont propres, et un système capable de stocker des données d'autre part.

Le JavaBean, souvent raccourci en "bean", qui est un simple objet Java qui suit certaines contraintes, et représente généralement des données du monde réel.
Les JavaBeans possèdent plusieurs caractéristiques :

- **Les propriétés :** un bean est conçu pour être paramétrable. On appelle "propriétés" les champs non publics présents dans un bean. Qu'elles soient de type primitif ou objets, les propriétés permettent de paramétrer le bean, en y stockant des données.
- **La sérialisation :** un bean est conçu pour pouvoir être persistant. La sérialisation est un processus qui permet de sauvegarder l'état d'un bean, et donne ainsi la possibilité de le restaurer par la suite. Ce mécanisme permet une persistance des données, voire de l'application elle-même.
- **La réutilisation :** un bean est un composant conçu pour être réutilisable. Ne contenant que des données ou du code métier, un tel composant n'a en effet pas de lien direct avec la couche de présentation, et peut également être distant de la couche d'accès aux données, c'est cette indépendance qui lui donne ce caractère réutilisable.
- **L'introspection :** un bean est conçu pour être paramétrable de manière dynamique. L'introspection est un processus qui permet de connaître le contenu d'un composant (attributs, méthodes et événements) de manière dynamique, sans disposer de son code source. C'est ce processus, couplé à certaines règles de normalisation, qui rend possible une découverte et un paramétrage dynamique du bean.

Un bean doit également avoir la structure suivante :

- Doit être une classe publique.
- Doit avoir au moins un constructeur par défaut, public et sans paramètres. Java l'ajoutera de lui-même si aucun constructeur n'est explicité.
- Peut implémenter l'interface Serializable, il devient ainsi persistant et son état peut être sauvegardé.
- Ne doit pas avoir de champs publics.
- Peut définir des propriétés (des champs non publics), qui doivent être accessibles via des méthodes publiques getter et setter, suivant des règles de nommage [10].

2. **Vue (JSP)** : est une technologie pour le développement des pages web incluant du contenu dynamique. Contrairement à une page HTML qui ne contient que du contenu statique qui reste par définition toujours le même, JSP peut changer selon l'identité du visiteur, de son navigateur Internet, de l'heure, de la configuration du système, des actions du visiteur, etc.

Une page JSP contient des balises standards, comme du HTML (ou du WML, XML, etc), comme toute page web normale, pourtant, une page JSP contient aussi des éléments JSP spécifiques (scriptlets), permettant au serveur l'insertion dynamique de contenu (contenu de BDD, préférences du visiteur). Lorsqu'un utilisateur accède à une page JSP, le serveur exécute les éléments JSP, fusionne les résultats avec les parties statiques de la page, et envoie le tout au navigateur [23].

3. **Contrôleur (Servlet)** : Technologie Java utilisée pour effectuer des traitements coté serveur en réponse aux requêtes provenant en général de poste clients distants. Bien que les Servlets puissent répondre à n'importe quel type de requête, elles sont généralement employées pour répondre à des requêtes de type HTTP et qui permettent de retourner dynamiquement des pages HTML [23].

Eclipse : Eclipse IDE [26] est un environnement de développement intégré libre, extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement qui supporte un ensemble vaste de langages de programmation .
La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plug-in. Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, par exemple IBM Lotus Notes 8, IBM Symphony ou Websphere Studio Application Developer [26].
Dans le cadre de notre projet, nous avons utilisé Eclipse JEE, qui est la version Eclipse pour JEE.

Apache : Apache est un serveur web Open Source, gratuit, très populaire. Apache est l'un des serveurs web les plus anciens et les plus fiables [27].

Apache Tomcat : Il s'agit du serveur d'application Java le plus utilisé au monde, il est un serveur léger (gratuit, libre, multiplateforme et assez complet) conçu pour servir les applications Java (Servlets, JSP, etc). Apache Tomcat est une implémentation open source d'un conteneur web intègre un serveur web qui permet donc d'exécuter des applications web reposant sur les technologies servlets et JSP[27].

Volley : est une bibliothèque HTTP qui simplifie et accélère la mise en réseau des applications Android. Volley est disponible sur GitHub [1], Il offre les avantages suivants :

- Programmation automatique des requêtes réseau.
- Plusieurs connexions réseau simultanées.
- Mise en cache transparente des réponses disque et mémoire avec cohérence de cache HTTP standard.
- Prise en charge de la hiérarchisation des demandes.

1. GitHub : https ://github.com/

- Demande d'annulation de l'API : annuler une seule demande ou définir des blocs ou des étendues de requêtes à annuler.
- Facilité de personnalisation, par exemple, pour réessayer et annuler.
- Une commande robuste qui facilite le remplissage correct de l'interface utilisateur avec les données extraites de manière asynchrone à partir du réseau.
- Outils de débogage et de traçage.

Volley excelle dans les opérations de type RPC utilisées pour remplir une interface utilisateur, telles que l'extraction d'une page de résultats de recherche sous forme de données structurées. Il s'intègre facilement à tous les protocoles et prend en charge les chaînes brutes, les images et les JSON.
En offrant une prise en charge intégrée des fonctionnalités dont l'utilisateur a besoin, Volley libère l'utilisateur de l'écriture du code standard et lui permet de se concentrer sur la logique spécifique de son application.
Volley ne convient pas aux opérations de téléchargement ou de streaming importantes, car il conserve toutes les réponses en mémoire lors de l'analyse. La bibliothèque de base de Volley est développée sur GitHub et contient le pipeline de distribution des requêtes principales ainsi qu'un ensemble d'utilitaires communément disponibles dans la "boîte à outils" de Volley. Le moyen le plus simple d'ajouter Volley au projet consiste à ajouter la dépendance suivante au fichier build.gradle de l'application :

- dependences {*implementation'com.android.volley* : *volley* : 1.1.1'} .

MYSQL : (My Structured Query Language) est un système de gestion de base de données relationnelle open source. Il est basé sur le langage de requête de structure (SQL), qui est utilisé pour ajouter, supprimer et modifier des informations dans la base de données. Les commandes SQL standard, telles que "ADD, DROP, INSERT et UPDATE" peuvent être utilisées avec MySQL [28].

WampServer : est une plateforme du développement web de type WAMP, servant à faire fonctionner localement (sans se connecter à un serveur externe) des scripts PHP. WampServer n'est pas en soi un logiciel, mais un environnement comprenant deux serveurs (Apache et MySQL), un interpréteur de script (PHP), ainsi qu'une administration pour les deux bases *SQL PhpMyAdmin* et *SQLiteManager* [29].

Android Studio : est l'environnement du développement intégré (IDE) pour le développement d'applications Android, basé sur IntelliJ IDEA. Outre le puissant éditeur de code et les outils du développement d'IntelliJ, Android Studio offre encore plus de fonctionnalités qui améliorent la productivité lors de la création d'applications Android [30].

SDK Android : (kit de développement logiciel) est un ensemble d'outils du développement permettant de développer des applications pour la plate-forme Android.

AVD Android : (Appareil Virtuel Android) est une configuration d'émulateur qui permet aux développeurs de tester l'application en simulant les capacités réelles de l'appareil. Nous pouvons configurer l'AVD en spécifiant les options matérielles et logicielles. AVD Manager permet de créer et de gérer facilement l'AVD avec son interface graphique. Nous pouvons créer autant d'AVD que nécessaire, en fonction des types d'appareils que nous voulons tester.

AJAX : (Asynchronous JavaScript and XML) n'est pas une technologie mais plutôt une architecture technique qui permet d'interroger un serveur de manière asynchrone pour obtenir des informations permettant de mettre à jour dynamiquement la page HTML. Le but principal d'AJAX est d'éviter le rechargement complet d'une page pour n'en mettre qu'une partie à jour. Ceci permet donc d'améliorer l'interactivité et le dynamisme de l'application web qui le met en œuvre.

Entreprise Architect : pour Réaliser les diagrammes UML qui ont servis a modéliser notre application, nous avons utilisé Entreprise Architect. Entreprise Architect est un outil d'analyse et de création UML, couvrant le développement de logiciels du rassemblement d'exigences, en passant par les étapes d'analyse, les modèles de conception et les étapes de test et d'entretien. Cet outil graphique basé sur Windows, peut être utilisé par plusieurs personnes et conçu pour aider à construire des logiciels faciles à mettre à jour. Il comprend un outil de production de documentation souple et de haute qualité [31].

FCM : (Firebase Cloud Messaging) est une solution de messagerie multi plateforme qui permet de livrer des messages de manière fiable, gratuite. À l'aide de FCM, on peut indiquer à une application cliente qu'un nouveau courrier électronique ou d'autres données sont disponibles pour la synchronisation, avec la possibilité d'envoyer des messages de notification pour encourager le réengagement et la rétention des utilisateurs. Pour les cas d'utilisation tels que la messagerie instantanée, un message peut transférer une charge utile maximale de 4 Ko vers une application cliente.
Une implémentation FCM comprend deux composants principaux pour l'envoi et la réception :

- Un environnement sécurisé tel que Cloud Fonctions de Firebase ou un serveur d'applications sur lequel créer, cibler et envoyer des messages.
- Application cliente iOS, Android ou web (JavaScript) recevant des messages.

1. **Inscription à la FCM**
   Une application cliente doit d'abord s'inscrire auprès de FCM avant que la messagerie puisse avoir lieu. L'application cliente doit effectuer les étapes d'enregistrement présentées dans la Figure 4.2.

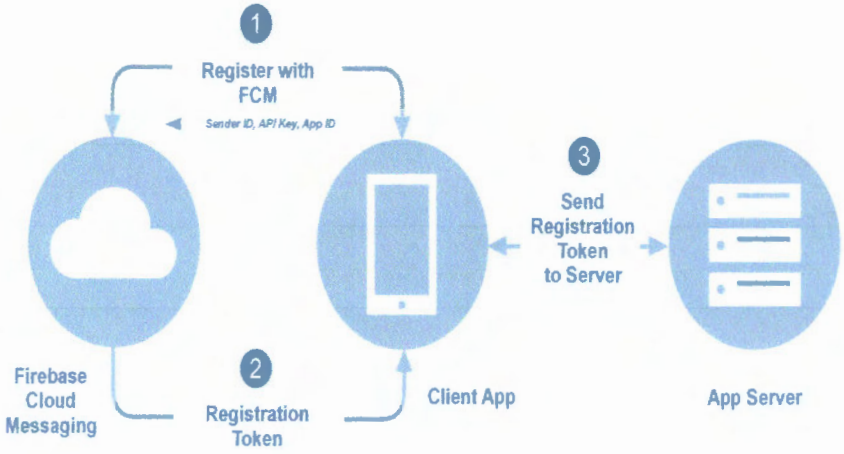


FIGURE 4.2 – Inscription à FCM [32].

1) L'application cliente contacte FCM pour obtenir un jeton d'enregistrement en transmettant l'ID de l'expéditeur, la clé API et l'ID de l'application à FCM.
2) FCM renvoie un jeton d'enregistrement à l'application cliente.
3) L'application cliente (éventuellement) transmet le jeton d'enregistrement au serveur d'applications.

2. **Messagerie en aval**
La Figure 4.3 suivant illustre comment FCM stocke et transfère les messages en aval :

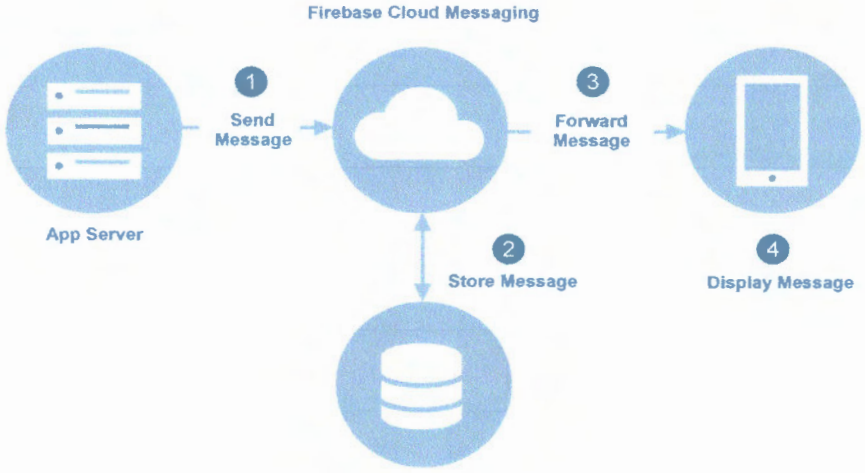


FIGURE 4.3 – Envoie de message FCM [32].

Lorsque le serveur d'applications envoie un message en aval à l'application cliente, il utilise les étapes suivantes énumérées dans le diagramme ci-dessus :
1) Le serveur d'applications envoie le message à FCM.

2) Si le périphérique client n'est pas disponible, le serveur FCM stocke le message dans une file d'attente pour une transmission ultérieure. Les messages sont conservés dans la mémoire FCM pendant 4 semaines maximum.

3) Lorsque le périphérique client est disponible, FCM transfère le message à l'application cliente sur ce périphérique.

4) L'application cliente reçoit le message de FCM, le traite et l'affiche à l'utilisateur, par exemple, si le message est une notification à distance, il est présenté à l'utilisateur dans la zone de notification [32].

4.3 Architecture du système

Après avoir présenté les différents choix techniques nous allons détailler l'architecture du système. Cette dernière est présenté dans la Figure 4.4. Elle représente l'implémentation de l'architecture client/serveur en utilisant la technologie JEE web et en respectant le modèle MVC.

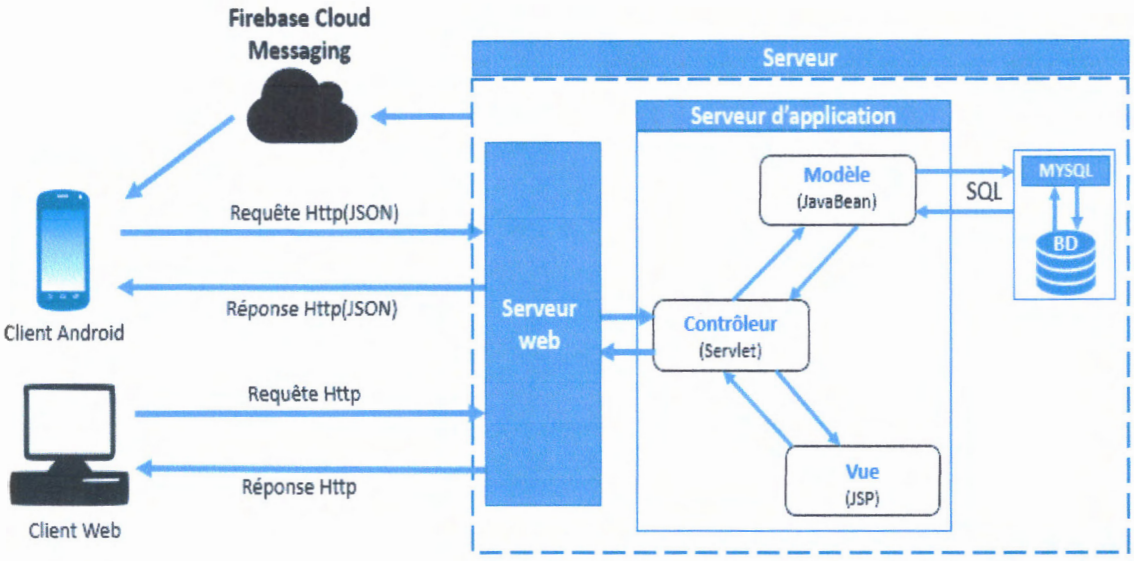


FIGURE 4.4 – Architecture global de notre application.

1. **Clients**

   La communication entre le client et le serveur débute par l'envoie des requêtes du client au contrôleur (Servlet). Ce dernier envoie les données au modèle (JavaBean) pour le traitement, une fois les données traitées, le contrôleur renvoie les résultats au client. Nous avons deux types de client :

   - **Client Android** : application qui tourne sur smartphone au-dessus du système Android développé en java en utilisant la bibliothèque volley.
   - **Client web** : application web développée en java.

2. **Serveur**
   Nous avons utilisé Appache Tomcat 8.0.

3. **Protocole de communication**
   Dans ce travail, nous nous somme basés sur le protocole HTTP pour réaliser la communication entre la partie cliente mobile et le serveur web (Le HTTP est un protocole qui définit la communication entre un serveur et un client).

4. **Format d'échange de données**
   Nous avons utilisé le format JSON pour échanger les données entre le client Android et le serveur.

5. **Gestion des notifications**
   Afin de gérer l'envoie de notification du serveur vers le client Android nous avons utilisé FCM.

4.4 Difficultés rencontrées

- Au début du projet nous avons utilisé Genymotion [2] qui est un émulateur Android complet pour ordinateur, mais il est très lourd et il prend beaucoup de temps pour l'exécution, donc nous avons utilisé le smartphone pour éviter ce problème.

- Google Maps est un service de cartographie en ligne, Mais malheureusement elle est devenue payante « Vous aurez besoin d'une clé API valide et d'un compte de facturation Google Cloud Platform pour accéder à nos produits principaux » précise Google sur son blog dédié. La meilleure méthode pour utiliser la carte géographique est avec «Maps JavaScript API» et pour afficher l'itinéraire est avec «Directions API» donc ça nous a fait un problème.

4.5 Conclusion

Au cours de ce dernier chapitre nous avons présenté les outils, les plateformes, les langages de programmation qui nous ont aidés à concevoir notre application et la représentation de l'architecture global de l'application.

2. Genymotion : https ://www.genymotion.com/.

Conclusion générale

Dans le contexte de ce projet de fin d'études, nous avons développé une application mobile dont l'objectif est de faciliter les opérations de secours. L'application Vise principalement à intégrer le public et les secouristes bénévoles dans l'opération de secours afin de permettre une intervention rapide et efficace. L'application contient deux parties, une partie web et une partie mobile. Elle a été modélisée en utilisant le langage UML suivant une approche ayant été conçue principalement pour les applications web. La réalisation a été faite en utilisant une combinaison de technologies et de langages.

La réalisation de ce travail nous a été d'une grande utilité dans la mesure où elle nous a permis d'approfondir nos connaissances théoriques dans le domaine de développement web et mobile. Nous avons aussi acquis une expérience technique avec un ensemble de langages, méthodes et technologies tels que UML, processus simplifié et les plates-formes Android, Java EE, WampServer et FCM, etc. Les bénéfices de cette expérience n'ont pas été uniquement techniques mais aussi sur le plan social.

Comme perspectives, nous avons l'intention de terminer l'implémentation certaines fonctionnalités telles que les statistiques et la visualisation sur la carte géographique. Enfin, nous souhaitons que notre application soit adoptée par les services de secours publics et qu'elle soit utile.

Bibliographie

[1] Brian LeRoux et Nitobi André Charland. *Mobile application development : Web vs native.* Article development led by queue.acm.org.

[2] Supinfo. https://www.supinfo.com/articles/single/145-application-mobile-native-web-hybride.

[3] Slideshare. https://fr.slideshare.net/Rouinihouss/introduction-aux-systmes-dexploitation-mobile?next_slideshow=1.

[4] Apple. https://www.apple.com/.

[5] Android. https://www.android.com/.

[6] Microsoft. https://www.microsoft.com/.

[7] Blackberry. https://www.blackberry.com/.

[8] Socialcompare. https://socialcompare.com/fr/comparison/mobile-os-comparison-developer-view.

[9] Zdnet. https://www.zdnet.fr/actualites/chiffres-cles-les-os-pour-smartphones-3979024 htm.

[10] Coyote. *Créez votre application web avec Java EE.* Openclassrooms, (février 2013).

[11] Ideematic. https://www.ideematic.com/dictionnaire-digital/application-web/.

[12] Pascal Roques. *UML2 Modéliser une application web.* EYROLLES, (juin 2008).

[13] Le point. https://www.lepoint.fr/societe/des-pompiers-lancent-une-application-pour-des 23.php.

[14] Permis de sauver. https://www.permisdesauver.info/.

[15] Sauvlife. https://sauvlife.fr/.

[16] Afpr-premiers répondants. https://www.afprappli.com/.

[17] W3. https://www.w3.org/Style/CSS/#specs.

[18] javascript. https://www.javascript.com/.

[19] Php. https://php.net.

[20] Gauthier Picard et Laurent Vercouter. *Initiation à la programmation orientée-objet avec le langage Java.*

[21] Mathieu Nebra. *Apprenez à créer votre site web avec HTML5 et CSS3.* Openclassrooms, (décembre 2011).

[22] Brice Canvel Claude Petitpierre, André Maurer. *HTML-CSS.* (Automne 2010).

[23] Michaël TRANCHANT. *Java WebServer Tomcat, JBoss, JRun, JOnAS.* (Décembre 2008).

[24] Présentation de json. https://www.json.org/.

[25] Oracle. https://www.oracle.com/fr/.

[26] Eclipse foundation. https://www.eclipse.org/.

[27] Apache Software Foundation. Apache tomcat. http://tomcat.apache.org/.

[28] Mysql. https://www.mysql.com/fr/.

[29] Wampserver. http://www.wampserver.com/.

[30] Google Developers. Developers. https://developer.android.com/.

[31] Sparx systems. https://sparxsystems.com/.

[32] Google Developers. Firebase. https://firebase.google.com/.

[33] Laurent AUDIBERT. *UML 2 de l'apprentissage à la pratique.* Ellipses, (octobre 2014).