République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté de Science
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : *Système d'Information et de Connaissances (S.I.C)*

Thème

Gestion des interventions de protection civile en utilisant une base de données NoSQL –MongoDB -

Réalisé par :

- Mebarki Mohammed Anas
- Nabi Zakaria

Présenté le 25 Juin 2020 devant le jury composé de MM.

- *ABDERRAHIM M.A* *(Président)*
- *EL YEBDRI Z.* *(Encadreur)*
- *BOUDEFLA M.A* *(Examinateur)*

Année universitaire : 2019-2020

Remerciements

Avant tout nous remercions Allah qui nous a donné le courage et la volonté pour accomplir ce travail.

Nous remercions notre professeur et encadreur M. El Yebdri Zeyneb, c'est grâce à elle que nous avons pu parcourir ce chemin et arriver à un tel résultat, merci Madame pour votre aide, encouragement, et pour vos conseils qui ont un grand impact dans la réussite de notre projet.

Nous tenons à remercier aussi les membres du Jury M. Boudefla Amine et M. Abderahim Med Amine qui nous ont honoré par leur présence et accepté d'évaluer notre travail.

Et bien sûr nous remercions nos parents pour leurs soutiens, les enseignements du département d'informatique pour leurs efforts durant tous notre chemin universitaire.

Dédicaces

C'est avec profonde gratitude et sincères paroles, que je dédie ce travail de fin d'études à mes chers parents pour leurs sacrifies et l'amour, qu'ils m'ont donné, Dieu leurs prête santé et longue vie.

A mon cher cousin Iheb qui m'a toujours encouragé ;

A mes chers frères,

A mon binôme pour tout ce qu'il a fait pour la réussite de ce travail ;

A tout membre de ma famille.

Mebarki Mohammed Anas

A mes chers parents, aucun hommage ne pourrait être à la hauteur de l'amour Dont ils ne cessent de me combler. Que dieu leur procure bonne santé et longue vie.

A mon binôme Anas pour son sérieux et détermination.

A tous mes amis, En témoignage de l'amitié sincère qui nous a liée. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de réussite.

A tous les gens qui ont cru en moi et qui me donnent l'envie d'aller en avant, Je vous remercie tous, votre soutien et vos encouragements me donnent la force de continuer.

Nabi Zakaria

Sommaire

Remerciements 2
Dédicaces 3
Sommaire 5
Liste des figures 7
INTRODUCTION GENERALE 8
Introduction générale 9
1 . CONTEXTE DU TRAVAIL 12
1.1 Introduction 13
1.2 Protection civile 13
1.3 Organigramme de la protection civile : 14
1.4 Centre de coordination opérationnelle (CCO) 14
1.5 Gestion d'intervention : 14
1.6 Etude de l'existant 15
1.7 Critique de l'existant 15
1.8 Conclusion 16
2 Base de données NoSQL 17
2.1 Introduction 18
2.2 Base de données NoSQL 18
2.3 Limites des bases de données relationnelles 18
2.4 Classification des bases de données NoSQL 19
2.4.1 Orienté clé – valeur 19
2.4.2 Orienté colonnes 19
2.4.3 Orientée graphes 19
2.4.4 Orientée documents 19
2.5 Comparaison entre les différents base NoSQL 20
2.6 Base de données orienté document -MongoDB- 21
2.7 Mongoose 21
2.8 Conclusion 21
3 . CONCEPTION DU SYSTEME « ProtectLife » 22
3.1 Introduction 23
3.2 Analyse des besoins 23
3.3 Présentation de l'UML 24
3.3.1 Diagramme de cas d'utilisation 24
3.3.2 Diagramme de séquence 26
3.3.3 Diagramme de classe 28
3.3.4 Modèle logique 30
3.4 Conclusion : 31
4 . IMPLEMENTATION DU SYSTEME 32
4.1 Introduction 33
4.2 Architecture du système ProtectLife 33
4.2.1 SERVEUR REST API 35
4.2.2 Applications web du ProtectLife 41
4.2.3 Applications Mobiles Android 46
4.3 Présentation d'un cas réel d'une intervention en utilisant notre système 50
4.4 Conclusion 54
CONCLUSION GENERALE 55

Résumé 58
Abstract.....58
ملـــخـــص.....**Erreur ! Signet non défini.**

Liste des figures

Figure 1 Organigramme de la protection civile 14
Figure 2 types des bases de données NoSQL [4] 19
Figure 3 L'utilisation des catégories de NoSQL 20
Figure 4 Diagramme de cas d'utilisation de l'Administrateur de l'unité principale 24
Figure 5 Diagramme de cas d'utilisation de l'Administrateur de l'unité secondaire 25
Figure 6 Diagramme de cas d'utilisation de l'agent de CCO de l'unité principale 25
Figure 7 Diagramme de cas d'utilisation de l'agent de CCO de l'unité secondaire 26
Figure 8 Diagramme de séquence de l'opération de l'agent de CCO 27
Figure 9 Diagramme de séquence d'une opération d'appel d'urgence 28
Figure 10 Architecture de l'application 33
Figure 11 Communication entre NodeJs et MongoDB 35
Figure 12 Cycle d'une requête 38
Figure 13 E/S NodeJs vs PHP [16] 40
Figure 14 NodeJs requêtes par seconds[17] 40
Figure 15 Unité principale - les agents 41
Figure 16 Unité principale - Nouveau agent 42
Figure 17 Unité principale - Modifier agent 42
Figure 18 List agents pour l'agent de CCO - unité principale 43
Figure 19 Page d'intervention - Unité principale 43
Figure 20 VueJs vs JQuery vs ReactJs [18] 45
Figure 21 Agent_ProtectLife Application 47
Figure 22 Nouvelle intervention - Unité principale (Partie 1) 51
Figure 23 Nouvelle intervention - Unité principale (Partie 2) 51
Figure 24 Intervention Reçue - Unité Secondaire (Partie 1) 52
Figure 25 Envoyer l'intervention au chef d'agrée 52
Figure 26 Intervention reçue - Agent_ProtectLife 53
Figure 27 Opération effectuer par le chef d'agrée - Agent_ProtectLife 54

INTRODUCTION GENERALE

Introduction générale

Contexte

Il ne fait désormais plus aucun doute que l'informatique est la révolution la plus importante et la plus innovante qui a marqué la vie de l'humanité moderne. Ceci s'applique aussi sur le domaine de la localisation de personne qui s'accélère de plus en plus.

Tellement la vie humaine est tellement importante, de nos jours, nous exploitons la technologie informatique dans tous les domaines pour sauver la vie des gens. La protection civile représente un élément clé dans notre société.

Notre travail, consiste à réaliser une solution dédiée pour automatiser le système de la protection civile afin de l'utiliser dans leurs services

Problématique et Objectifs

Chaque jour, le centre de coordination opérationnelle (CCO) de chaque wilaya reçoit 1 000 appels en moyenne qui demandent de secours. Pour chaque appel le CCO de l'unité principale doit appeler l'unité la plus proche de l'appelant pour intervenir.

Cependant, Un rapport doit être rempli par l'agent de CCO de l'unité principale suite aux étapes de l'intervention sur une feuille brouillon afin de remplir toutes les informations sur un exemplaire en Microsoft Word. Cette opération est faite en deux étapes bouclées :

- Le chef d'agrée appelle l'agent de CCO de son unité (Unité secondaire) pour lui donner les nouvelles informations.
- L'agent de CCO de l'unité secondaire appelle l'agent de CCO de l'unité principale pour lui donner les nouvelles informations données par le chef d'agrée.

Ces rapports sont envoyés par fixe au centre de protection civile d'Alger, et sont enregistrés dans une archive de papier. Pour cela, la gestion de ces dossiers est confrontée à des pertes d'informations, alors que ces données sont importantes.

Ainsi, l'agent de CCO de l'unité principale doit communiquer avec les équipes intervenantes pour prendre les nouvelles informations, alors qu'il doit être libéré pour donner des instructions et des conseils à l'appelant ou répondre aux nouveaux appels.

Aussi, l'intervention est donnée à l'unité du secteur de l'appelant en négligeant les calculs de la meilleure distance et le trafic routier, alors que ces deux derniers sont obligatoires pour avoir une solution optimale.

Notre but dans ce travail est de créer un système qui aide la protection civile à gérer et optimiser le temps des interventions. Et pour cela ce système doit :

1. Prendre toutes les informations indispensables concernant les appelants.
2. Libérer l'agent de CCO pour pouvoir donner les instructions aux appelants.
3. Chercher l'unité la plus proche de la personne qui appelle en prenant en considération le trafic routier.
4. Suivre les interventions en temps réel.
5. Faciliter la recherche de l'information (les interventions, les agents, les engins …).
6. Partager les données entre les unités de la protection civile.

Pour cela, nous avons exploité toutes les technologies avancées pour concevoir et réaliser notre système baptisé : **ProtectLife** qui comporte 5 applications :

1. Application desktop MainUnit_ProtectLife : dédié à l'unité principale de chaque wilaya, et qui consiste à gérer les nouvelles interventions.
2. Application desktop SecondaryUnit_ProtectLife : dédié aux unités secondaires, et qui consiste à recevoir les interventions envoyées par l'application MainUnit_ProtectLife afin de les transmettre aux équipes d'interventions.
3. Application mobile Agent_ProtectLife : dédié aux agents des interventions, et qui consiste à recevoir les interventions envoyées par l'application SecondaryUnit_ProtectLife.
4. Application mobile Mob_ProtectLife : dédié aux citoyens, et qui consiste à envoyer la localisation exacte à API_ProtectLife au cas d'appel au 14.
5. API (Application Programming Interface) API_ProtectLife : un serveur sécurisé pour relier entre les applications et les bases de données

Afin de garantir une meilleure prise en charge la masse des données traitées, nous avons opté pour une base de données NoSQL.

Plan du mémoire

Ce mémoire est organisé comme suit :

Le premier chapitre présente une étude du système existant de la protection civile en décrivant ses composantes, son mécanisme et ses lacunes.

Le deuxième chapitre est consacré pour la conception de notre système à savoir le processus de réalisation et la spécification des besoins ainsi la modélisation utilisant les digrammes du langage de modélisation UML.

Le troisième et dernier chapitre, concerne l'aspect technique de notre contribution, de ce fait, nous avons présenté l'architecture globale du projet et ses différentes applications ainsi que ses différents modules. Par la suite nous présentons les outils utilisés puis nous terminons par une présentation des écrans de notre système.

Enfin, nous clôturons ce mémoire par une conclusion générale.

1. *CONTEXTE DU TRAVAIL*

1.1 Introduction

Nous présenterons dans ce chapitre une étude en profondeur du contexte de notre travail à savoir la protection civile Tlemcen puis le service du centre de coordination opérationnelle. Ensuite nous allons traiter la gestion des interventions. Et nous terminons ce chapitre par l'étude du système existant et ces limites.

1.2 Protection civile

La protection civile, communément appelé « pompiers » décrit le service qui englobe tous les acteurs de la sécurité civile et de la lutte contre l'incendie. La protection civile est un service de secours dont le but est l'assistance et l'aide à la population.

La protection civile a été réalisé dans les années 1964 et dont l'effectif est 70 000 agents.

Elle comporte plusieurs services de secours, dont nous citons : les incendies accident domestique (chute, brûlure, asphyxie, électrocution, noyade) ou physique (hémorragie, arrêt cardiaque, détresse psychologique, malaise) …

De plus chaque wilaya possède sa propre direction ainsi que ses propres moyens, ses spécialités qui change d'une wilaya à une autre et une seule unité principale avec ses propres unités secondaires. [1]

1.3 Organigramme de la protection civile :

L'organigramme représente graphiquement la structure de « La protection civile ».

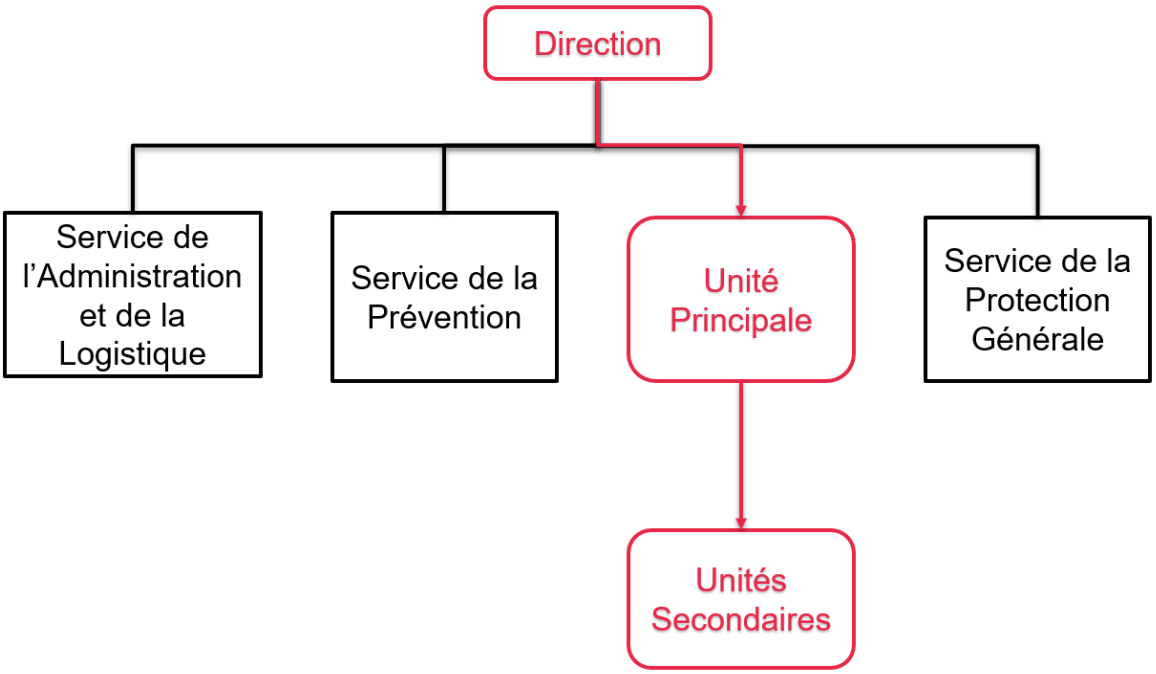


Figure 1 Organigramme de la protection civile

La direction de protection civile est composée de plusieurs services. Nous s'intéressons au service de l'unité principale et au service de l'unité secondaire.

1.4 Centre de coordination opérationnelle (CCO)

Un centre existant dans toutes les unités principales et les unités secondaires dédié à la réception des appels téléphonique24h/24 afin de faire des **interventions** ou donner des conseils et des renseignements.

1.5 Gestion d'intervention :

L'intervention est commencée après avoir reçu un appel téléphonique en demandant de secours. L'intervention se fait en communication entre l'unité principale, l'unité secondaire et l'équipe intervenante.

1.6 Etude de l'existant

L'étude de l'existant est une phase importante pour bien comprendre le système actuel. Notre but est de remédier aux problèmes et limites détectés et apporter des solutions requises ainsi que des suggestions d'améliorations. Nous expliquons dans ce qui suit, le mécanisme d'intervention.

Au début, l'agent de CCO réponds l'appel entré au téléphone en saisissant les informations données sur une feuille de brouillon. Ensuite il coupe l'appel pour appeler le CCO secondaire de l'unité qui est dans le secteur de l'appel pour lui donner les informations et intervenir.

Après, l'agent de CCO secondaire appelle le numéro donné et il prend les informations importantes. Ensuite, il lance une sonnette codée pour indiquer l'équipe qui doit intervenir pour cet appel

En sortant le chef d'agrée de l'équipe intervenue doit prendre les informations de l'agent de CCO de son unité par téléphone ou radio.

Pour chaque situation, le chef d'agrée doit informer l'agent de CCO de son unité qui à son tour doit informer l'agent de CCO de l'unité principale.

Au final, le rapport doit être rempli par l'agent de CCO principale suite aux informations données

1.7 Critique de l'existant

Le temps est une notion importante pour la vie humaine. Une minute gagnée peut sauver plusieurs vies mais la méthode de travail actuel possède un nombre important de problèmes qui engendre une perte de temps importante tels que :

1. Surcharge des taches pour l'agent de CCO principale (il répond au téléphone, il communique avec les équipes intervenantes et il écrit les rapports …) ce qui provoque un chevauchement des appels et la perte des informations.
2. La recherche difficile de l'adresse exacte.
3. Mauvaise gestion de la localisation géographique de l'unité la plus proche.
4. Manque du suivi sur le trafic routier.
5. Mauvaise gestion pour suivre les interventions.

1.8 Conclusion

Dans ce chapitre, nous avons pu donner une présentation claire du contexte de travail. En outre, nous avons projeté les limites du système actuel. Pour cela, nous avons commencé par la présentation de la gestion des interventions qui se fait au niveau du centre de coordination opérationnelle de la protection civile. Par ailleurs, nous nous intéressons, dans le chapitre suivant aux bases de données NoSQL.

2 *Base de données NoSQL*

2.1 Introduction

Chaque jour, des Chaque jour, des trillions d'octets de données sont générées dont les sources varient entre messages sur les réseaux sociaux, Images et vidéos publiées en ligne, Signaux GPS de téléphones mobiles …. C'est ce qu'on appelle Big Data ou données massives

2.2 Base de données NoSQL

NoSQL appelé aussi Not Only SQL. Il est inventé en 2009 lors d'un évènement sur les bases de données distribuées. Ce terme est utilisé pour tout type de SGBD non relationnel. [2]C'est une nouvelle approche de stockage et de gestion de données. Cette dernière est utilisée lors :

1. Un passage à l'échelle via un contexte hautement distribué
2. Gestion de données complexes et hétérogènes
3. Pas de schéma pour les objets
4. Quantité de données énorme (Pétabytes)
5. Besoin de temps de réponse
6. Cohérence de données faible
7. Les principaux axes sont la haute disponibilité et le partitionnement des données
8. Permet de gérer de très grosses volumétries de données

2.3 Limites des bases de données relationnelles

1. Incapable de gérer de très grands volumes de données (de l'ordre du pétaoctet)
2. Impossible de gérer des débits extrêmes (plus que quelques milliers de requêtes par seconde)
3. Le modèle relationnel est parfois peu adapté au stockage et à l'interrogation de certains types de données (données hiérarchiques, faiblement structurées, semi-structurées)

4. Les propriétés ACID entraînent de sérieux surcoûts en latence, accès disques, temps CPU (verrous, journalisation, etc.)
5. Performances limitées par les accès disque [3]

2.4 Classification des bases de données NoSQL

Clé- valeur, document, colonnes et graphes sont les 4 types de bases de données NoSQL. Etudions d'un peu plus près chacun de ces types

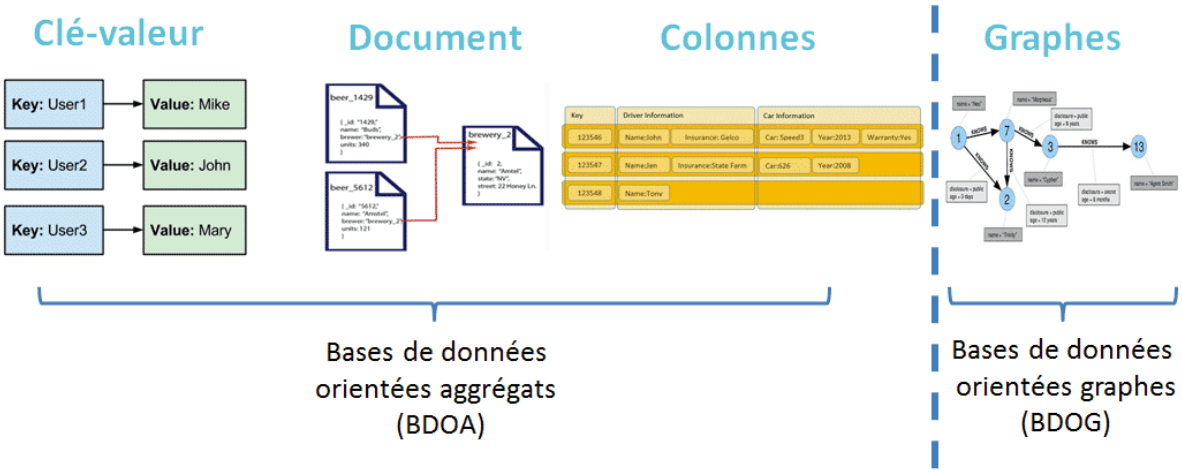


Figure 2 types des bases de données NoSQL [4]

2.4.1 Orienté clé – valeur

Les données sont stockées en clé-valeur : une clé plus un BLOB (dans lequel on peut mettre : nombre, date, texte, XML, photo, vidéo, structure objet).

2.4.2 Orienté colonnes

Ces bases de données se rapprochent des bases de données relationnelles, à ceci près qu'elles permettent de remplir un nombre de colonnes variable.

2.4.3 Orientée graphes

Ces bases de données, basées sur la théorie des graphes, sont gérées par nœuds, relations et propriétés. Elles gèrent des données spatiales, sociales ou financières (dépôts/retraits).

2.4.4 Orientée documents

Ces bases de données stockent des données semi-structurées : le contenu est formaté JSON ou XML, mais la structure n'est pas contrainte.

2.5 Comparaison entre les différents base NoSQL

Selon le site officiel, le taux d'utilisation des bases de données NOSQL en 2019 est représenté par l'histogramme suivant :

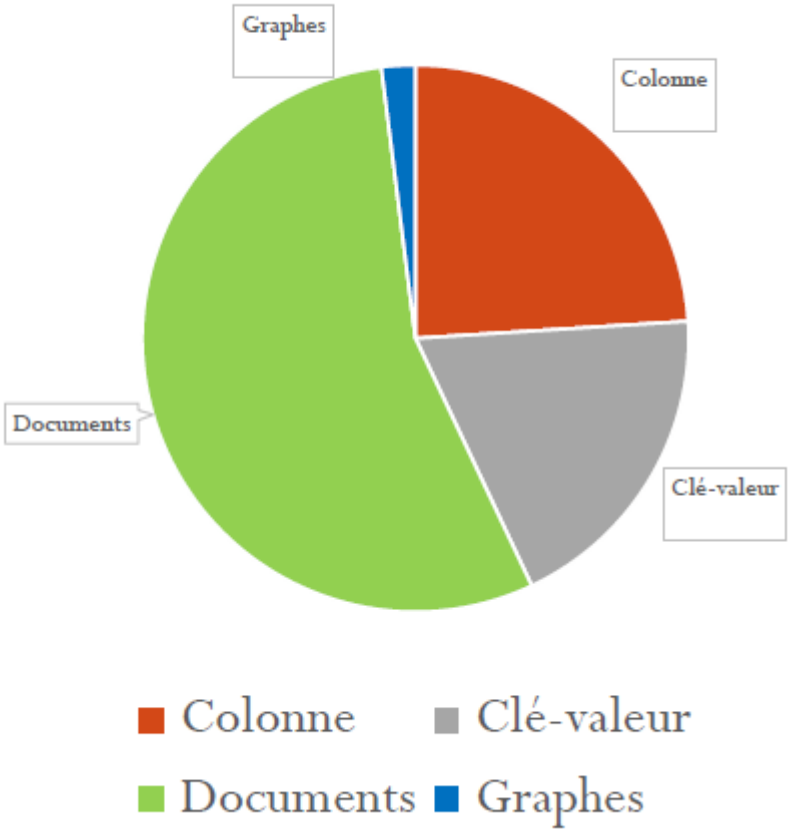


Figure 3 L'utilisation des catégories de NoSQL

Et dans notre travail, on s'intéresse aux bases de données orienté document, et plus particulièrement le SGBD MongoDB est la plus connue des bases de données NoSQL. Et parmi ses avantages :

1. Facile à installer MongoDB
2. Base de données haute vitesse
3. Base de données sans schéma
4. Base de données évolutive horizontalement
5. La performance est **très élevée**

2.6 Base de données orienté document -MongoDB-

MongoDB est la plus connue des bases de données NoSQL. Il s'agit d'une base de données Open-Source orientée document. MongoDB est une base de données évolutive et accessible. Il est en **C++**. Dans MongoDB, **JavaScript** peut être utilisé comme langage de requête. Il est très utile dans les cadres JavaScript populaires. Des performances étonnantes et de nouvelles fonctionnalités ont propulsé cette base de données NoSQL à la première place de notre liste. [5]La requête de MongoDB se fait comme suite :

```
db.Unites.find({}, { nom: 1, _id: 0 });
```

Afin de de connecter notre serveur API, nous avons utilisé le module **Mongoose.**

2.7 Mongoose

Mongoose est un module Node.js qui s'installe avec **NPM** (Node Package Manager). Il sert de passerelle entre notre serveur NodeJs et notre serveur MongoDB. Et la connexion entre l'API et la base de données se fait comme suite :

```
mongoose
  .connect(le_lien_de_notre_base_de_données, {
    useNewUrlParser: true,
    useCreateIndex: true,
    useFindAndModify: false,
    useUnifiedTopology: true,
  })
  .then(() => {
    console.log("db connecction successful !");
  });
```

2.8 Conclusion

MongoDB est le facteur le plus important pour la réalisation de notre a cause de la masse de données qu'on va traiter. Dans le chapitre suivant, nous nous intéressons aux besoins de l'application pour répondre aux attentes du système.

3. CONCEPTION DU SYSTEME « *ProtectLife* »

3.1 Introduction

Pour mener à bien le projet, nous devons tout naturellement avoir recours à un formalisme de conception dont nous choisissons le langage de modélisation UML « Langage de modélisation unifié » qui est le langage de modélisation graphique qui va nous permettre de décrire les besoins, de spécifier et modéliser notre cas d'application ainsi que représenter le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par le logiciel.

Dans ce chapitre, nous commençons par tirer les nouveaux besoins de notre système puis nous passons à la conception d'où nous allons faire appel à la boite à outils UML à savoir : le diagramme des cas d'utilisation, le diagramme de séquences et le diagramme de classes.

3.2 Analyse des besoins

Dans un souci de concevoir une application avec plus de fonctionnalités possibles et dans le but d'avoir une interface plus conviviale et plus facile à utiliser tout en étant plus efficace, nous allons concevoir un système de gestion des interventions qui vise à pallier les limites de l'existant. [6]

Dans cette partie nous allons identifier les besoins de notre application. Nous citons ci-dessous quelques-uns :

- L'application doit **minimiser** le temps de réponse de l'intervention.
- L'application doit **gérer** les interventions.
- L'application doit **envoyer** l'intervention à l'unité la **plus proche**.
- L'application doit **localiser** les appels entrés.
- L'application doit permettre l'agent de CCO d'être **en contact** avec la personne qui appelle.
- L'application doit **gérer** les utilisateurs des applications et leurs droits d'accès.
- L'application doit **suivre** les équipes intervenantes en temps réel.
- L'application doit **remplacer** les papiers.
- L'application doit **gérer** la communication entre les différentes applications

3.3 Présentation de l'UML

Nous allons utiliser pour la modélisation de notre système le langage de modélisation UML. Ce dernier est très riche pour présenter les objectifs de notre système. Un ensemble de diagrammes de la boite à outils d'UML vont être utilisée à savoir : diagramme de cas d'utilisation, diagramme de séquence, diagramme de classe.

3.3.1 Diagramme de cas d'utilisation

Le but de ce diagramme et d'avoir une vision globale sur les interfaces du futur système. [7] Les diagrammes suivants montrent les cas d'utilisation des différents acteurs du nouveau système de la protection civile :

- Administrateur de l'unité principale : Utilise l'application desktop pour consulter les interventions, les agents, les engins et les statistiques.

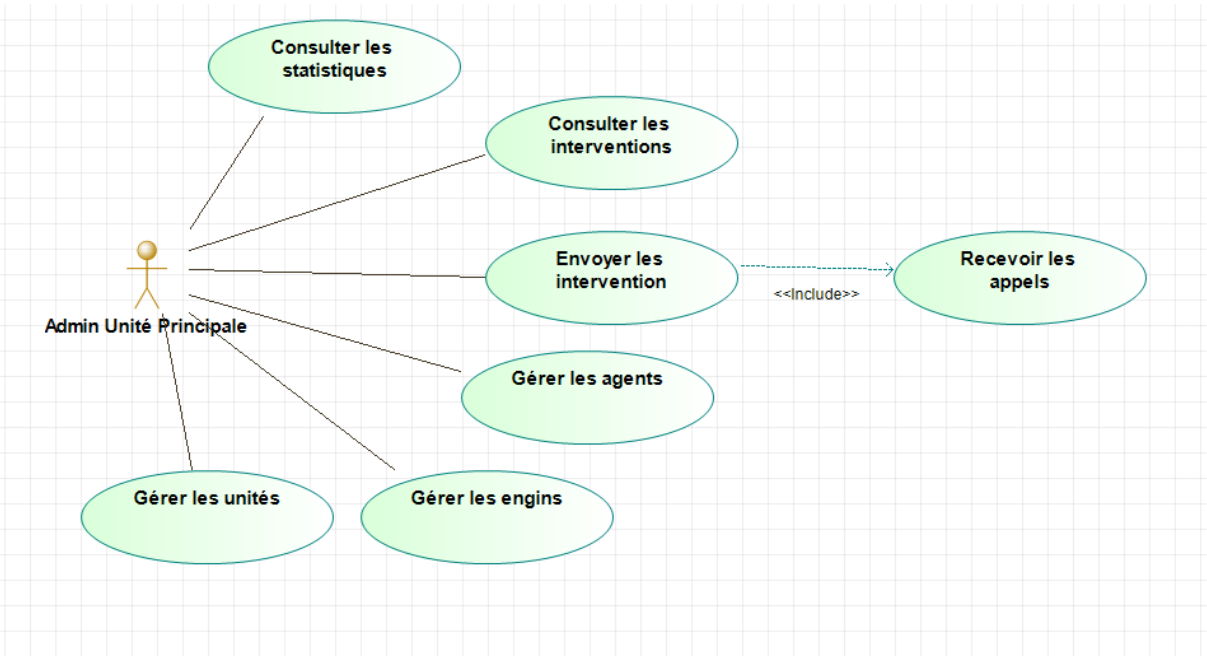


Figure 4 Diagramme de cas d'utilisation de l'Administrateur de l'unité principale

- Administrateur de l'unité secondaire : Utilise l'application desktop pour consulter les interventions, et gérer la liste des agents, des engins et des statistiques.

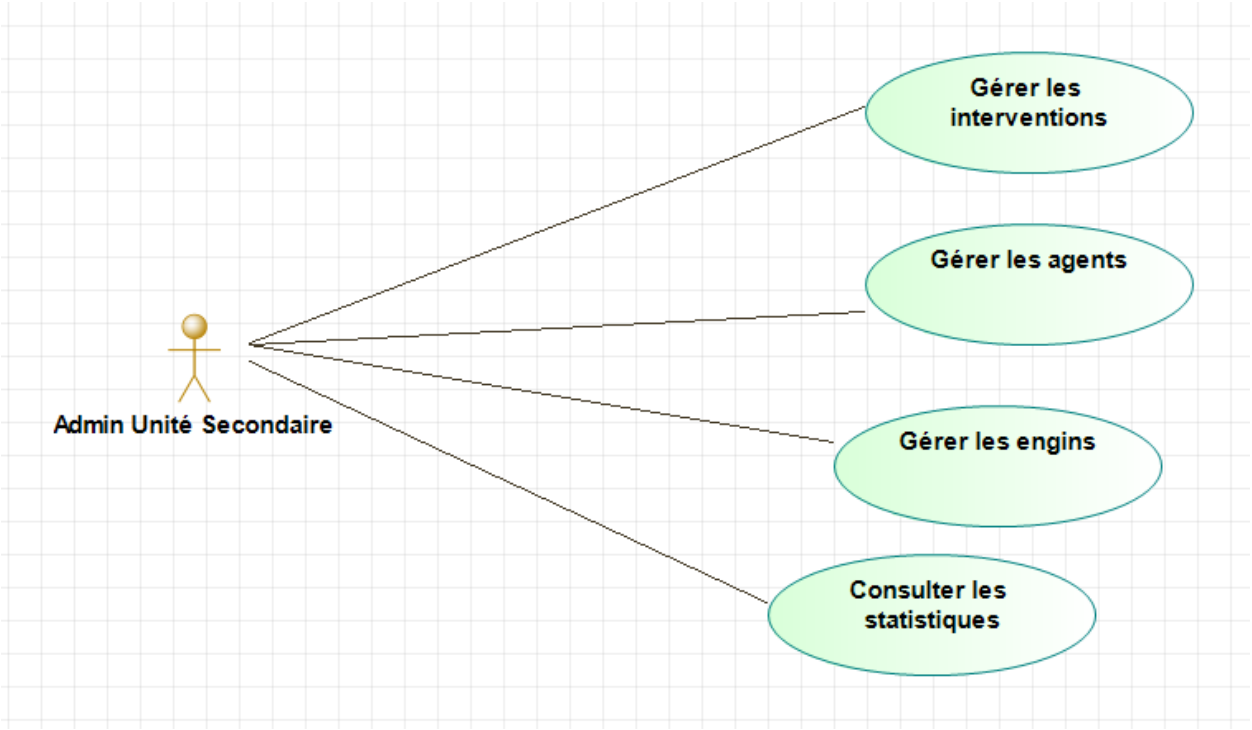


Figure 5 Diagramme de cas d'utilisation de l'Administrateur de l'unité secondaire

- Agent de CCO de l'unité principale : Utilise l'application desktop pour consulter les interventions de toutes les unités secondaires, et envoyer l'intervention après avoir reçu un appel de secours

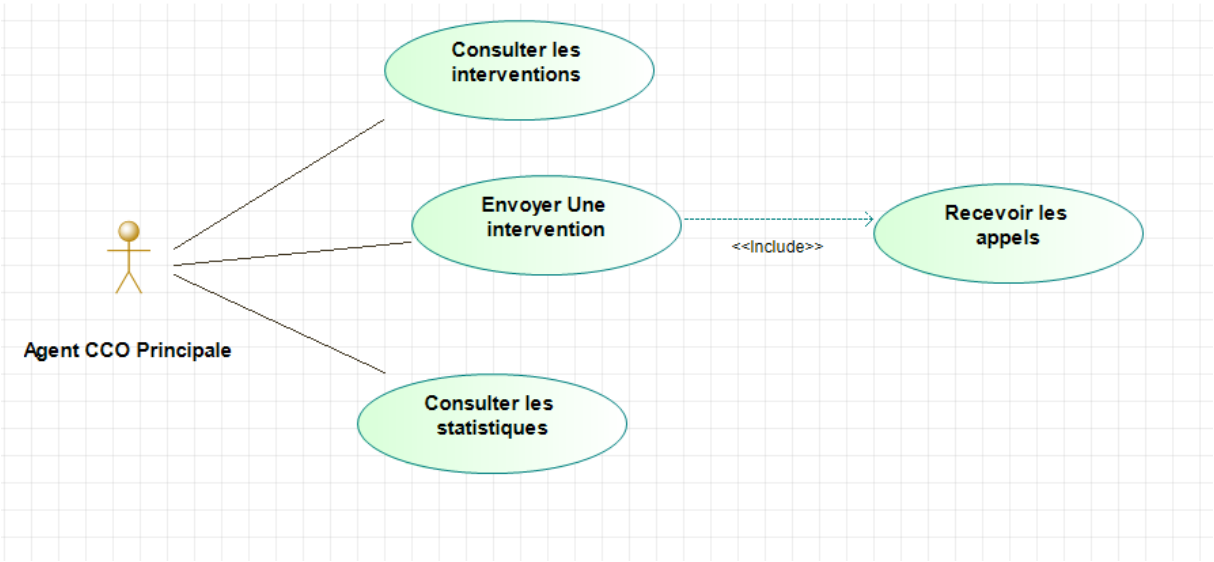


Figure 6 Diagramme de cas d'utilisation de l'agent de CCO de l'unité principale

- Agent de CCO de l'unité secondaire : Utilise l'application desktop pour gérer les interventions reçues en les envoyant aux équipes de son unité, consulter le planning et les statistiques.

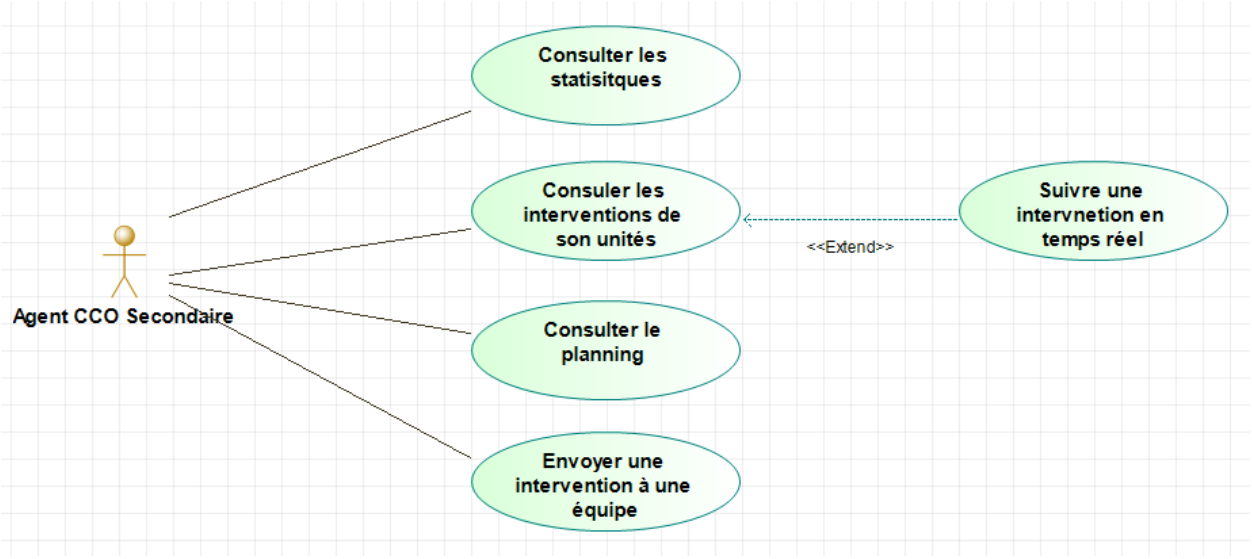


Figure 7 Diagramme de cas d'utilisation de l'agent de CCO de l'unité secondaire

Remarques :

- Les acteurs du projet doivent être authentifiés pour pouvoir utiliser les applications et chaque acteur selon ses droits d'accès.

- Un admin de l'unité principale a le droit de gérer son unité et ses unités secondaires.

3.3.2 Diagramme de séquence

Les cas d'utilisations avec ses multiples relations (extend, include, etc.), nous donne une vue presque réelle de l'application. Cette richesse nous montre une vue globale de l'application mais pour voir réellement la succession des actions des acteurs il nous faut un autre modèle qui nous détaille le séquencement des opérations ce diagramme s'agit du diagramme des séquences. Ce dernier comme son nom l'indique il développe un cas d'utilisation en montrant les différentes opérations permettant de réaliser l'action du cas en question. [8]

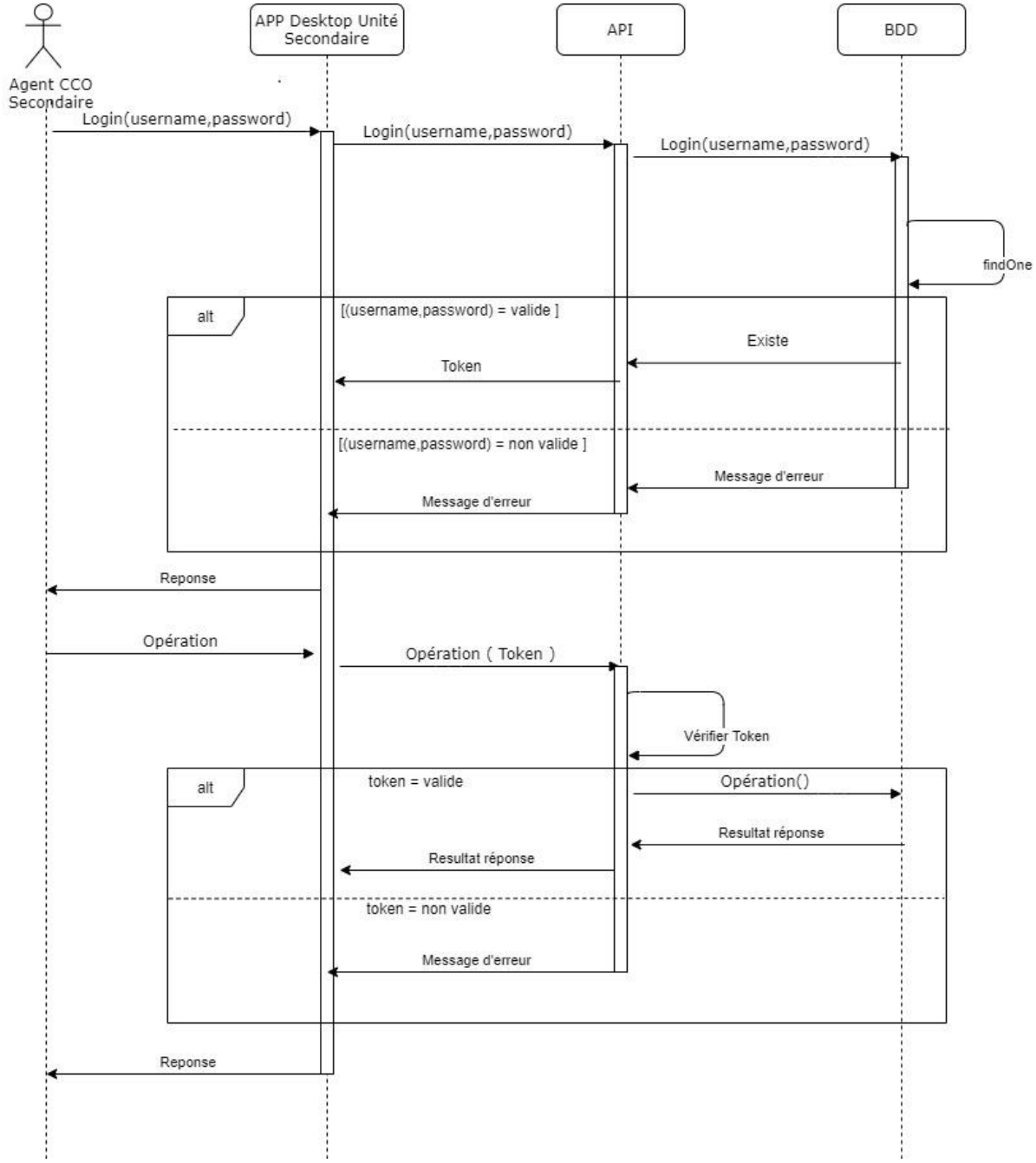


Figure 8 Diagramme de séquence de l'opération de l'agent de CCO

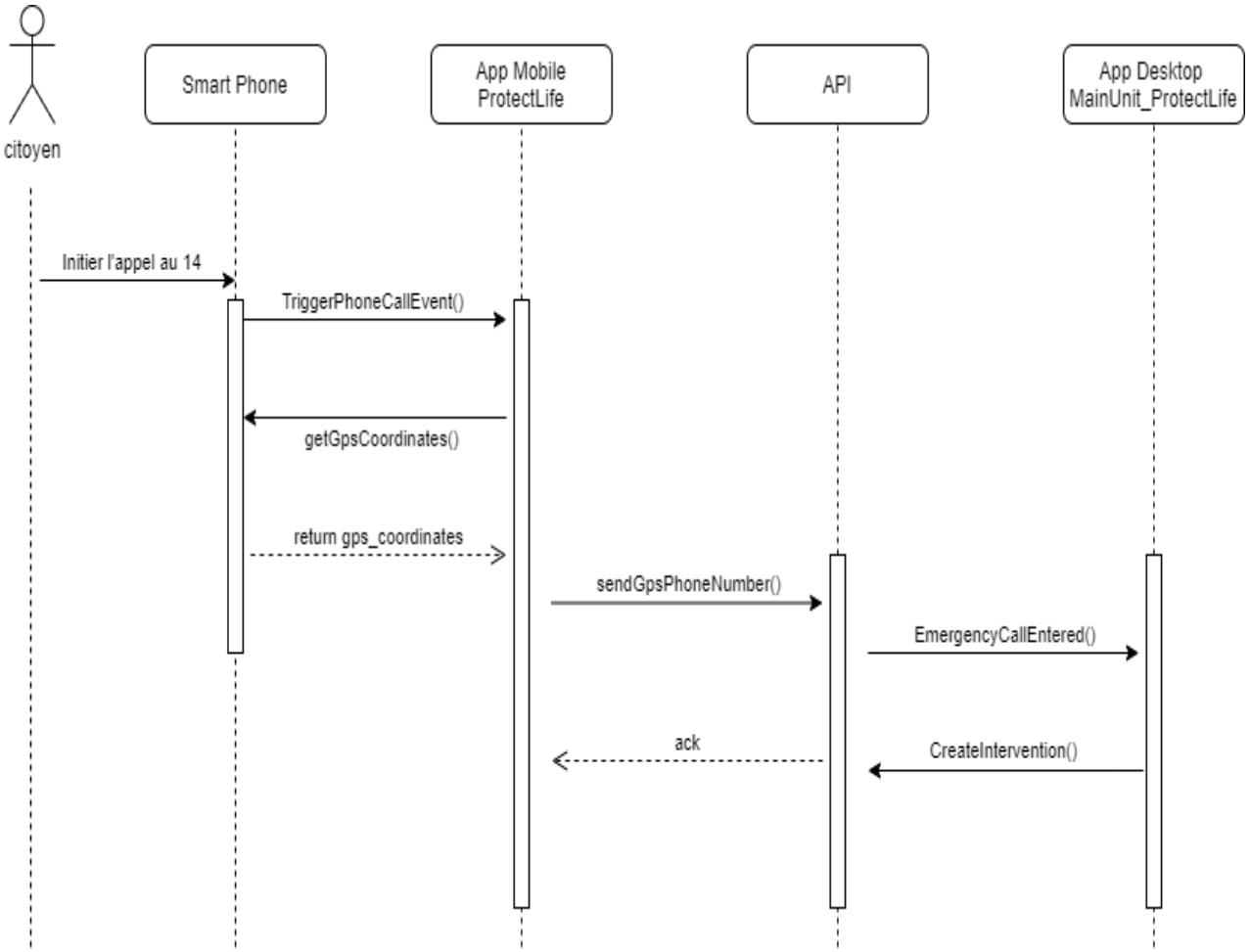
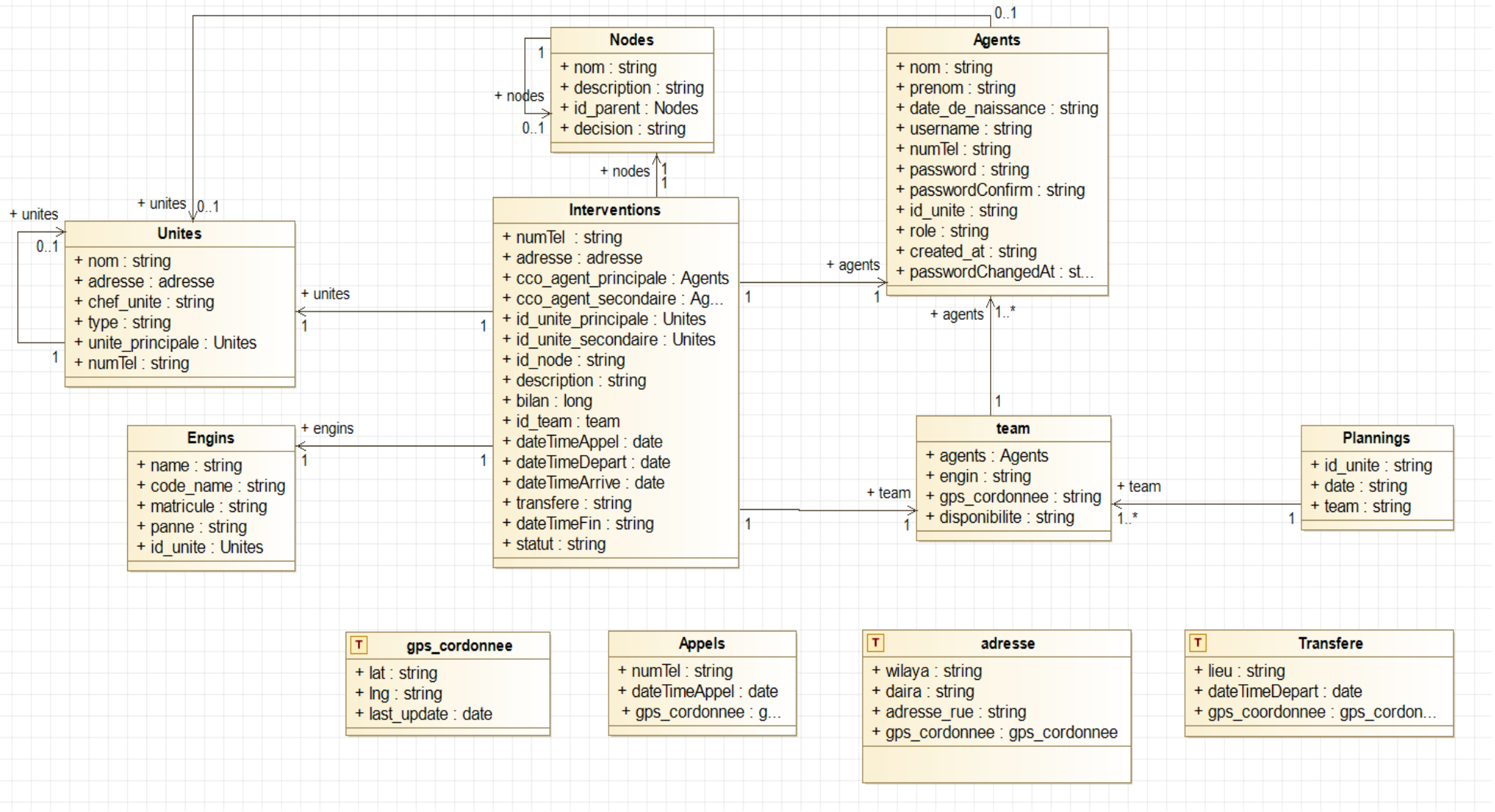


Figure 9 Diagramme de séquence d'une opération d'appel d'urgence

3.3.3 Diagramme de classe

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet parce qu'il modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application. [9]


Nodes
+ nom : string
+ description : string
+ id_parent : Nodes
+ decision : string
+ nodes
1
0..1
Agents
+ nom : string
+ prenom : string
+ date_de_naissance : string
+ username : string
+ numTel : string
+ password : string
+ passwordConfirm : string
+ id_unite : string
+ role : string
+ created_at : string
+ passwordChangedAt : st...
0..1
Unites
+ nom : string
+ adresse : adresse
+ chef_unite : string
+ type : string
+ unite_principale : Unites
+ numTel : string
+ unites
0..1
Interventions
+ numTel : string
+ adresse : adresse
+ cco_agent_principale : Agents
+ cco_agent_secondaire : Ag...
+ id_unite_principale : Unites
+ id_unite_secondaire : Unites
+ id_node : string
+ description : string
+ bilan : long
+ id_team : team
+ dateTimeAppel : date
+ dateTimeDepart : date
+ dateTimeArrive : date
+ transfere : string
+ dateTimeFin : string
+ statut : string
+ agents
1..*
Engins
+ name : string
+ code_name : string
+ matricule : string
+ panne : string
+ id_unite : Unites
+ engins
team
+ agents : Agents
+ engin : string
+ gps_cordonnee : string
+ disponibilite : string
+ team
Plannings
+ id_unite : string
+ date : string
+ team : string
gps_cordonnee
+ lat : string
+ lng : string
+ last_update : date
Appels
+ numTel : string
+ dateTimeAppel : date
+ gps_cordonnee : g...
adresse
+ wilaya : string
+ daira : string
+ adresse_rue : string
+ gps_cordonnee : gps_cordonnee
Transfere
+ lieu : string
+ dateTimeDepart : date
+ gps_coordonnee : gps_cordon...

3.3.4 Modèle logique

A partir du diagramme de classe que nous avons effectué, on peut réaliser le modèle logique, vue que notre base donnée est NoSql et en raison d'optimisation :

1. Nous avons créé les collections des classes Unités, Engins, Interventions, Appels, Agents et Plannings.
2. Nous avons mis les associations « un à plusieurs » et « plusieurs à plusieurs » engendre la migration de l'ID entre les collections.
3. Nous avons mis le schéma validateur pour vérifier les données entrées à notre base de données

Au final, nous avons abouti ce schéma de la classe hôpital comme suivant :

```
{
    nom: {
        type: String,
        unique:true
    },
    gps_coordonnee: {
        lat: {
            type: Number,
            required: [true, "Vous devez saisir latitude"],
        },
        lng: {
            type: Number,
            required: [true, "Vous devez saisir longitude"],
        },
    },
    numTel: {
        type: String,
        required: [true, "Vous devez saisir un numéro telephone"],
        unique:true
    },
    created_at: {
        type: Date,
        default: dateTime
    },
}
```

3.4 Conclusion

L'étape de l'analyse et de conception est l'étape la plus importante dans l'étape développement, car le résultat nous guide dans l'étape d'implémentation. Dans le chapitre suivant nous allons montrer comment nous avons implémenté le résultat de ce chapitre pour réaliser notre application.

4. *IMPLEMENTATION DU SYSTEME*

4.1 Introduction

Après avoir analysé les besoins et défini la méthodologie de conception, nous allons implémenter notre système via un environnement adéquat.

Ce chapitre aborde l'implémentation de notre système, nous allons présenter l'architecture de notre système, notre choix de technologie d'implémentation, les outils, les techniques et les langages de programmation que nous allons utiliser pour chaque application. Nous terminons avec quelques interfaces utilisateurs.

4.2 Architecture du système ProtectLife

Afin de mener à la mise en œuvre de notre système, nous nous sommes basés sur l'optimisation de notre système afin d'avoir le meilleur temps de réponse. Pour arriver à notre but nous avons intégré les technologies les plus modernes en ajoutant nos propres fonctions afin d'optimiser le temps de réponse. Parmi ces technologies : NodeJs, Mongoose, ElectronJs, VueJs, Android, NoSQL ….

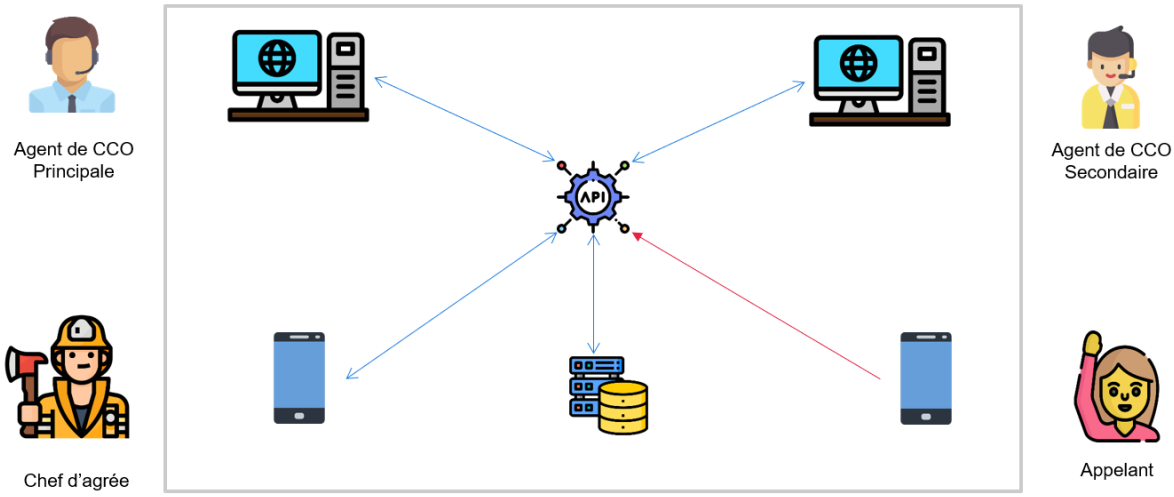


Figure 10 Architecture de l'application

Pour assurer la rapidité et la possibilité de déboguer les erreurs pendant l'implémentation, nous avons utilisé les plateformes suivantes :

Visual Studio Code : est un éditeur de code open-source, gratuit et multiplateforme (Windows, Mac et Linux), développé par Microsoft, L'éditeur supporte plusieurs dizaines de langages de programmation d'ont les langages de programmation orienté web (HTML, CSS, JavaScript, VueJs …). [10]

Postman : est la chaîne d'outils complète pour les développeurs d'API, utilisée par 6 millions de développeurs et plus de 200 000 entreprises dans le monde pour accéder à 130 millions d'API par mois. Postman simplifie et accélère l'utilisation des API en aidant les développeurs à chaque étape de leur flux de travail. Il est disponible pour les utilisateurs de MacOs, Windows, Linux et Chrome. [11]

GitHub *:* est un service d'hébergement Web permettant de contrôler les versions à l'aide de Git. Il est principalement utilisé pour le code informatique. Il offre toutes les fonctionnalités de contrôle de version distribuées et de gestion de code source de Git. [12]

Modelio : est un outil de modélisation UML disponible sur les plates-formes Windows, Linux et Mac. Il intègre également le support de la modélisation des exigences, du dictionnaire, des règles métier et des objectifs. [13]

Android Studio : est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntelliJ IDEA et utilise le moteur de production Gradle. Il peut être téléchargé sous les systèmes d'exploitation Windows, macOS et Linux.[14]

Trello : est un outil de gestion de projet en ligne, lancé en septembre 2011. Il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement. [15]

Après avoir préparé l'environnement de l'implémentations de notre système, nous avons commencé à implémenter nos différentes applications.

Dans cette partie nous allons présenter les 3 types d'applications en justifiant notre choix.

4.2.1 SERVEUR REST API

Afin de pouvoir échanger et traiter les données entres les autres applications et notre base de données, nous avons mis le serveur API_ProtectLife. Pour sa création nous avons utilisé la plateforme NodeJs pour pouvoir communiquer avec le serveur MongoDB à l'aide du module Mongoose.

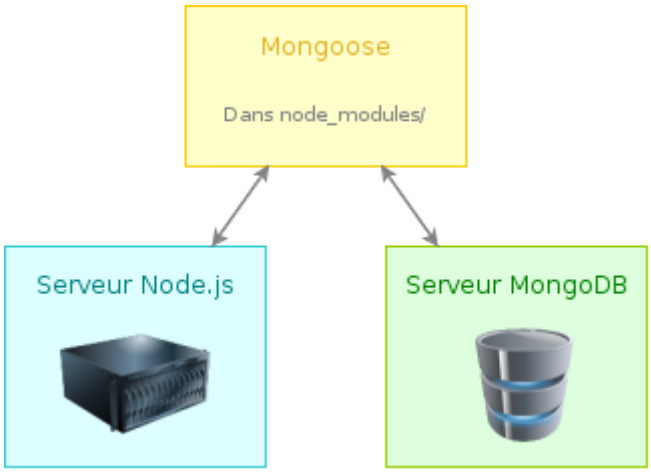


Figure 11 Communication entre NodeJs et MongoDB

Pour la réalisation du serveur, nous avons utilisé l'architecture MVC (modèle, vue et contrôleur) pour donner une conception claire et efficace à l'implémentation et la maintenance. Cette architecture est donc décomposée en trois étapes :

1. Le modèle (Model) : correspond aux schémas des classes de notre base donnée avec les méthodes de vérification et de lectures comme le model suivant de la classe hôpital :

```
const hospitalSchema = new mongoose.Schema({
    nom: {
        type: String,
        unique:true
    },
    gps_coordonnee: {
        lat: {
            type: Number,
            required: [true, "Vous devez saisir la latitude"],
        },
        lng: {
            type: Number,
            required: [true, "Vous devez saisir la longitude"],
        },
    },
    numTel: {
```

```
        type: String,
        required: [true, "Vous devez saisir un numéro telephone"],
        unique:true
    },
});
```

2. Le contrôleur (Controller) : contient les actions effectuées par les utilisateurs des applications ProtectLife l'exemple suivant :

```
exports.getAllHospitals = catchAsync(async (req, res, next) => {
    const hospitals = await Hopital.find();
    if (!hospitals) {
        return next(new AppError("la liste des hopitaux est vide.", 404));
    }
    res.status(200).json(hospitals)
});
```

3. Une vue (View) contient la présentation de l'interface graphique qui représentes les applications mobiles et les applications desktops où les données sont reçues des contrôleurs.

Après avoir terminé l'architecture de l'application, nous avons ajouté des fonctions, qui s'appellent les middlewares qui ont l'accès aux données entrées, pour garantir la performance et la fiabilité. Notre serveur utilise les middlewares suivants :

1. Middleware niveau application : les middlewares de ce niveau sont souvent utilisés pour vérifier les données entrées pour garantir la sécurité. Le middleware suivant est utilisé pour protéger le serveur API_ProtectLife contre les attaques par déni de service distribuées (DDoS).

```
app.use("/API", rateLimit({
  // pour maximiser 100 requêtes / heure pour le meme @IP
  max: 100,
  windowMs: 60 * 60 * 1000,
  message: " Trop de requêtes de cette adresse IP, veuillez réessayer dans une heure! "
}));
```

Nous avons aussi la protection contres :

- Attaques XSS
- Attaque HTTP Paramètre Pollution
- NoSQL injection
- Sécurité http headers

2. Middleware niveau routeur : Les middlewares niveau routeur fonctionnent de la même manière que le middleware niveau application, à l'exception ils lient chaque requête demandée vers le contrôleur compatible. Ces middlewares nous garantirent l'utilisation de la format CRUD (Create, Read, Update, Delete) pour la meilleure gestion des routes comme l'exemple suivant :

```
router
    .get('/hospital', authController.protect, hospitalController.getAllHospitals)
    .get('/hospital/:id', authController.protect, hospitalController.getHospital)
    .post('/hospital/', authController.protect, authController.restricTo('admin'),
                        hospitalController.createHospital)
    .patch('/hospital/:id', authController.protect, authController.restricTo('admin'),
                        hospitalController.modifyHospital)
    .delete('/hospital/:id', authController.protect, authController.restricTo('admin'),
                        hospitalController.deleteHospital)
```

Voyons les verbes http les plus utilisés :

GET : c'est le verbe HTTP qui permet d'accéder à une ressource, sans aucune modification.

POST : c'est le verbe qui permet de créer un nouvel objet dans la base donnée.

PATCH : permets de modifier / mettre à jour une ressource.

PUT : permets de modifier / remplacer une ressource.

DELETE : permets de supprimer une ressource.

3. Middleware de traitement d'erreurs : Les middlewares niveau d'erreur nous permettent de gérer l'affichage de certaines erreurs qui peuvent être utiles pour les hackers. Dans l'exemple suivant nous allons montrer un middleware simplifié des middlewares de gestion d'erreur.

```
 module.exports = (err, req, res, next) => {
  err.statusCode = err.statusCode || 500;
  err.status = err.status || "error";
// partie développement
  if (process.env.NODE_ENV == "developement") {
    // pour afficher l'erreur en entier au développeur
    sendErrorDev(err, res);
  }
// partie production
else if (process.env.NODE_ENV == "production") {
    if (err.code === 11000)
sendErrorProd(new AppErr(`Duplicate field value: Please use another value!`, 401), res);
    else if (err.code === "JsonWebTokenError" || err.code === "TokenExiredError")
sendErrorProd(new AppErr("Invalid token.Please log in again", 400), res);
    else sendErrorProd(new AppErr("Something went worg!", 500), res);
  }
};
```

La figure suivante montre le cycle d'une requête entrée à l'application API_ProtectLife

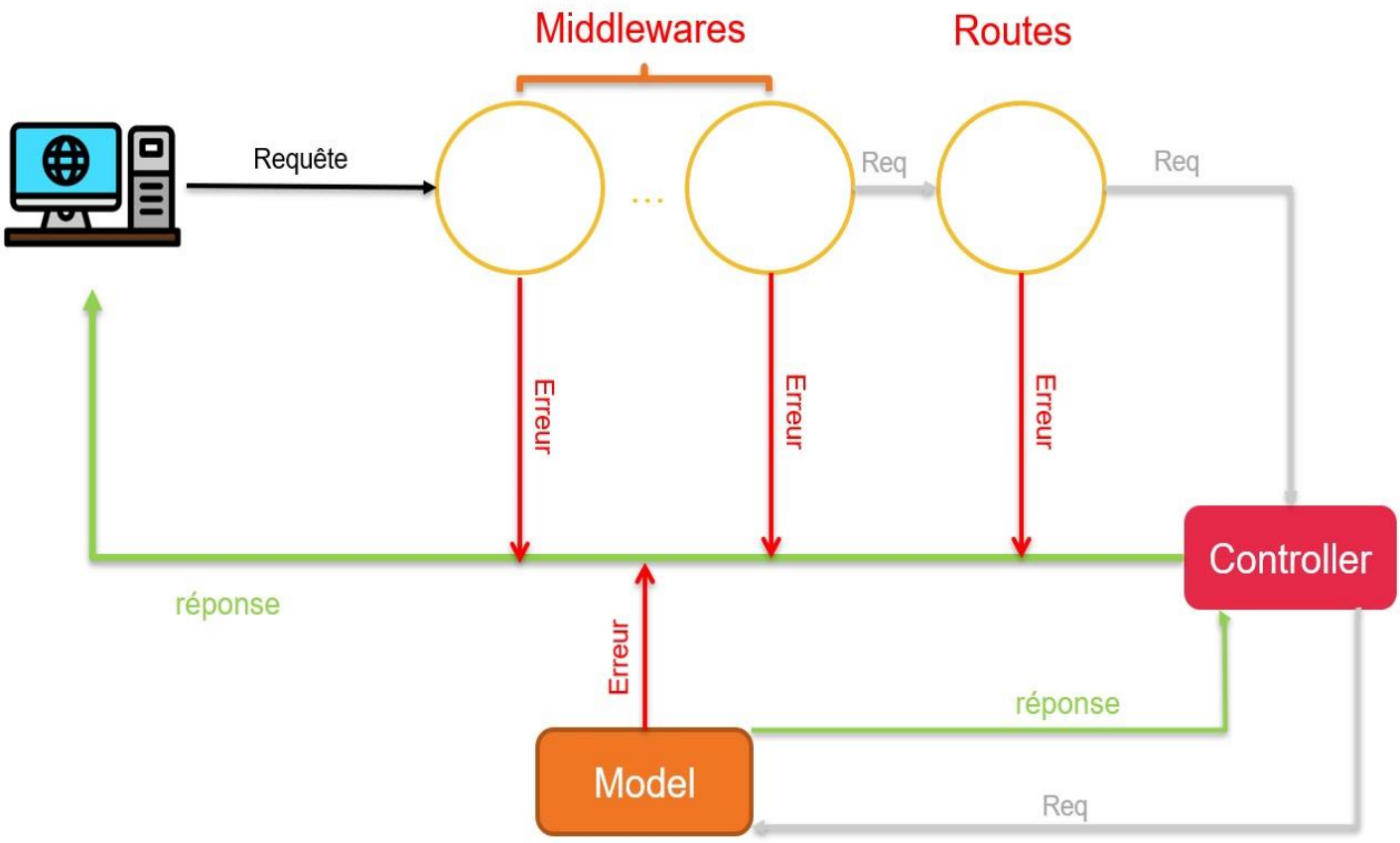
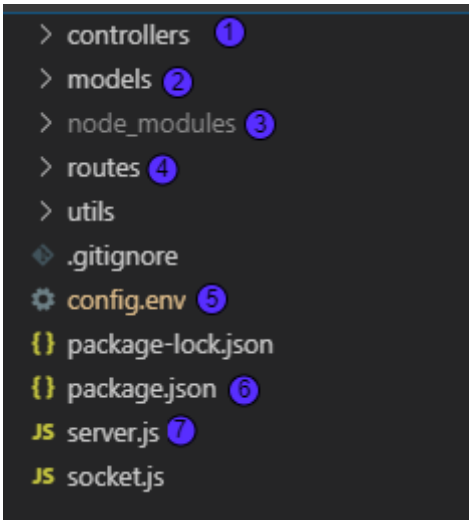


Figure 12 Cycle d'une requête

Et nous obtenons une structure des fichiers suivante :



1. Le dossier des contrôleurs contient tous les scripts des contrôleurs.
2. Le dossier Modèles contient les fichiers de script de chaque modèle
3. Ce dossier contient tous les packages installés par Npm et répertoriés dans (6)
4. Ce dossier contient les fichiers de script des routes pour chaque contrôleur.
5. Config.env ce fichier contient toutes les variables d'environnement telles que dans notre cas le nom de domaine et le port dans lequel notre serveur va initier la connexion et les informations d'identification de la base de données
6. Comme mentionné dans (3) ce fichier contient tous les paquets nécessaires pour que notre serveur fonctionne correctement.
7. Server.js est le fichier de départ de notre serveur, ce sera le premier fichier de script à exécuter et il lancera l'environnement de notre serveur tel que le serveur http et la connexion avec la base de données.

Nous avons utilisé NodeJs pour communiquer avec les différentes applications de notre système ProtectLife. NodeJs a les avantages suivants :

1. Node.js utilise un modèle d'E/S non bloquant basé sur les événements et la gestion asynchrone, ce qui le rend léger et efficace.

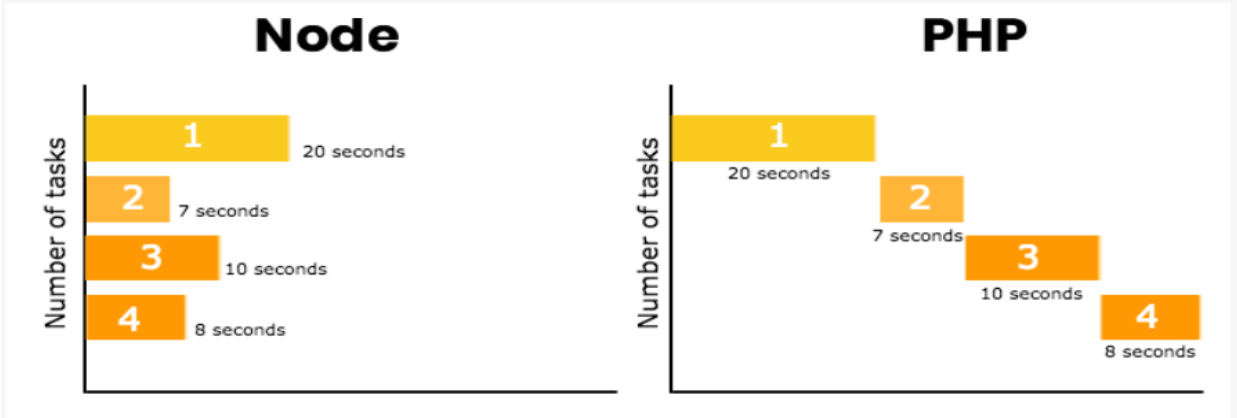


Figure 13 E/S NodeJs vs PHP [16]

2. Facile d'acquisition et d'apprentissage (JavaScript)
3. Analyse et exécute du code JavaScript très rapidement (Google Chrome V8)

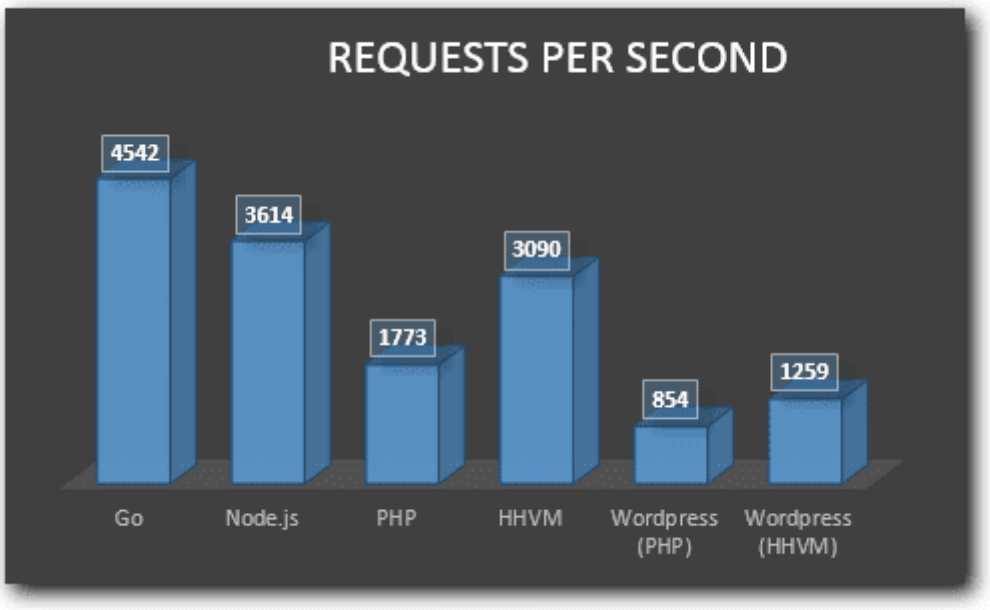


Figure 14 NodeJs requêtes par seconds[17]

4. Portabilité du code JavaScript aussi bien côté serveur que client.
5. Multiplateformes (mobile, desktop, tv)

4.2.2 Applications web du ProtectLife

Pour la meilleure utilisation du système ProtectLife dans les unités nous avons pensé à utiliser une application web ou l'utilisateur n'a qu'à télécharger les données JSON de l'API_ProtectLife et gagner du temps en évitant de télécharger les balises html des sites web. Pour cela nous avons utilisé **ElectronJs** qui est un framework pour créer des applications multi-plateformes de bureau avec des technologies web grâce à **Chromium** et **NodeJs**. Ensuite, Nous avons créé les deux applications suivantes :

4.2.2.1 MainUnit_ProtectLife

Cette application est dédiée à chaque unité principale. Cette application est utilisée pour pouvoir effectuer les opérations comme la gestion des agents, des hôpitaux, des engins, des unités secondaires et la plus importante la gestion des interventions.

Cette application est utilisée par deux types d'utilisateur :

1. Administrateur : Un administrateur de l'unité principale peut gérer toutes les données **de son unité et ses unités secondaires**. Par exemple il peut ajouter, modifier ou supprimer des agents de l'unité souhaitant comme le montre les figures suivantes capturées de notre application MainUnit_ProtectLife :

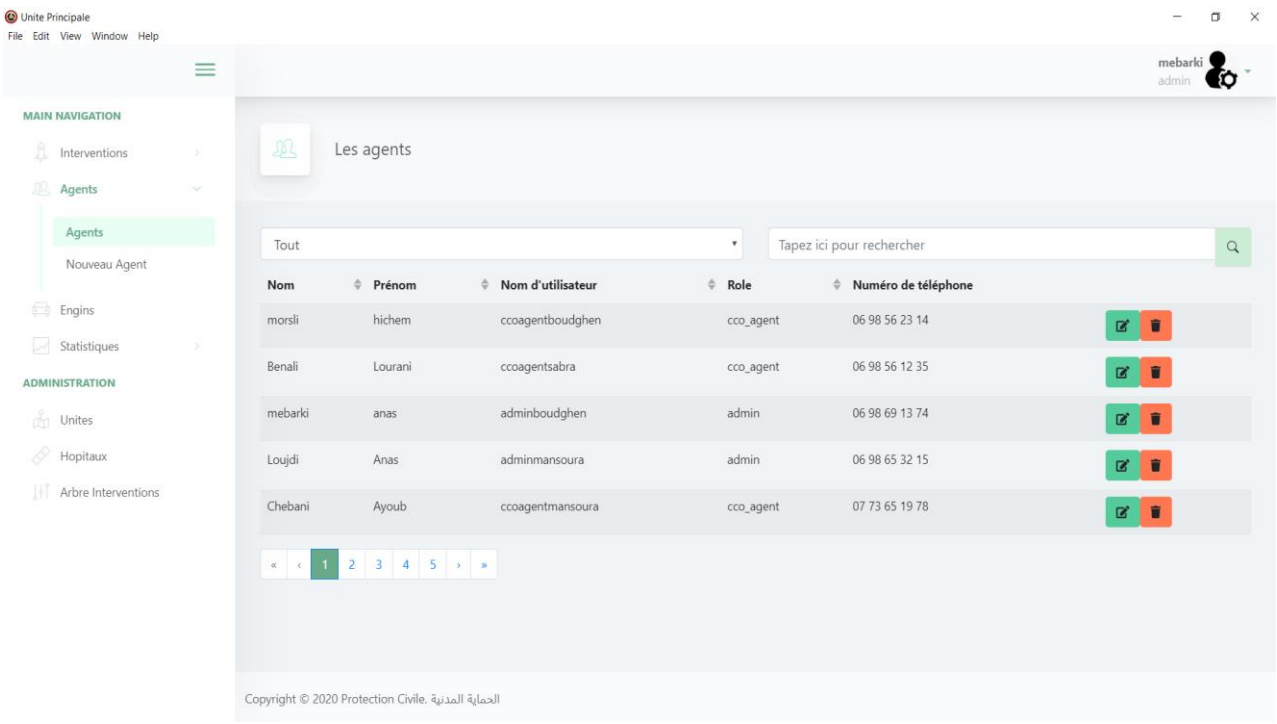


Figure 15 Unité principale - les agents

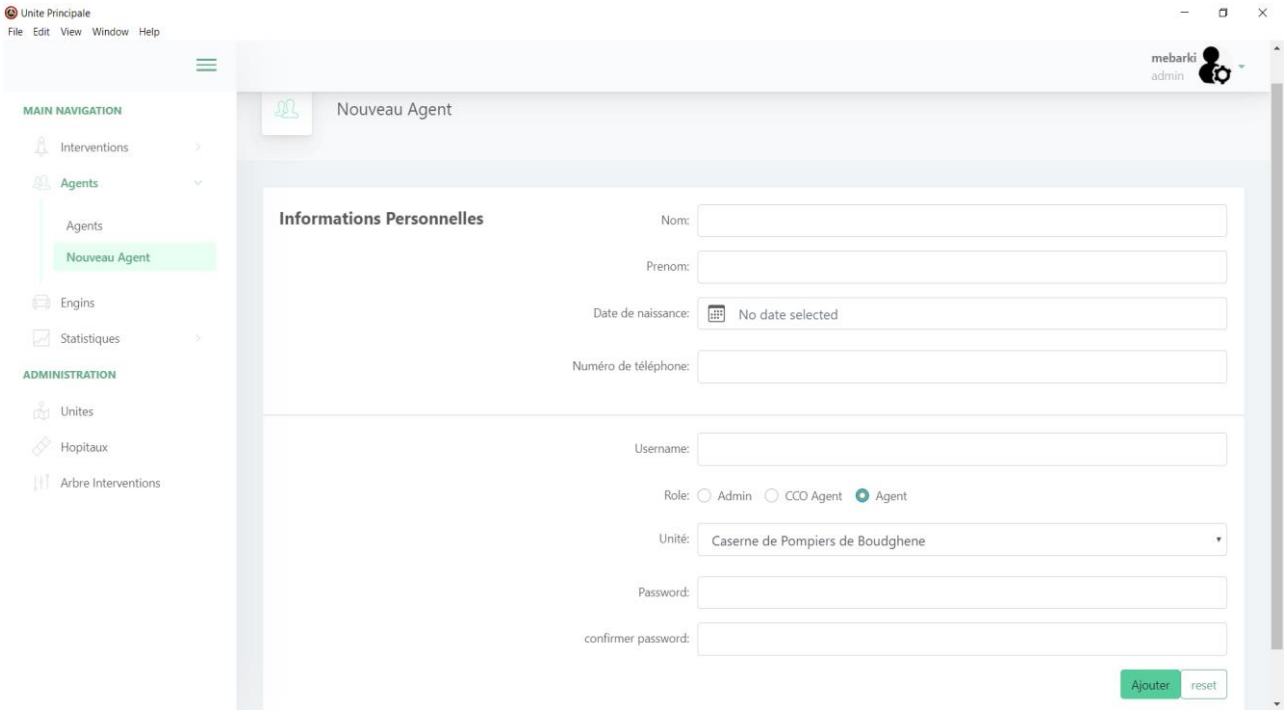


Figure 16 Unité principale - Nouveau agent

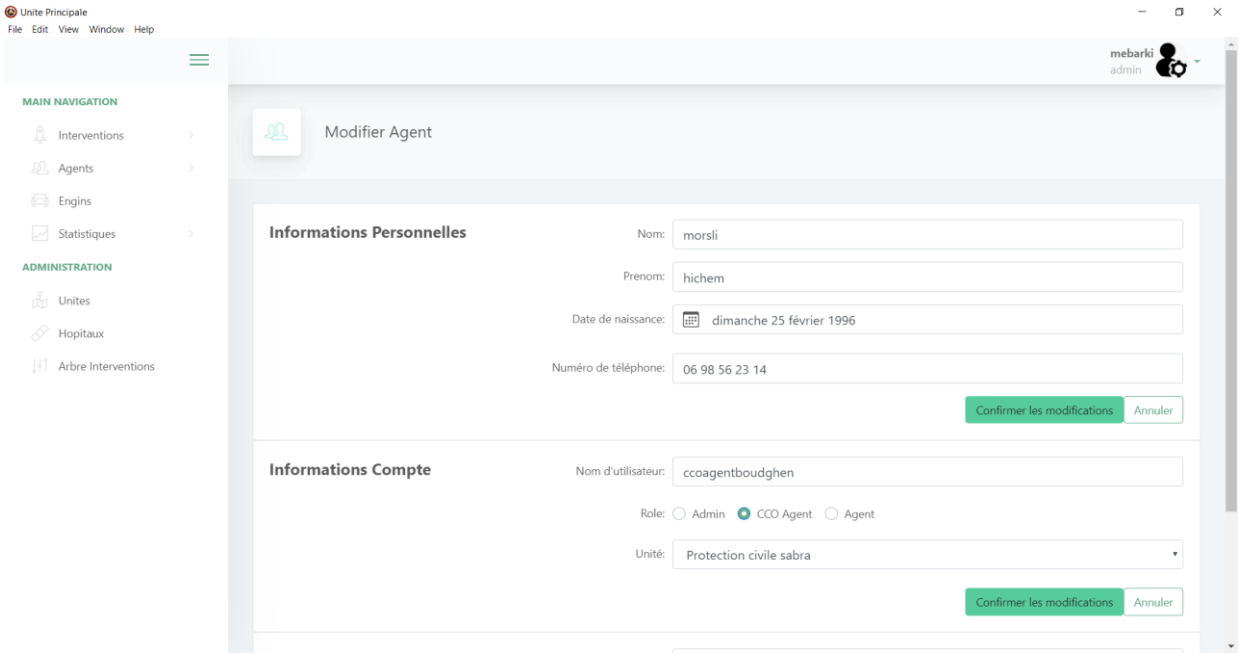


Figure 17 Unité principale - Modifier agent

2. Agent de CCO : Un agent de CCO de l'unité principale peut **que consulter** toutes les données **de son unité et ses unités secondaires sauf les données privées pour l'admin**. En plus il peut gérer les interventions demandées :

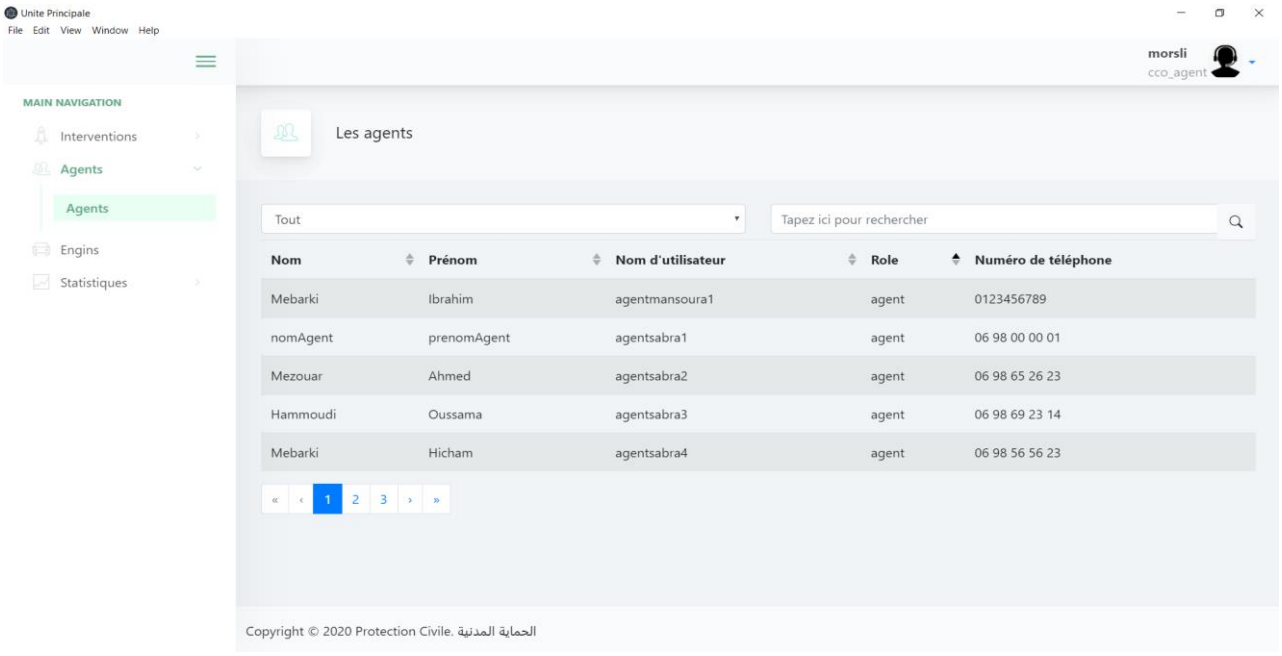
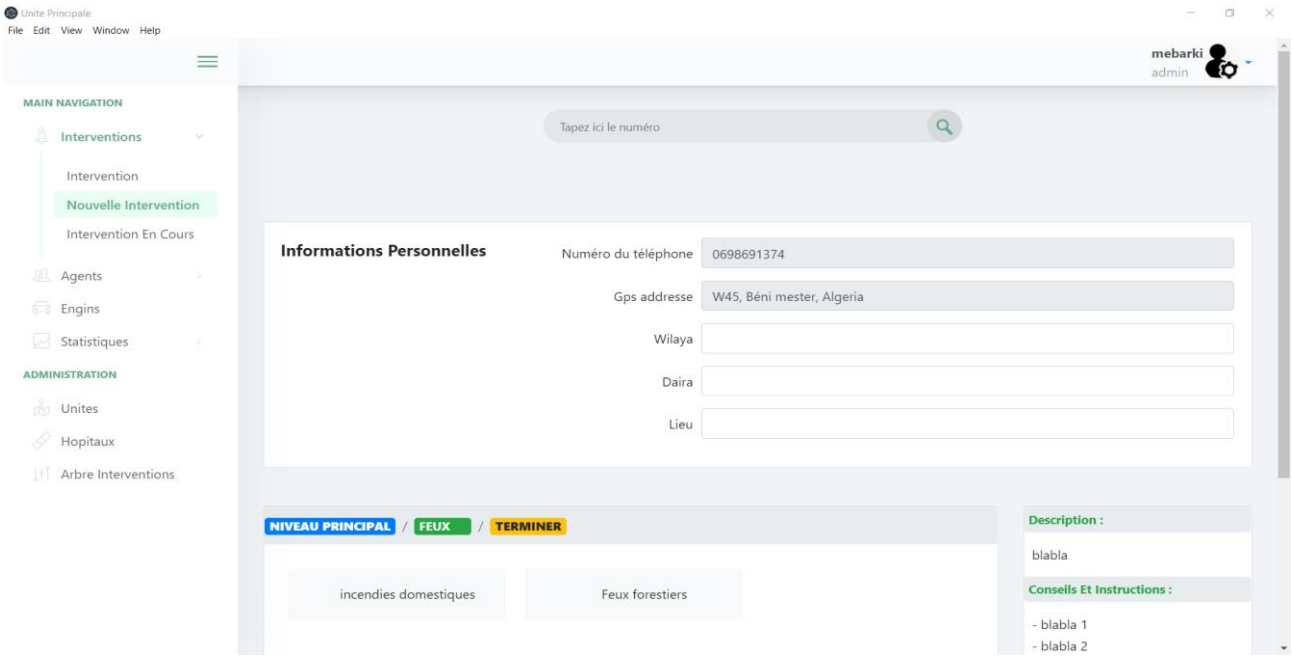


Figure 18 List agents pour l'agent de CCO - unité principale



.

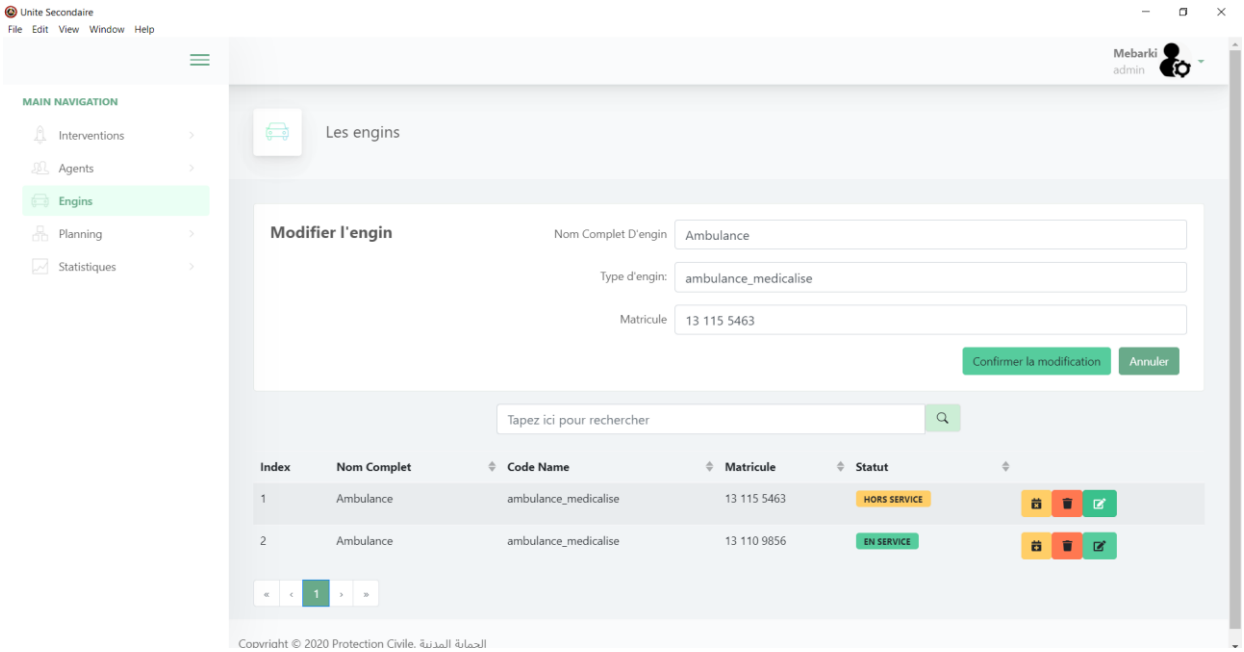
Figure 19 Page d'intervention - Unité principale

4.2.2.2 SecondaryUnit_ProtectLife :

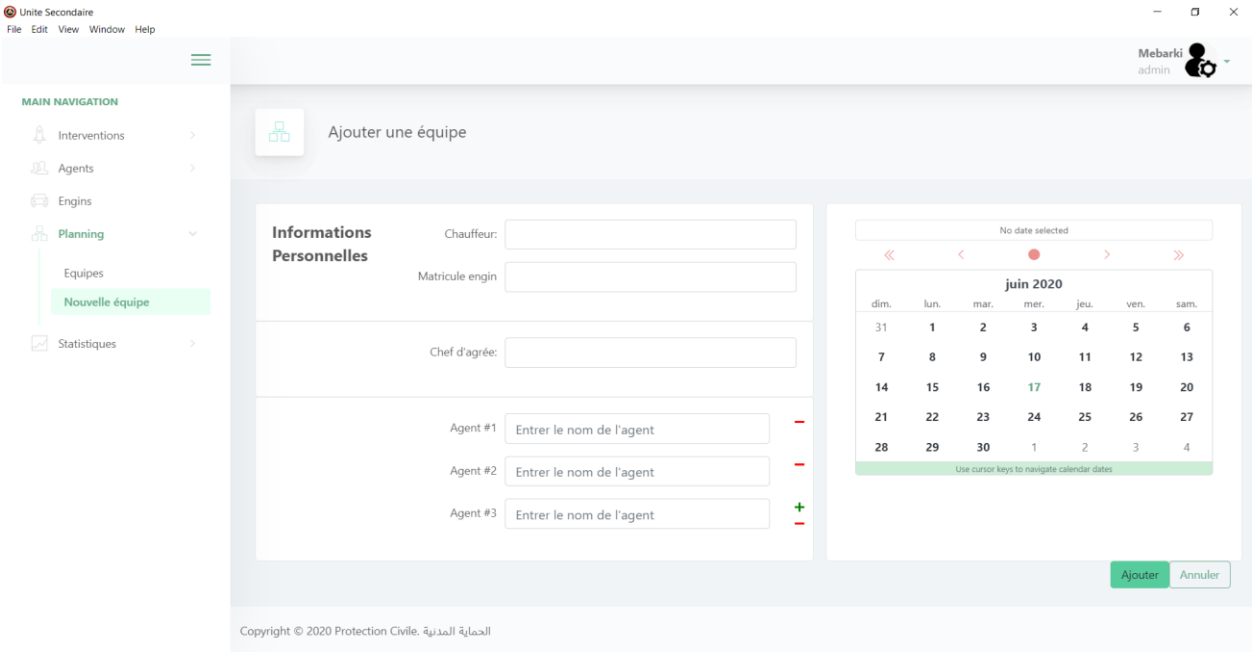
Cette deuxième application est dédiée à chaque unité principale. Cette application est utilisée pour pouvoir effectuer les opérations comme la gestion des agents, des engins et la plus importante la gestion des interventions de sa propre unité.

Cette application est utilisée par deux types d'utilisateur :

1. Administrateur : Un administrateur de l'unité secondaire peut gérer que les données de son unité. Par exemple il peut ajouter, modifier ou supprimer des engins de son unité comme dans la figure suivante :



2. Agent de CCO : L'agent de CCO de l'unité secondaire peut gérer le planning des équipes intervenante et les interventions mais il peut que consulter les agents et les engins.



Maintenant que nous avons introduit et défini ce que chacune des deux applications peut faire, nous allons détailler les technologies utilisées. Tout d'abord, nous avons utilisé framework de javascript **VueJs** pour les raisons suivantes :

1. Application mono page dite **SPA** (Single Page Application) pour ne pas charger les données les balises répéter comme le header et le sidebar de notre application qui va nous permettre d'économiser la taille de l'application.
2. Avoir une architecture modulable et facile à gérer
3. Avoir un chargement plus rapide que les autres frameworks connue comme JQuery

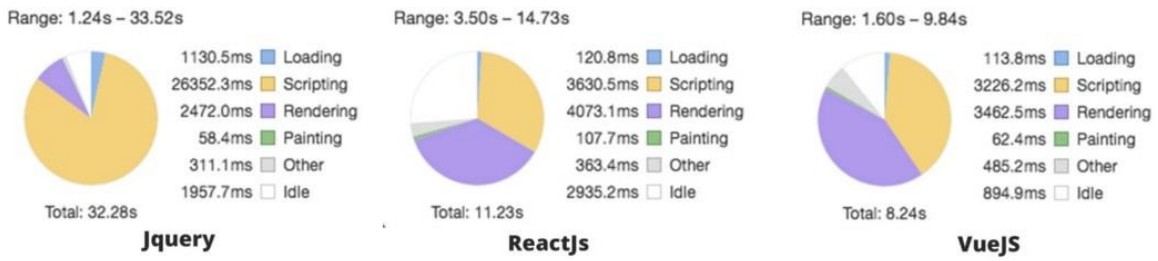


Figure 20 VueJs vs JQuery vs ReactJs [18]

Ensuite, nous avons utilisé **Bootstrap-vue**[19] qui est un framework de **Bootstrap 4**[20] et VueJs afin de ne pas utiliser JQuery qui est trop lent pour un système ou l'optimisation de temps est la plus importante. En ce qui concerne la communication entre ces deux applications web et le serveur **API REST**[21], nous utilisons **Axios**[22] plugin au lieu du **XMLHttpRequests** [23] fonctionnalité native qui existe en JavaScript.

4.2.3 Applications Mobiles Android

Pour la création des deux applications mobiles nous avons pensé à aller vers la plateforme Android afin de pouvoir gérer notre temps restant pour l'implémentation et beaucoup plus parce que l'Android qui possède le plus grand marché des smartphones en Algérie. Nous avons créé deux applications mobiles suivantes :

4.2.3.1 Mob_ProtectLife

Cette application est une application publique s'exécutant dans l'arrière-plan du smartphone en attendant un appel émis vers le 14 (numéro de la protection civile) pour envoyer les cordonnées exacte à l'API_ProtectLife sous un format JSON à l'aide du GPS du smartphone.

4.2.3.2 Agent_ProtectLife

Cette application est dédiée aux équipes intervenantes pour pouvoir gérer les interventions comme suivre le meilleur chemin vers l'appelant et donner les informations en temps réel pour les unités qui suivent les interventions.

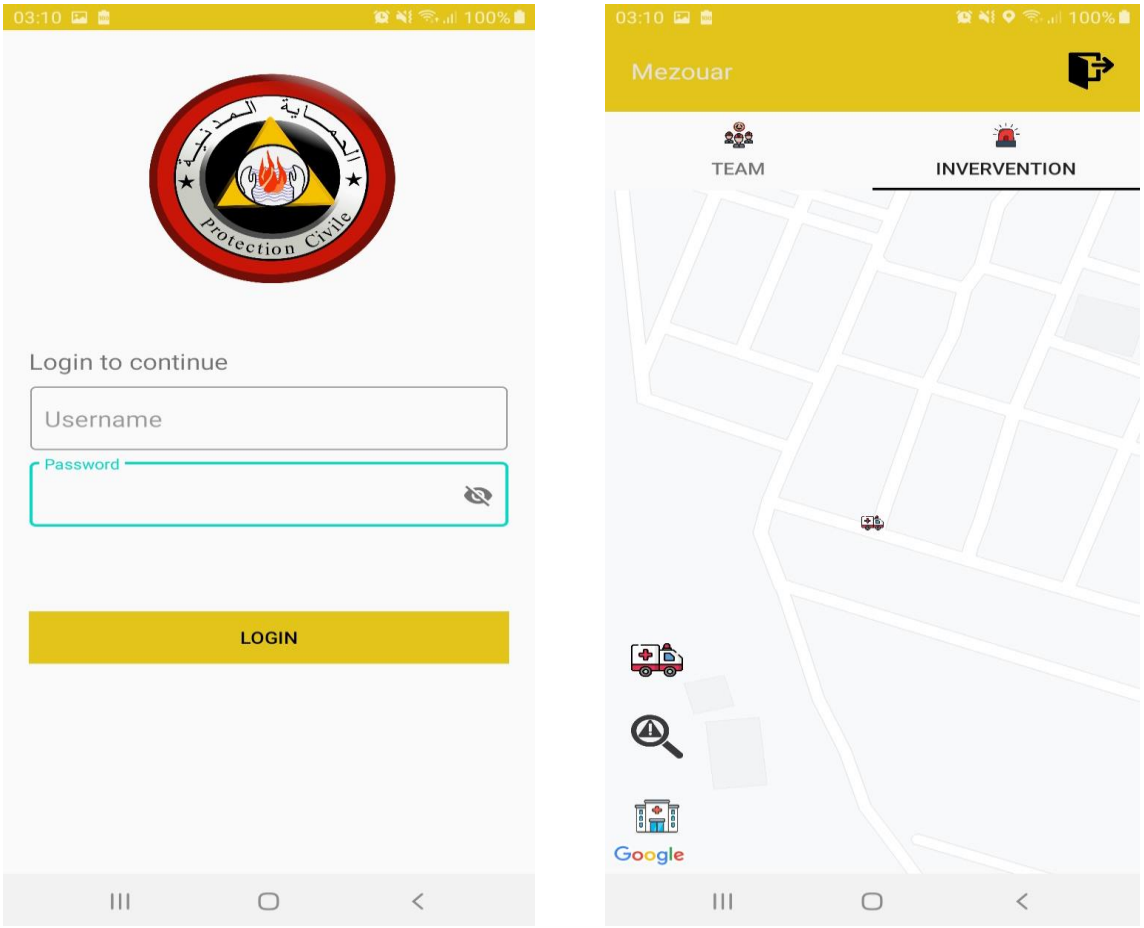


Figure 21 Agent_ProtectLife Application

Les applications Android natives peuvent être développées en utilisant deux langages de programmation Java / Kotlin, Notre choix est Java et qui est basé sur la connaissance que nous avons. L'application mobile jouera le rôle de vues (V) pour notre architecture MVC du Serveur, car elle envoie des requêtes à l'API_ProtectLife pour récupérer ou modifier des données. Nous avons aussi utilisé les technologies suivantes :

Retrofit : est bibliothèque facilitant le téléchargement de données JSON ou XML à partir de l'API. Une fois les données téléchargées, elles sont analysées dans l'objet POJO (Plain Old Java Object) qui est défini pour chaque requête. Pour la configuration de cette bibliothèque, il suffit d'ajouter ces deux lignes dans le fichier Gradle précisément dans la section dependencies et le studio Android fera le travail de synchronisation de la bibliothèque avec notre projet d'application.

```
dependencies {
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
}
```

Dans l'exemple suivant, nous montrons comment nous avons reçu les données via Retrofit lorsqu'un agent se connecte


```
{
        "id_unite": "5ee0bc91fa70ad13947afea6",
        "agent_id": "5ee6628f9550ca2fc40bbe97",
        "agent_nom": "Mohamed",
        "agent_role": "agent",
        "agent_username": "Mohamed_mohamed"
}
```


Une fois que le Retrofit a récupéré les données sous format JSON, il les convertit en POJO et initialise un objet de la class Agent prédéfini avec les valeurs obtenues à partir de l'analyse JSON :

```
package com.pfe.enginapp.models;

public class Agent {
    String agent_nom;
    String agent_prenom;
    String agent_role;
    String agent_username;
    String agent_id;
    String id_unite;
    String type;

    public String getAgent_prenom() {
        return agent_prenom;
    }

    public void setAgent_prenom(String agent_prenom) {
        this.agent_prenom = agent_prenom;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getId_unite() {
        return id_unite;
    }
```

```
    public  void setId_unite(String id_unite) {
        this.id_unite = id_unite;
    }

    public String getAgent_id() {
        return agent_id;
    }

    public void setAgent_id(String agent_id) {
        this.agent_id = agent_id;
    }

    public String getAgent_nom() {
        return agent_nom;
    }

    public String getAgent_role() {
        return agent_role;
    }


    public String getAgent_username() {
        return agent_username;
    }

    public void setAgent_nom(String agent_nom) {
        this.agent_nom = agent_nom;
    }

    public void setAgent_role(String agent_role) {
        this.agent_role = agent_role;
    }

    public void setAgent_username(String agent_username) {
        this.agent_username = agent_username;
    }

}
```

Socket.IO : est une bibliothèque qui permet une communication en temps réel, bidirectionnelle et basée sur des événements entre le navigateur/application mobile et le serveur.[24]

```
dependencies {
    implementation('com.github.nkzawa:socket.io-client:0.5.0') {
        exclude group: 'org.json', module: 'json'
     }
}
```

Cette bibliothèque détecte les évènements déclenchés de l'API_ProtectLife afin d'informer l'agent. Comme la notification d'une intervention reçu à l'équipe pour intervenir.

L'événement qui sera déclenché comme suit :

```
io.emit("interventionStart", intervention.id_team, intervention._id)
```

Toutes les applications Agent_ProtectLife écoutent cet événement à l'aide de la fonction suivante :

```
socket.on("interventionStart",          new          Emitter.Listener()          {
    @Override
    public void call(final Object... args) {
                // code à exécuter lors de la capture de cet événement
        }
    }
});
```

Google Maps API : est un outil permettant d'afficher la carte dans l'application et effectuer des opérations comme durée routier entre deux points que nous utilisons souvent pour trouver la plus proche équipe. Cette bibliothèque permet aussi d'extraire les coordonnées GPS en temps réel de l'appareil [25].

```
dependencies {
    implementation 'com.google.android.gms:play-services-location:15.0.1'
}
```

4.3 Présentation d'un cas réel d'une intervention en utilisant notre système

Après avoir répondu d'un demandeur de secours au téléphone, l'agent de CCO accède à la page **Nouvelle intervention**. Puis il écrit le numéro de l'appelant dans la barre de recherche pour avoir la géolocalisation comme montré dans la figure suivante :

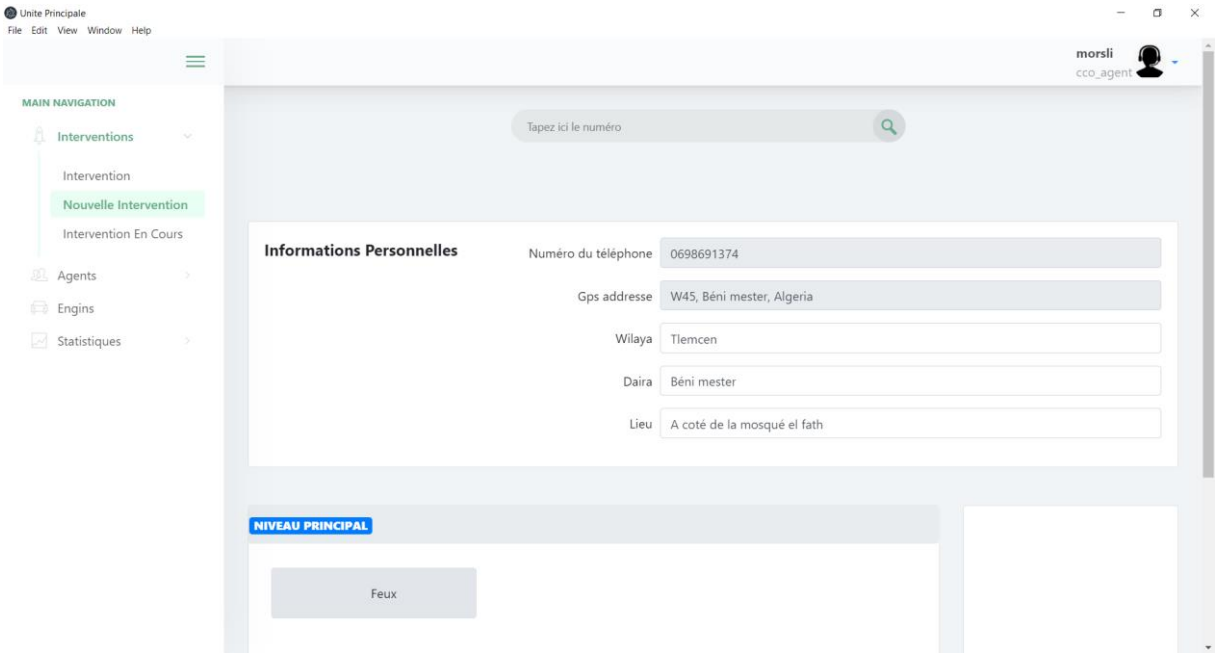


Figure 22 Nouvelle intervention - Unité principale (Partie 1)

Ensuite l'agent remplis les champs wilaya, daïra et lieu pour confirmer la position de l'appelant et après il choisit le type de secours qui est sous format arbre. Et au dernier choix le système affiche les unités les plus proches de l'appelant afin de pouvoir transférer l'intervention à l'unité secondaire

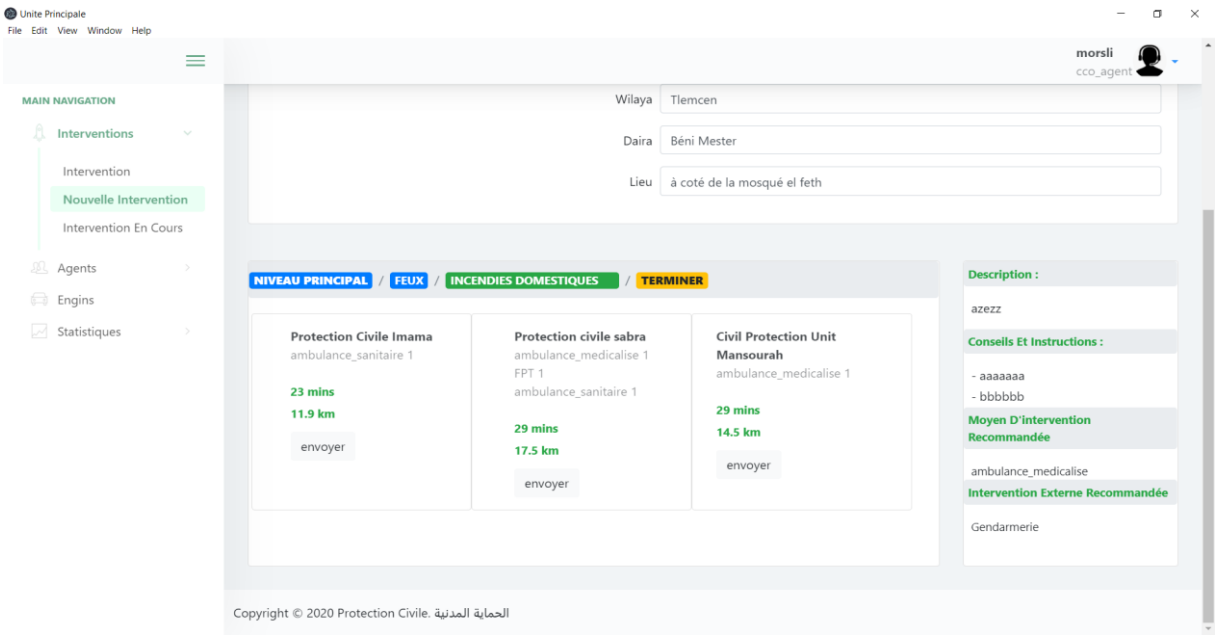


Figure 23 Nouvelle intervention - Unité principale (Partie 2)

Ensuite, l'unité secondaire reçoit l'intervention comme suit :

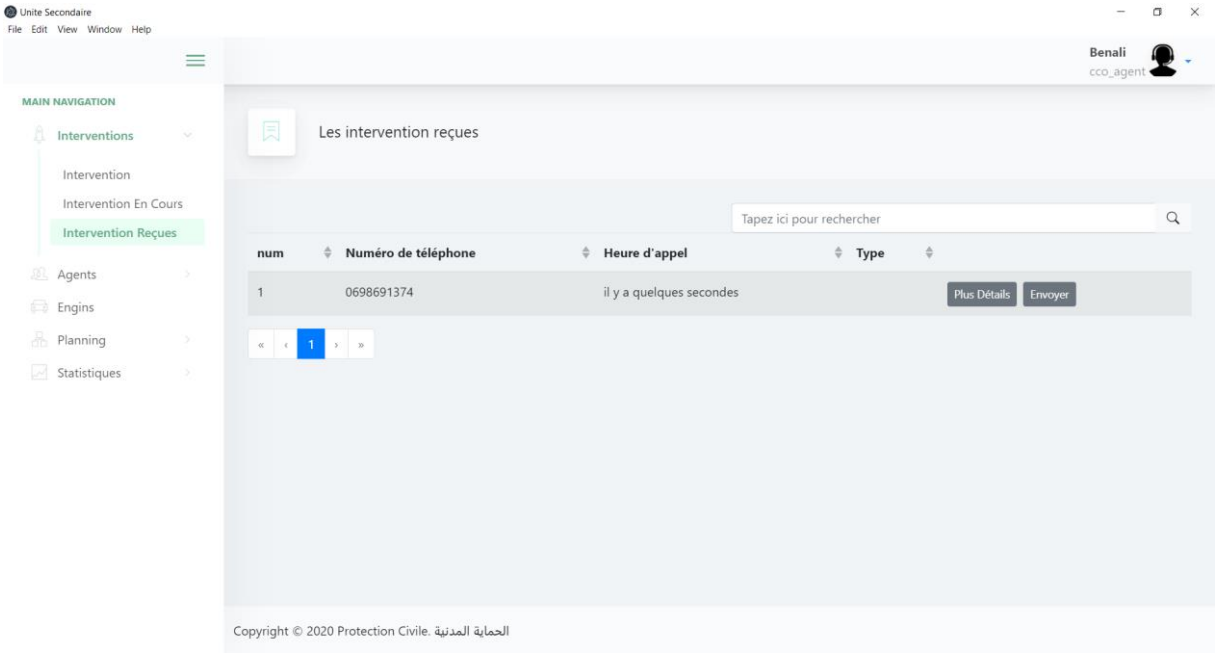


Figure 24 Intervention Reçue - Unité Secondaire (Partie 1)

En cliquant sur envoyer, le système affiche la liste des équipes avec la durée pour arriver à l'appelant comme indiqué dans la figure suivante :

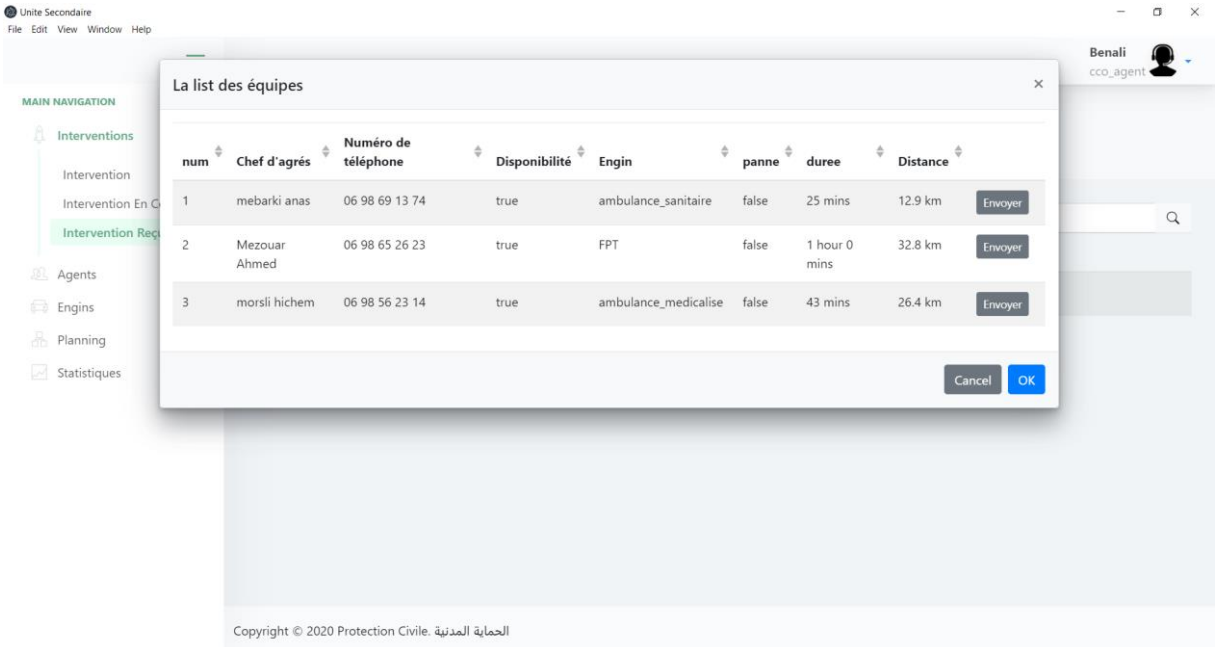


Figure 25 Envoyer l'intervention au chef d'agrée

Maintenant l'équipe intervenante choisie reçoit une notification de l'intervention reçue et la confirmer comme suite :

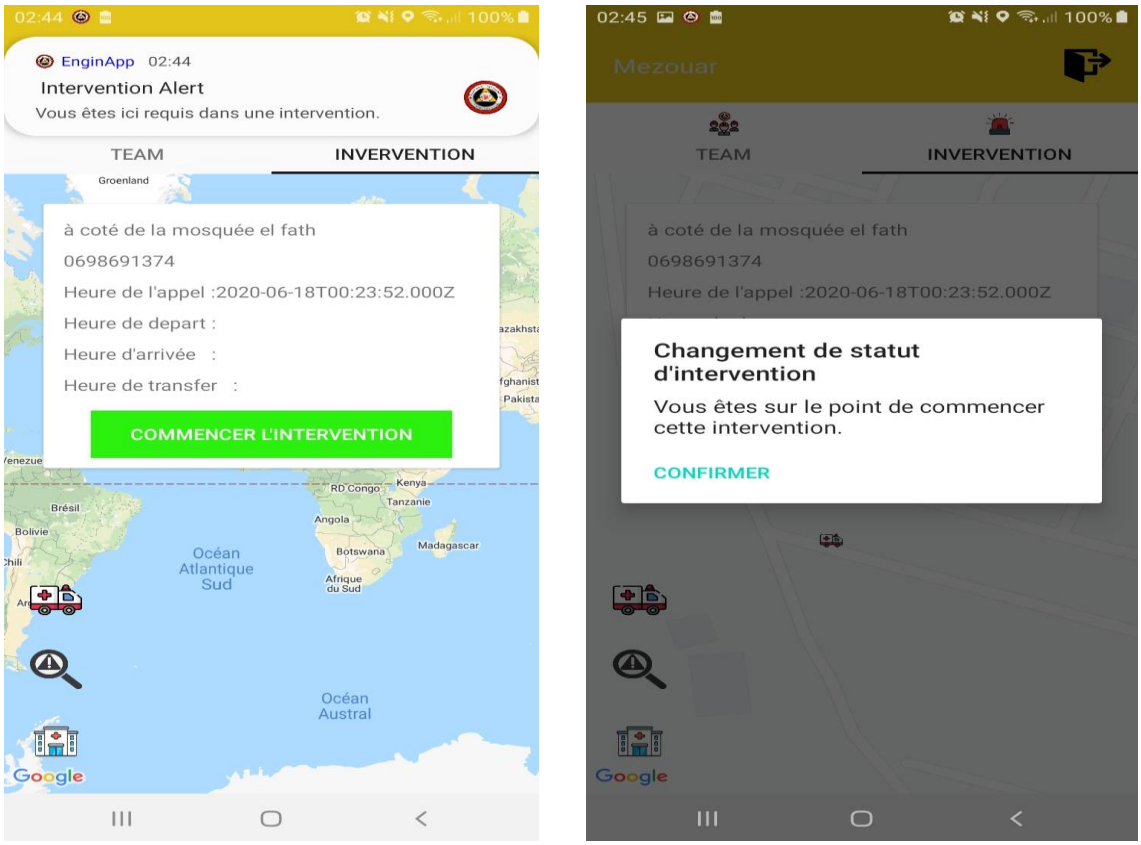


Figure 26 Intervention reçue - Agent_ProtectLife

Maintenant, le chef d'agrée doit toujours effectuer des opérations sur l'application comme commencer l'intervention, Arriver à destination, transfère vers l'hôpital … pour chaque opération les agents de CCO de l'unité principale et l'unité secondaire reçoivent des notifications pour le suivie de cette intervention en temps réel.

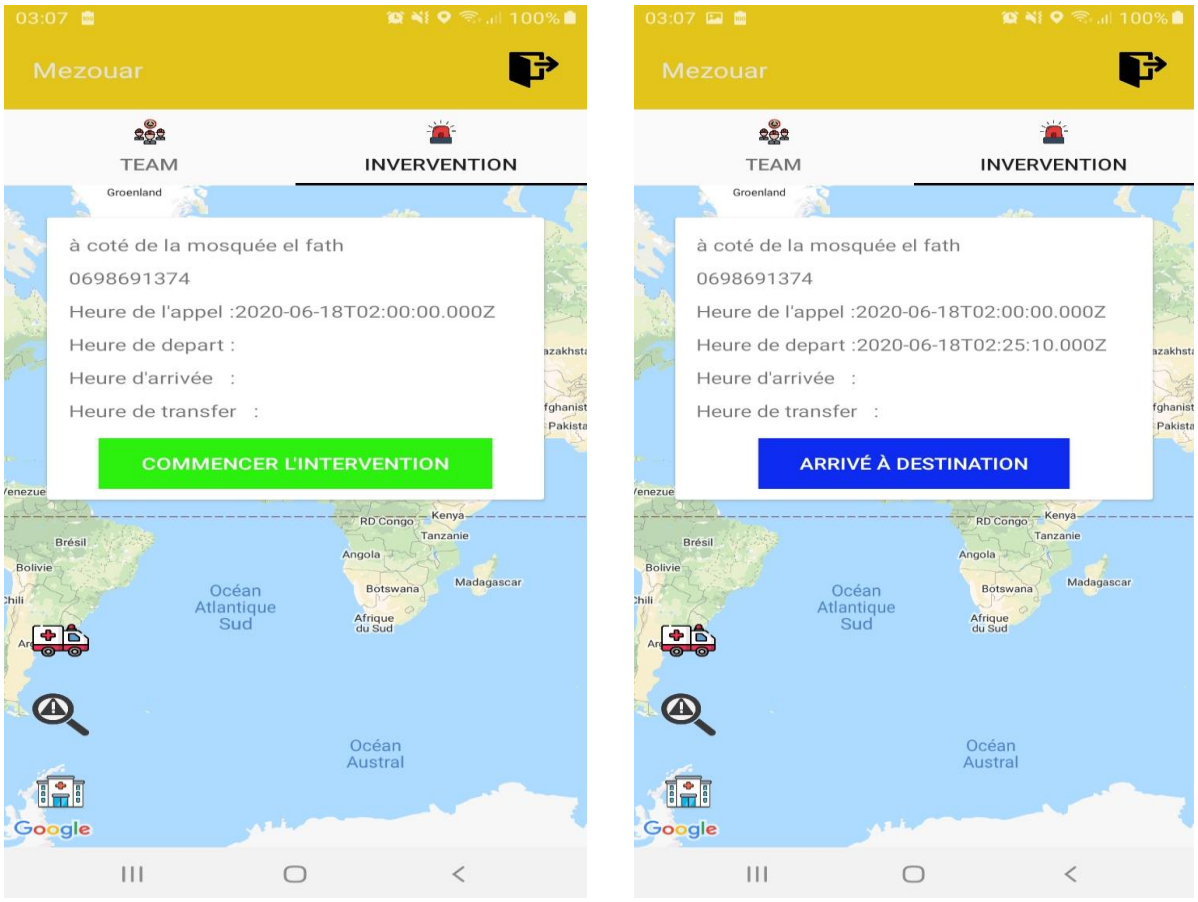


Figure 27 Opération effectuer par le chef d'agrée - Agent_ProtectLife

Au final le chef d'agrée doit écrire le bilan de l'intervention pour terminer cette intervention.

4.4 Conclusion

Dans ce chapitre nous avons détaillé la partie d'implémentation afin de pourvoir comprendre l'importance de notre travail qui nous a permis d'avoir un premier aperçu du domaine et d'acquérir des compétences qui nous seront utiles dans la vie professionnelle.

CONCLUSION GENERALE

Conclusion générale

L'objectif de notre travail était de participer au développement des applications pour la gestion des interventions, qui a pour but de répondre aux interventions en meilleur temps de réponse en proposant des outils et des fonctions, ce qui nous a amené à mettre en œuvre quelques solutions informatiques dans le développement Web.

La première partie du projet était consacré à l'étude et analyse des besoins, et nos critiques du système existant dans le but de connaitre la difficulté et la complexité de chaque étape de l'organisation. Cette étape nous a permis de décomposer le projet en plusieurs petites parties, de plusieurs fonctions.

La seconde partie était consacrée au développement de l'application en elle-même. Cette étape comprend l'analyse, la conception et l'implémentation des fonctions qui constitue notre système, qui sont la modélisation de la base de données et le développement de l'application.

La troisième partie était pour l'implémentations de nos 5 applications. Dans cette partie nous avons implémenté des applications en utilisant les meilleurs outils de développement pour optimiser le temps de réponse qui est trop important pour les interventions. Nous avons aussi implémenté des applications qui peuvent être encore plus développer par d'autre développeur grâce à l'architecture que nous avons choisi. L'API REST de notre système ProtectLife peut communiquer avec un nombre illimité des applications avec la garantie des droits et les meilleurs outils de sécurités.

Au final, avec notre système nous comptons d'être parmi les héros en sauvant plusieurs vies qui sont trop cher pour l'Algérie

Comme perspective de notre travail, nous comptions ajouter d'autres fonctionnalités à savoir :

1. La connexion de notre système avec les caméras des surveillances et les drones pour visualiser l'endroit de l'appelant.
2. La connexion de l'application desktop avec un téléphone analogique.
3. La possibilité d'envoyer les données de la localisation sans devoir avoir l'abonnement de connexion.

Références Bibliographique

[1] Protection civile - http://www.protectioncivile.dz/. Dernière visite : 15 juin 20

[2] NoSQL - https://openclassrooms.com/fr/courses/4462426-maitrisez-les-bases-de-donnees-nosql. Dernière visite : 15 juin 20

[3] SQL faiblesse - Fondamentaux pour le Big Data - Télécom ParisTech - **Pierre Senellart**. Dernière visite : 15 juin 20

[4] Type NoSQL - https://www.digora.com/fr/blog/definition-base-nosql-datastax-mongodb .Dernière visite : 15 juin 20

[5] MongoDB - https://www.mongodb.com/fr .Dernière visite : 15 juin 20

[6] Analyse des besoins - http://specief.org/index.php/ingenierie-des-exigences/ Dernière visite : 15 juin 20

[7] Diagramme de cas d'utilisation - https://www.uml-diagrams.org/use-case-diagrams.html. Dernière visite : 15 juin 20

[8] Diagramme de séquence https://www.uml-diagrams.org/sequence-diagrams.html. Dernière visite : 15 juin 20

[9] Diagramme de classe - https://www.uml-diagrams.org/class-diagrams-overview.html . Dernière visite : 15 juin 20

[10]Visual Studio Code - https://code.visualstudio.com . Dernière visite : 15 juin 20

[11]Postman - https://www.postman.com/ . Dernière visite : 15 juin 20

[12]GitHub - https://github.com/ . Dernière visite : 15 juin 20

[13] Modelio - https://www.modeliosoft.com/ . Dernière visite : 15 juin 20

[14]Android Studio - https://developer.android.com/studio . Dernière visite : 15 juin 20

[15] Trello - https://trello.com/ . Dernière visite : 15 juin 20

[16] Figure 12 - https://aglowiditsolutions.com/blog/node-js-vs-php/ - **RONAK PATEL** . Dernière visite : 15 juin 20

[17] Figure 13 -https://medium.com/@varshney.shivam786/node-js-vs-php-for-backend-4078a3f65741- SHIVAM VARSHNEY . Dernière visite : 15 juin 20

[18] Figure 19 - https://medium.com/thothzocial-engineering/rendering-speed-performance-challenge-with-famous-front-end-framework-196c876a68af - **warat wongmaneekit**. Dernière visite : 15 juin 20

[19] Bootstrap-vue - https://bootstrap-vue.org/

[20] Bootstrap 4 - https://getbootstrap.com/

[21] API Rest - https://restfulapi.net/

[22] Axios - https://www.npmjs.com/package/axios

[23] XMLHttpRequests - https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[24] Socket.IO - https://socket.io/

[25] Google Maps API - https://developers.google.com/maps/documentation/javascript/tutorial

Résumé

La protection civile a pour mission de protéger la vie des citoyens, leurs biens ainsi que l'environnement. Notre travail consiste à faciliter cette mission à ses différents agents pour avoir un meilleur suivie des interventions et encourager les autres services de l'Algérie d'aller vers le digital.

Pour cela, nous avons conçu et réalisé notre système baptisé « ProtectLife » . Afin d'atteindre nos objectifs, nous avons réalisé 4 applications (2 applications web et 2 applications mobiles). Les deux premières sont dédiées aux services de protection civile pour faciliter la gestion des différentes interventions ; quant aux applications mobiles, une est dédiée pour l'agent intervenant et l'autre au citoyen pour avoir sa localisation exacte au cas d'appel au 14. Afin de pouvoir échanger et traiter les données entres les ces 4 applications et notre base de données d'une manière sécurisée, nous avons mis en place l'API REST « API_ProtectLife ».

Toutefois, vue que le nombre d'interventions ne cesse de croitre, nous avons opté pour une base de données NoSQL qui repond à notre besoin.

Mots clés : protection civile, mongodb, API REST, application mobile

Abstract

The mission of civil protection is to protect the lives of citizens, their property and the environment. Our work consists in facilitating this mission to its various agents to have a better follow-up of the interventions and to encourage the other services of Algeria to go digital.

For this, we have designed and implemented our system called "ProtectLife". In order to achieve our objectives, we have created 4 applications (2 desktop applications and 2 Android applications). The first two are dedicated to the civil protection services to facilitate the management of the different interventions; as for the Android applications, one is dedicated to the intervening agent and the other to the citizen to have his exact location in case of call to 14. In order to be able to exchange and process data between these 4 applications and our database in a secure way, we have implemented the REST API "API_ProtectLife".

However, as the number of interventions continues to grow, we have opted for a NoSQL database that meets our needs.

Keywords : civil protection, mongodb, API REST, Android Application

ملـــخـــص

تتمثل مهمة الحماية المدنية في حماية حياة المواطنين وممتلكاتهم والبيئة. يتمثل عملنا في تيسير هذه المهمة لمختلف وكلائها من أجل متابعة التدخلات على نحو أفضل وتشجيع الدوائر الأخرى في الجزائر على أن تصبح رقمية.

ولهذا الغرض ، قمنا بتصميم وتنفيذ نظامنا المسمى « ProtectLife». ومن أجل تحقيق أهدافنا ، أنشأنا 4 تطبيقات (تطبيقان لسطح مكتبيان وتطبيقات أندرويد). الأولان مكرسان لخدمات الحماية المدنية لتيسير إدارة التدخلات المختلفة ؛ أما بالنسبة لتطبيقات أندرويد ، واحد مكرس للعميل المتدخلين والآخر للمواطن للحصول على موقعه الدقيق في حالة الاتصال إلى 14. وبغية التمكن من تبادل ومعالجة البيانات بين هذه التطبيقات الأربعة وقاعدة بياناتنا بطريقة آمنة ، قمنا بتنفيذ برنامج REST API "Api_ Protectlife"

إن استمرار التزايد في عدد التدخلات كان سبب من الأسباب التي فرضت علينا استعمال قاعدة بيانات من نوع NoSql .

الكلمات المفتاحية: ***الحماية المدنية ;***