Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Informatique

Travail de Bachelor

2017

Rapport

SEMS

Smart Environment Monitor System

Etudiant : William Greppi

Superviseurs

Pascal Bruegger

François Kilchoer

Expert : Bernard Frossard

Fribourg, juillet 2017

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Historiques des versions

Version 1 : Version du rapport de début de documentation du serveur de communication REST et de la base du serveur métier.
Version 2 : Version du rapport comprenant la base de la documentation du serveur de communication REST.
Version 3 : Version du rapport comprenant la base de la documentation de la concurrence et de l'algorithme de traitement des alarmes.
Version 4 : Version finale du rapport, comprenant la documentation de l'application Web de monitoring ainsi que les tests fonctionnels et utilisateurs globaux.

Sommaire

1. Introduction

Dans ce chapitre est introduite la nature du document, suivi des informations primaires sur le projet ainsi que de son contexte.

Sommaire

1.1 Document

Dans ce rapport est décrit le travail de Bachelor nommé SEMS. Ce projet fait partie d'un ensemble de trois applications, deux mobiles et un serveur. Il consiste à l'établissement d'un serveur qui va gérer la communication entre le personnel soignant et accompagnants avec les personnes qui nécessitent un suivi.

1.2 Contexte

De nos jours les infirmiers (-ères), aide-soignants (-es) ou éducateurs (-trices) sont confronté(e)s à un nombre de patients ou de résidents à s'occuper de plus en plus important. Dans les homes et les lieux de résidence pour personnes âgées ou handicapés, le personnel est souvent en sous-effectif et la supervision de ces personnes devient problématique. De plus dans le cas des personnes âgées, il est assez difficile pour elles de passer d'une vie indépendante à une vie supervisée, assistée. L'idée est de donner une certaine indépendance tout en monitorant les aspects sécuritaires comme une chute, une inactivité suspecte ou, pour les personnes atteintes de la maladie d'Alzeimer, un mouvement hors du lieu de vie.

1.3 Description

Le projet SEMS concerne la mise en place d'un système de gestion d'environnement intelligent pour le monitoring des résidents et du corps médical. Des applications mobiles, une concernant les résidents et une concernant le corps médical permettront d'envoyer des informations au serveur et de les recevoir.

L'application des résidents et du corps médical enverra tous les certains temps des informations au serveur. Le rôle du serveur est de définir les cas d'alarmes (sortie du lieu de vie, rythme cardiaque élevé ou chute par exemple) et de les traiter. Un système de priorité sera mis en place permettant de définir l'importance de l'alarme, ce qui définira la personne qui sera contactée. Dans le cas où la tâche est urgente, le système informera la personne du corps médical disponible la plus proche. Dans le cas où la tâche est grave mais pas urgente, ce système appellera la personne responsable. Une application graphique permettra en outre de monitorer le système et de consulter son état.

Le projet global, nommé "EMS" comprend également deux autres travaux de Bachelor en plus de SEMS. Le premier nommé HMMA (Health Monitoring Mobile App) qui est le pendant de l'application mobile RMMA pour le personnel soignant et qui recevra les alarmes. Le second

nommé RMMA (Resident Monitoring Mobile App) qui est l'application pour les résidents et qui enverra les alarmes.

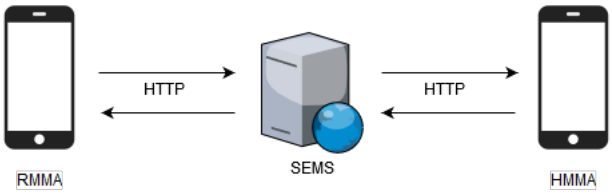

FIGURE 1.1 – Représentation des interactions du système

1.4 Continuité

Le chapitre suivant traitera l'analyse du projet.

2. Analyse

Dans ce chapitre est décrit la partie analyse du projet. Il contiendra notamment la description des spécifications du projet, l'analyse des travaux existants, l'analyse de l'état de l'art, l'analyse de l'architecture du serveur, ainsi que l'analyse l'application de monitoring.

Sommaire

2.1 Spécifications

Ce projet contient quatre objectifs principaux. Ils traitent la partie serveur, la partie algorithmique et l'application graphique de monitoring.

2.1.1 Mettre en place l'infrastructure utilisant uMove

Ce premier objectif consiste à étudier le fonctionnement de uMove et de développer une infrastructure l'utilisant. Cela concerne la création des entités ainsi que la préparation des diverses couches du programme.

2.1.2 Mettre en place un serveur de communication

Ce deuxième objectif consiste à mettre en place un serveur HTTP REST permettant la communication entre les clients et l'infrastructure uMove mise en place dans le premier objectif.

2.1.3 Développer un algorithme de traitement des alarmes

Ce troisième objectif consistera à modifier les actions de(s) observateur(s) du monde afin de traiter les alarmes. Ces alarmes seraient envoyées par les senseurs lorsqu'une personne tomberait du lit, sortirait du lieu de vie ou aurait un rythme cardiaque dangereux.

2.1.4 Développer une application graphique de monitoring des entités

Ce dernier objectif consisterait à développer une application graphique permettant le monitoring des entités. Elle permettrait notamment de consulter la liste des résidents avec leur position et leur état.

2.2 Travaux existants

Précédant ce travail, deux travaux ont été effectués qui pourraient aider à l'établissement du projet. Ces travaux sont la librairie uMove et un travail de bachelor sur la gestion graphique d'environnements intelligents

2.2.1 uMove

uMove est une librairie permettant de concevoir la base d'un système intelligent qui a été effectué par M. Bruegger durant sa thèse de doctorat et améliorée par M. Meuwly et M. Cogno durant leurs travaux de Bachelor. Un système intelligent permet de gérer facilement un environnement comme c'est le cas avec ce projet.
Cette librairie contient diverses couches d'abstraction permettant de clarifier les objectifs de chacune d'elles. Il y a notamment des senseurs, qui sont les sources des éléments, comme la fréquence cardiaque ou la géolocalisation. Des sengets, qui utilisent plusieurs senseurs et qui ont une position réelle. Ensuite, il y a les entités, qui sont la représentation "définitive" de l'entité, par exemple s'il s'agit d'une personne se trouvant à la cafétéria. Au-dessus, il y a les vues, qui permettent d'effectuer un filtre sur un groupe d'entités, comme par exemple une pour consulter tous les patients. Et en haut se trouvent les observateurs. Les observateurs sont les opérations à effectuer à chaque fin d'itérations. Dans le cadre de ce projet, les observateurs permettent de traiter les alarmes envoyées depuis les téléphones mobiles.

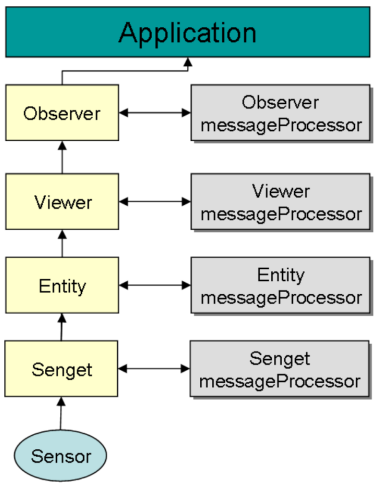

FIGURE 2.1 – Les différentes couches de uMove, thèse de M. Bruegger

Des informations supplémentaires sont consultables au lien suivant : `http://diuf.unifr.ch/pai/umove/`.
Le rapport de thèse se trouve à `https://doc.rero.ch/record/24442/files/BrueggerP.pdf`.
Une description plus détaillée se trouve dans le rapport de M. Meuwly et M. Bruegger.

2.2.2 Travail sur la gestion graphique des environnements intelligents

En 2014, M. Meuwly a effectué un travail de Bachelor dans le thème des environnements intelligents. Durant son travail, il a utilisé la librairie uMove et l'a améliorée. Il a développé une interface graphique permettant de gérer un environnement intelligent. Le rapport de son projet est disponible au lien suivant :
`https://multidoc.eia-fr.ch/record/1762/files/Report_SmartEnvironmentManager.pdf`

2.3 Etat de l'art

L'analyse de l'état de l'art a permis de démontrer que d'autres projets ont été réalisés dans cette idéologie de traiter les alarmes des patients en utilisant un serveur de communication. Deux de ces projets ont été analysés. Ils comprennent également une application mobile pour le patient.

2.3.1 Patient Data Viewer

Patient Data Viewer est un projet représentant exactement la même idée et concept. Une application Android permet de traiter des patients d'un hôpital et informe le serveur de son état. Le serveur transmet alors ces informations aux docteurs.

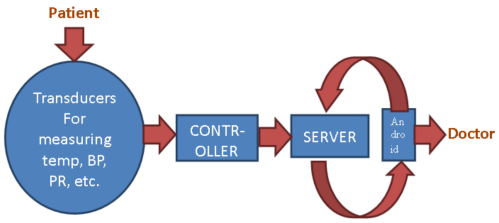

FIGURE 2.2 – Patient Data Viewer, Samyuktha Challa, G. Geethakumari et CSN Prasad

Pour consulter davantage d'informations, `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6139641`

2.3.2 Mobile Health

Mobile Health est un autre projet qui concernerait un concept semblable, mais pour un hôpital. Il y aurait une application cliente, utilisant GPS et senseurs qui communiqueraient directement avec un service d'urgence pour lesquels chaque docteur possède l'application correspondante.
La figure 2.3 représente le schéma du projet en question.

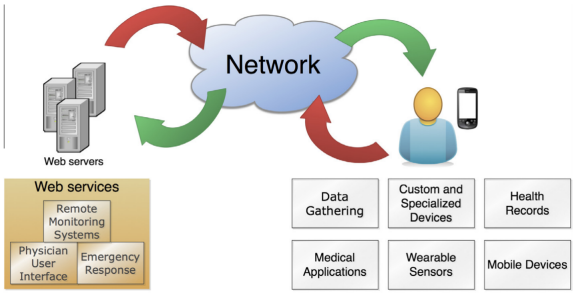

FIGURE 2.3 – Mobile Health, B.M.C. Silva et al. / Journal of Biomedical Informatics 56 (2015) 265–272

Davantage d'informations sont consultables au lien suivant, `http://www.sciencedirect.com/science/article/pii/S1532046415001136`

2.4 Analyse du projet

Dans cette section sera traitée l'analyse au niveau des tâches du projet, cela concerne notamment l'architecture du serveur de communication, l'infrastructure uMove, l'algorithme et l'application de monitoring.

2.4.1 Analyse du serveur de l'environnement intelligent

L'analyse pour la mise en place de l'infrastructure consiste à l'étude de la création du monde et des entités, ainsi qu'à leurs déplacements et au signalement d'alarmes, mais aussi à la compréhension des observateurs.

Données

Les données à gérer ont pu être séparées en cinq éléments démontrés par le tableau 2.2. La figure 2.4 représente ces cinq éléments sous une forme graphique de Mind Mapping.

Résident	Un résident possède un nom, un prénom, un numéro de téléphone, un compte, une ou des maladies, un rythme cardiaque, une activité et une date de naissance.
Responsable	Un responsable possède un nom, un prénom, un compte, les résidents qu'il s'occupe et un numéro de téléphone.
Compte	Un compte est composé d'un nom d'utilisateur, d'un mot de passe et d'un numéro de rôle.
Maladie	Une maladie est composée d'un nom, d'un type et la liste des médicaments pour son traitement.
Médicament	Un médicament contient un nom.

TABLE 2.1 – Analyse des données

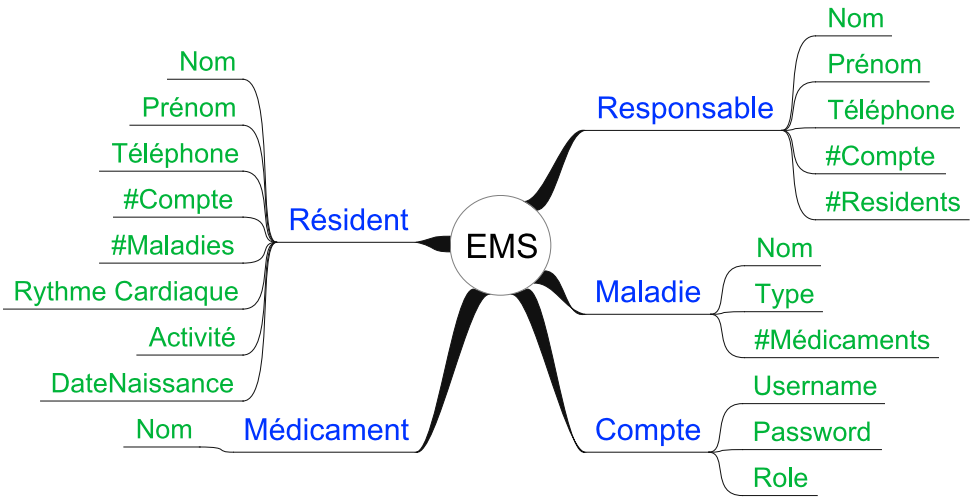

FIGURE 2.4 – Schématisation des données

Système de base de données

Le framework Spring, également utilisé pour la mise en place du serveur REST, permet d'effectuer directement des opérations CRUD en utilisant la technologie Hibernate. La mise en place d'une liaison entre un objet Java et sa représentation dans la base de données est effectués avec deux éléments. Ces éléments sont les entités et les dépôts CRUD des entités.
Hibernate a l'avantage de gérer lui-même les tables de la base de données. Il n'est pas nécessaire de traiter la création des tables, des champs et des contraintes.

Entités Une entité est la classe contenant tous les champs d'une table dans la base de données. Il est défini par l'annotation "@entity" au-dessus de la déclaration de la classe.

Listing 2.1 – Création d'une entité

```
@entity
public class Resident{
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Integer id;
  private String name;

  // ...
}
```

La particularité est l'utilisation de la classe mère `Integer` (ou Long) pour la définition des Id. L'annotation `@GeneratedValue` permet la génération automatique de l'Id.

Dépôts Crud Un dépôt CRUD permet l'exécution d'opérations sur ensemble d'entités. La déclaration d'un dépôt consiste à la création d'une interface qui étend un dépôt CRUD. Cette interface permet aussi l'ajout d'opérations si nécessaire.

Listing 2.2 – Création d'un dépôt CRUD

```
@Repository
public interface ResidentRepository extends CrudRepository<Resident, Long>{
 // ...
}
```

L'utilisation des opérations d'un dépôt CRUD se fait en déclarant un attribut avec l'annotation `@Autowired` comme représenté dans le code 2.3. Il n'est pas nécessaire de créer l'objet.

Listing 2.3 – Liaison d'un dépôt CRUD

```
@Autowired
private ResidentRepository residentRepository;
```

Entités

Deux rôles principaux ont été défini dans le système. Un rôle pour les responsables et un rôle pour les résidents.
Chaque entité contient un attribut du type Object permettant d'ajouter des informations supplémentaires nommées entityExtAttribute. Les données propres aux acteurs pourrait y être ajoutées.

Définition des senseurs

Trois senseurs devraient être définis. Un permettant la récupération du rhytme cardiaque du résident, un permettant la récupération de la position de la personne et un traitant des chutes du résident.

Création du monde

La création du monde en utilisant la librairie uMove et le travail de M. Meuwly consiste principalement à l'utilisation des méthodes suivantes :

createFirstEntity()	Création du monde
createEntity()	Création d'une entité dans le monde
entity.setPolyganalEntity-Geometry(vector)	Définition de la surface géométrique d'une entité
entity.setNewSpatialLocation (coordinates) ;	Définition de la localisation d'une entité

TABLE 2.2 – Analyse des données

L'exemple de code ci-dessous démontre la création du monde et d'une entité, ainsi que l'ajout d'une personne dans l'entité maison.

Listing 2.4 – Exemple de création du monde et d'entités

```
// Creation du monde et d'une maison
Actor world = entityFactory.createEntity(null, "World", "World", "");
Actor house = entityFactory.createEntity(null, "House 104", "House 104",
    world, null);

// Definition geometrique de la maison
Vector<XYZVector> vector = new Vector();
XYZVector subVector = new XYZVector(20, 20, 20);
house.setPolyganalEntityGeometry(vector);
house.setNewSpatialLocation(new Coordinates(0,0, 10,
    CoordinatesFormatEnum.BD72));

// Creation d'une personne dans le systeme uMove
Actor person = entityFactory.createEntity(null, "Person 1", "Person 1",
    house, null);
```

Définition du monde

Pour le début du projet, il a été pensé de partir sur un monde à un seul étage. Le couloir et les salles du D20 représentés dans la figure 2.5 ont été utilisés. Sur la gauche se trouvent les 5 salles de classe qui représenteraient les chambres des résidents. Sur le centre, en (8) se trouve le couloir. Sur la droite en (6) se trouveraient les toilettes et en (7) le séjour des résidents. La figure 2.6 représente le plan de référence du bâtiment.

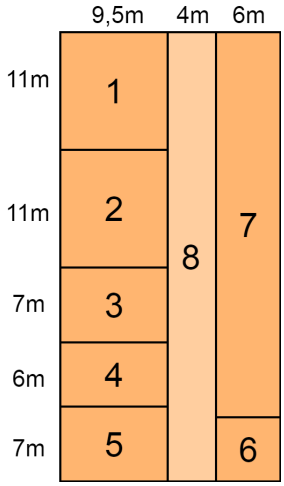

FIGURE 2.5 – Le monde définit pour l'environnement intelligent

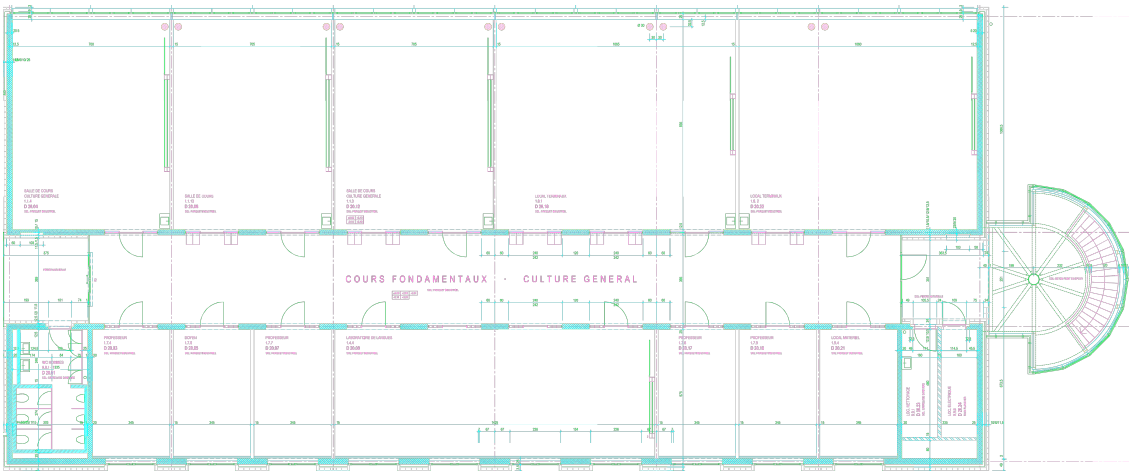

FIGURE 2.6 – Le plan du bâtiment (ged.hefr.ch, 1990)

2.4.2 Architecture du serveur de communication

L'analyse de l'architecture du serveur de communication est séparée en deux parties. Une partie concernant les opérations à traiter et une partie contenant l'étude de fonctionnement d'un serveur HTTP REST.

Technologie

Le serveur de communication respecte le concept d'un serveur HTTP REST. La communication se fait grâce à des HTTP Request et des HTTP Response. Le format JSON a été utilisé pour le formatage des données au lieu du XML, car il est moins verbeux, plus léger et mieux géré par les applications mobiles.
La figure 2.7 représente l'architecture du fonctionnement de la communication entre le serveur et les applications mobiles.

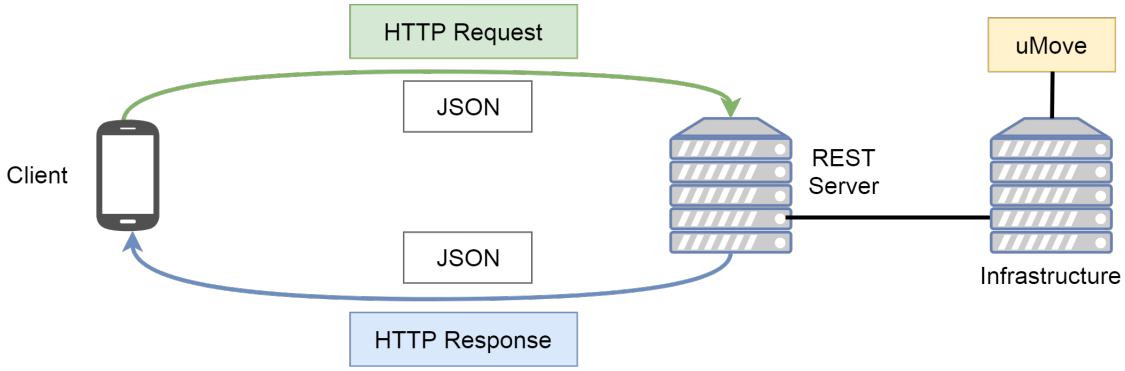

FIGURE 2.7 – Architecture du serveur de communication

Fonctionnement d'un serveur REST

Un serveur REST dispose de plusieurs type d'opérations. Ces opérations sont de types :
— GET : Retourne un ensemble de données
— POST : Envoie un corps de données et retourne un résultat
— PUT : Envoie un corps de données pour la mise à jour et retourne un résultat
— DELETE : Indique les données à supprimer et retourne un résultat.

Opérations à traiter

L'analyse de l'architecture du serveur de communication a permis de démontrer qu'il sera nécessaire d'implémenter au moins les six opérations décrites dans le tableau 2.3 suivant. Le schéma 2.8 représente graphiquement ces opérations.

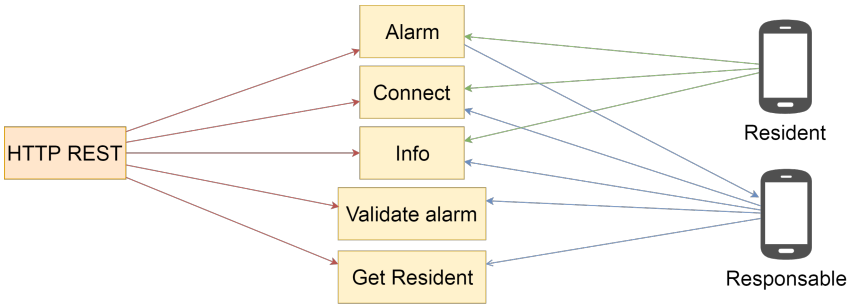

FIGURE 2.8 – Les opérations permises par le serveur HTTP REST

Connexion	Permet la connexion des résidents et du corps médical.
Déconnexion	Permet la déconnexion des résidents et du corps médical.
Info	Reçoit les informations des résidents et le corps médical.
Alarm	Reçoit les alarmes envoyées par les résidents.
Validate Alarm	Reçoit la validation du traitement de l'alarme par le responsable.
getResident	Retourne les résidents responsables.

TABLE 2.3 – Les opérations REST principales

Fonctionnement du serveur

Pour faciliter l'implémentation du serveur de communication REST ainsi que de la communication avec la base de données, le framework "Spring" a été utilisé. Il permet d'implémenter facilement un serveur REST ainsi que les interfaces avec la base de données.
Le framework Spring définit trois éléments principaux, décrit dans le tableau 2.4 suivant.

Boot Application	Main de l'application
Controller	Définit des routes
Object	Contient des informations json

TABLE 2.4 – Les opérations REST principales

Démarrage Le démarrage de l'application consiste à l'exécution de l'application Spring. Il est possible d'instancier un modèle et d'effectuer des opérations avant.

```
@SpringBootApplication
public class Application {

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

Contrôleurs Un contrôleur est une classe contenant des routes. Il se définit avec la dénomination "RestController".

```
@RestController
public class AlarmController {
  // Liste des routes...
}
```

Les routes dans un contrôleur se définissent avec la dénomination "@RequestMapping". Cette dénomination prend essentiellement en pamètres les informations suivantes :

value = "/login"	Chemin de la route.
method = RequestMethod.PUT	Verbe HTTP de la méthode, GET, POST, PUT, etc.

TABLE 2.5 – Attribut de la dénomination des routes

Les informations de Body envoyées par les applications sont demandées par la dénomination @RequestBody. Cette dénomination prend une classe comme paramètre contenant les informations du JSON.
Un exemple se trouve au-dessous de création de routes permettant la connexion d'un utilisateur, la récupération de ses informations et sa déconnexion.

Listing 2.5 – Exemple de création de routes avec le framework Spring

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
public int login(@RequestBody Connection connection){
  return people.add(connection.getPerson());
}

@RequestMapping(value = "/info", method= RequestMethod.POST)
public Person info(@RequestParam(value="id") int id){
  return people.get(id);
}

@RequestMapping(value = "/logout", method= RequestMethod.POST)
public void logout(@RequestParam(value="id") int id){
  people.remove(id);
}
```

Notification Push

FCM pour "Firebase Cloud Messaging", successeur de GCM "Google Cloud Messaging" est une technologie d'envoi de messages dans le Cloud développé par Google. Elle permet d'envoyer des notifications passant par le cloud et permettant aux applications clientes (Android, iOS, Web, etc.) de recevoir une alerte lors de la réception de messages. Le grand avantage de celle-ci est qu'elle est très simple à implémenter côté client et côté serveur.
Le fonctionnement du "Cloud Messaging" est le suivant :

1. Le serveur envoie un message au Cloud de Firebase de Google.
2. Le Cloud de Firebase enregistre le message dans la base de données de Google.
3. Lorsque les destinataires sont disponibles, le Cloud de Firebase récupérera les messages qui leur sont destinés et les leurs enverront.
4. A la réception d'un message, les applications clientes traiteront le message reçu et mettront à jour l'état de l'application (Ajout d'une notification, diffusion d'un message, etc.).

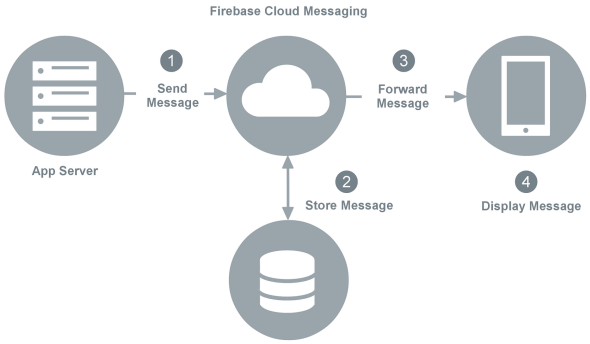

FIGURE 2.9 – Le fonctionnement de FCM. Source : Xamarin Developper

L'envoi de notification Push s'effectue comme dans l'exemple 2.6. Cela consiste à la création d'un client Http envoyant un JSON contenant un message au serveur de Google FCM.

Listing 2.6 – Exemple d'envoi de notifications Push

```
HttpClient httpclient = HttpClientBuilder.create().build();
HttpPost request = new HttpPost(URL_SEND_MESSAGE);
JSONObject sentMessage = new JSONObject();
JSONObject dataContent = new JSONObject();
dataContent.put("message", "hello");
sentMessage.put("data", dataContent);
sentMessage.put("to", "/topics/heia.rmma");
request.setEntity(new StringEntity(sentMessage.toString()));
request.setHeader("Content-Type", "application/json");
request.setHeader("Authorization", "key=" + FCMConsts.FCMKey);
HttpResponse response = httpclient.execute(request);
```

Sécurité

La sécurité est un point important pour la réalisation du serveur de communication. En effet, pour que la connexion soit sécurisée l'utilisation de HTTPs est important pour que les trames envoyées ne soient pas décryptables sur le réseau. Concernant le traitement des sessions, un système de token pourrait être mis en place, avec JWT par exemple. Cela permettrait de définir quelle session est ouverte avec le token concerné. Mais encore, comme la technologie FCM est utilisée pour l'envoi de notification Push, il serait nécessaire de contrôler à ce que ces messages envoyés aux serveurs Cloud de Google ne contiennent pas d'informations confidentielles.

2.4.3 Traitement de la concurrence

Le traitement de la concurrence sera effectué en utilisant un moniteur. Un moniteur permet la définition de conditions utilisant les opérations suivantes :

— Wait : Attente dans une condition, jusqu'à ce qu'un signal le réveille
— Signal : Signalement d'une condition, qui réveille les éléments qui s'y trouvent

Le serveur étant effectué en Java, le moniteur travaillera en mode "Signal-and-Continue". Dans ce mode, l'opération Signal ne va pas faire attendre le thread courant, contrairement au mode Signal-and-Wait.

Exemple de moniteur

Afin de bien comprendre le fonctionnement d'un moniteur en java, le pseudo-code défini en 2.7 démontre le fonctionnement d'un moniteur.

Listing 2.7 – Exemple de création d'un moniteur en Java

```
class Monitor{
  private final Lock lock = new ReentrantLock();
  private final Condition waitCondition = lock.newCondition();

  public void sendMessage(message){
    lock.lock();
    try {
      model.sendMessage(message)
      waitReponse.await()  // Thread sleeping til someone has received it.
      response = getAnswer()
    } finally {
      lock.unlock();
    }
  }

  public void answerMessage(response){
    lock.lock();
    try {
      model.answerMessage(response)
      waitResponse.signal() // Awake the thread waiting response.
    } finally {
      lock.unlock();
    }
  }
}
```

Davantages d'informations se trouvent sur la page Wikipedia suivante : `https://en.wikipedia.org/wiki/Monitor_(synchronization)`

2.4.4 Algorithme de traitement des alarmes

L'analyse de l'algorithme de traitement des alarmes concernera ses fonctionnalités et la définition des priorités.

Fonctionnalités

Les fonctionnalités de l'algorithme sont les suivantes :

- Traitement de la priorité de l'alarme
- Attente de la réponse du responsable demandé, passage au suivant si timeout ou réponse négative
- Choix de la personne à appeler
- Traiter l'évolution de l'alarme (rythme cardiaque qui s'augmente, etc.)
- Traiter l'ajout d'un second type d'alarme pour le même résident.

Notion de priorités

Les priorités des alarmes sont traitées suivant leur type et leur valeur. Une chute de 20 cm est jugée comme moins prioritaire qu'une augmentée du rythme cardiaque supérieur à 150 battements par minute.
La priorité peut s'additionner lorsque des alarmes de plusieurs types surviennent pour le même résident.

Choix des responsables

Les responsables sont choisis suivant le type et l'urgence de l'alarme. Dans certains cas, cela sera la personne la plus proche qui sera appelée, dans d'autres ce sera la personne responsable qui sera appelée.

Cas de chute En cas de chute, c'est la personne la plus proche qui sera appelée. Si cette personne ne répond pas ou que le "timeout" s'est écoulé, la personne suivante la plus proche sera appelée.

Rythme cardiaque En cas d'augmentation du rythme cardiaque, ce sera la personne responsable qui sera appelée. Si cette personne ne répond pas ou que le "timeout" s'est écoulé, la personne la plus proche sera appelée.

Sortie du bâtiment En cas de sortie du bâtiment, ce sera la personne responsable qui sera appelée. Si cette personne ne répond pas ou que le "timeout" s'est écoulé, la personne la plus proche sera appelée.

Définition des opérations

Les opérations qui y seront définies sont les suivantes :

Mise à jour d'un résident La mise à jour d'un résident permet de modifier sa position et augmenter la priorité de l'alarme dans le cas d'un problème cardiaque.

Mise à jour d'un responsable La mise à jour d'un responsable permet de modifier sa position et modifier sa disponibilité.

Déconnexion d'un résident La déconnexion d'un résident désactive l'alarme qui n'est pas encore traitée.

Déconnexion d'un responsable La déconnexion d'un responsable attend que toutes les alarmes reçues par le responsable soient validées.

Timeout résident Un Timeout d'un résident qui a lieu lorsque son status n'a pas été mis-à-jour durant les 30 dernières secondes envoie une alarme.

Timeout responsable Un Timeout d'un responsable qui a lieu lorsque son status n'a pas été mis-à-jour durant les 30 dernières secondes le rend indisponible. Toutes les alarmes qui lui étaient affectées sont basculés à un autre résident en relançant l'algorithme de recherche.

Envoi d'alarme Lorsque le résident envoie une alarme, il va démarrer l'algorithme de recherche de responsable, attendre qu'elle soit acceptée et attendre qu'elle soit validée. A tout échec, il va recommencer sa tâche.

Acception d'une alarme Le responsable ayant reçu une alarme doit l'accepter. S'il l'accepte, le thread d'envoi d'alarme va attendre l'acceptation, s'il refuse sa disponibilité va passer en bas de la liste et l'algorithme va recherche un autre responsable.

Validation d'une alarme Le responsable ayant accepté une alarme doit la valider. S'il la valide, le thread d'envoi d'alarme va se terminer, s'il la invalide, l'algorithme va rechercher un autre responsable.

Envoi d'un message sans connaître le destinataire Le résident peut envoyer un message sans connaître le destinataire. À ce moment, un algorithme va rechercher un responsable disponible qui est son responsable direct, ou le plus proche s'il n'est pas disponible, et va envoyer le message au destinataire.

Envoi d'un message en connaissant le destinataire Le résident ou le responsable va pouvoir directement envoyer un message au destinataire.

Envoi d'une question à un résident Le responsable va pouvoir envoyer une question à un résident. L'envoi d'une question va bloquer la déconnexion jusqu'à ce qu'elle soit répondue.

2.4.5 Application de monitoring

L'analyse de l'application de monitoring concernera les cas d'utilisation, le choix de technologie ainsi que la méthode de communication.

Cas d'utilisation

Le diagramme de cas d'utilisation représenté par la figure 2.10 définit les cas suivants :

1. Se connecter
2. Après s'être connecté, l'utilisateur peut se déconnecter
3. Après s'être connecté, l'utilisateur peut consulter la liste des responsables
4. Après s'être connecté, l'utilisateur peut visionner l'ensemble du monde sous forme de carte du bâtiment
5. Après s'être connecté, l'utilisateur peut consulter la liste des résidents
6. En consultant la liste des résidents, l'utilisateur peut ajouter un nouveau résident
7. En consultant la liste des résidents ou en choisissant un résident sur la carte, l'utilisateur peut consulter le profil complet du résident

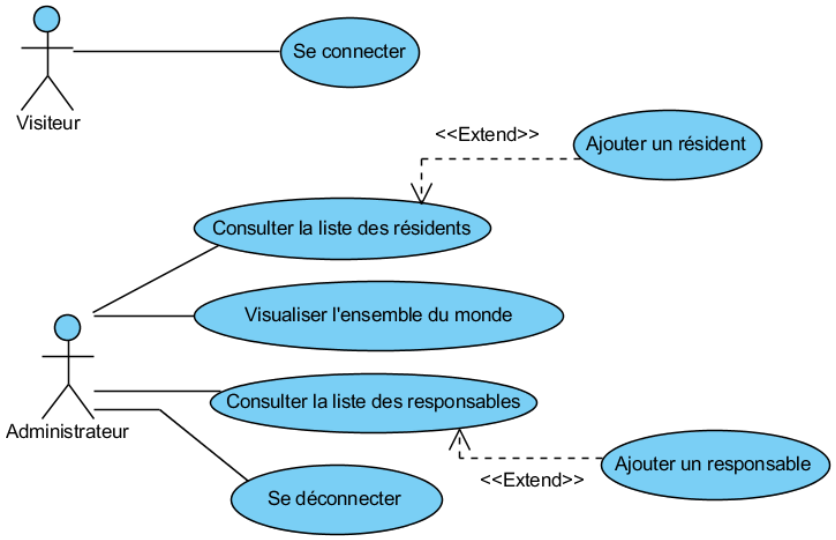

FIGURE 2.10 – Les routes permises par le serveur HTTP REST

Choix technologique

Le choix d'une application Web ou d'une application du type "Desktop" a été analysé. Une application Web permettrait une meilleure évolution future ainsi qu'une utilisation dans un environnement multi-utilisateur. La technologie Java FX serait choisi pour une application du type "Desktop". Cette technologie conviendrait le mieux parmi sa catégorie pour les connaissances actuelles ainsi que la possibilité de réutiliser une partie du code du serveur. La technologie Angular 2 serait utilisé dans le cadre d'une application Web pour les connaissances actuelles ainsi que sa facilité d'utilisation.
Les critères choisis pour le choix sont les suivants. Le tableau 2.6 permet la comparaison directe des solutions. Le choix d'une application Web en Angular 2 a obtenu un bien meilleur résultat que celui d'une application "Desktop".

Réutilisation du code Une application Java FX permettrait la réutilisation d'une partie du serveur effectué en Java EE, ce qui n'est pas le cas d'une application Web en Angular 2 utilisant du TypeScript.

Perspectives d'extension Comme trois rôles ont été définis dans le serveur (résident, responsable et administrateur), une application Web serait plus adaptée pour le traitement des droits d'accès différent contrairement à une application du type "Desktop".

Environnement multi-utilisateur Une application Web permet directement "Out of the box" l'usage dans un environnement multi-utilisateur. Une application Desktop nécessiterait le déploiement de l'exécutable sur tous les postes. (À savoir que les Java Applets permettant l'exécution d'une application Java dans un navigateur sont obsolètes)

Sécurité Une application Web est accessible depuis tout le domaine autorisé du serveur Web. De ce fait, une authentification serait nécessaire. Contrairement à une application "Desktop" qui pourrait être déployée que sur certains postes. Le but du projet étant de rendre l'application disponible partout, la sécurité se vaut.

Facilité d'implémentation Une application Web permettrait l'usage des composants graphiques d'HTML5 permettant de dessiner des formes et des écouteurs plus facilement qu'en Java FX. Mais encore, grâce à Angular 2 il est beaucoup plus facile d'implémenter un service pour récupérer des données depuis un serveur REST qu'en Java avec des HTTP Client.

Connaissances actuelles Les seules connaissances restantes pour une application Web à approfondir serait l'usage des composants graphiques d'HTML5. Une application Java FX nécessiterait l'étude du fonctionnement d'un Canvas ainsi qu'un approfondissement du fonctionnement des Httpclient.

Multiplateformes Une application Web permettrait son utilisation depuis tous les navigateurs. Elle serait aussi directement utilisable par les téléphones portables. Une application Java FX quant à elle serait disponible nativement sous Windows, Mac OS et Linux, bien que des solutions de portages existent comme JavaFX Port.

Critères	Poids	Java FX	Angular 2/Html5
Réutilisation code	1	1	0
Environnement multi-utilisateur	4	0	3
Sécurité	2	3	2
Facilité d'implémentation	2	3	5
Connaissances actuelles	3	3	2
Total (poids * valeur)	9	22	32

TABLE 2.6 – Comparaison des technologies pour l'application graphique

Détails de la technologie choisie

L'application de monitoring sera une application Web. Elle sera établie en utilisant le framework Angular 2 avec la technologie HTML5. Afin de faciliter l'établissement du design, le framework Bootstrap sera utilisé.

Communication

L'application Web récupérera les informations nécessaires à son fonctionnement depuis le serveur de communication HTTP REST mis en place. Les opérations principales nécessaires à son fonctionnement sont les suivantes :

- Authentification avec un compte administrateur
- Récupération de la liste des résidents et des responsables
- Liste des positions des acteurs ainsi que de leurs status de connexion et d'envoi d'alarmes
- Récupération des logs d'administration
- Ajout de résidents et de responsables

Dessiner des formes pour la carte

Les formes représentant le résidents sont des cercles et les formes représentant les responsables sont des étoiles. Ces dessins sont effectués en utilisant le composant svg. Les trois objets qui sont définis comme importants dans le projet sont définis dans le tableau 2.4.5 suivant. L'extrait de code défini par le listing 2.8 démontre la création de ces formes.

Rect	Un rectangle avec une longueur et une largeur
Circle	Un cercle avec un rayon
Polygon	Un polygone quelconque avec un certain nombre de points

Listing 2.8 – Création de formes en HTML5 dans un svg

```
<svg>
  <rect x="10" y="10" width="50" height="10" />
  <circle x="20" y="20" r="5" />
  <polygon points="0,0  5,5  10,10  15,15  20,20" />
</svg>
```

Leaflet

Leaflet est une librairie permettant d'optimiser le formatage d'une carte avec des boutons d'actions, le zoom et le déplacement par exemple. Il serait possible de préciser l'image du plan du bâtiment à son initialisation, ce qui serait vraiment pratique. La création d'une map se passe en Javascript de la manière suivante :

Listing 2.9 – Création d'une map avec Leaflet (leafletjs.com)

```
var map = L.map('map').setView([51.505, -0.09], 13);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {attribution:
    '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a>
    contributors'}).addTo(map);
L.marker([51.5, -0.09]).addTo(map)
.bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
.openPopup();
```

La figure 2.11 représente un exemple des outils fournis par Leaflet.

FIGURE 2.11 – Exemple de résultat avec Leaflet (leafletjs.com)

2.5 Synthèse

Le serveur d'environnement intelligent sera effectué en Java EE. Il utilisera la librairie uMove permettant la conception de l'environnement intelligent. Le serveur HTTP REST sera contenu dans le serveur d'environnement intelligent. Il utilisera la librairie Spring permettant de faciliter sa conception. L'application graphique permettant le monitoring des entités du système sera une application Web qui sera effectuée en utilisant le framework Angular 2.

2.6 Continuité

Le prochain chapitre traitera de la conception du projet afin d'approfondir les éléments traités dans ce chapitre.

3. Conception

Dans ce chapitre sera traitée la conception des différentes tâches qui sont le serveur métier et l'application Web de monitoring des entités physiques du système.

3.1 Conception du serveur métier

La conception du serveur métier contiendra le diagramme de classes, les détails du serveur de communication HTTP REST et les détails du serveur de l'infrastructure.

3.1.1 Diagramme de classes

Le diagramme de classes représenté par la figure 3.1 a permis de définir les différentes classes principales du système.
Les classes sont séparées dans les paquets suivants :

- Controllers : Contient les contrôleurs REST de l'application Spring. Ces contrôleurs permettent la définition des opérations REST détaillées dans les deux sections suivantes.
- Databases : Contient les éléments liés à la base de données, que sont les dépôts CRUD et les entités de la base de données.
- Model : Contient la partie logique de l'application, soit le Model, le contrôleur FCM ainsi que les appels à uMove.
- Beans : Contient les objets utilisés pour la communication entre les opérations REST et la partie métier.

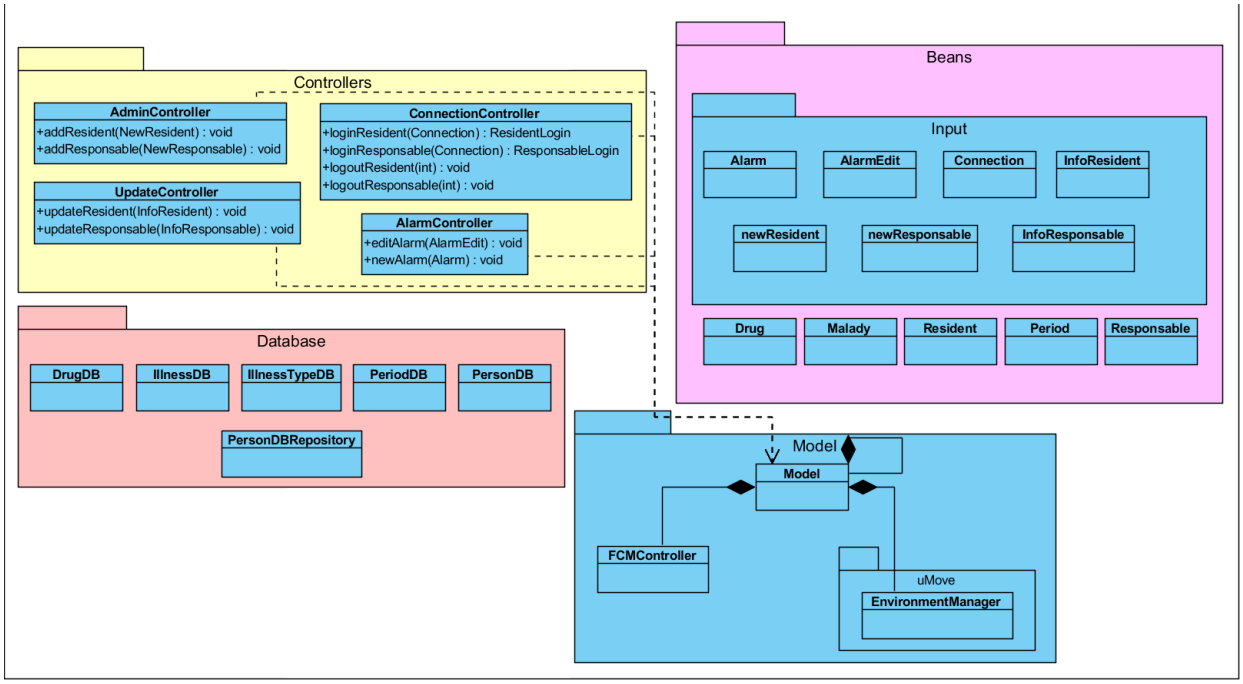

FIGURE 3.1 – Diagramme de classe du serveur métier

3.1.2 Méthodes pour le serveur de communication REST

Afin de permettre les différentes opérations depuis le serveur de communication, plusieurs opérations devront être implémentées dans la partie métier. Ces opérations sont les suivantes :

- Ajout d'un résident ou d'un responsable
- Connexion d'un résident ou d'un responsable
- Mise à jour des résidents et des responsables
- Déconnexion d'un résident ou d'un responsable

Ajout d'un résident ou d'un responsable

L'ajout d'un résident ou d'un responsable consiste à principalement à la création d'une instance d'entité, la complétion de ses attributs, la création de l'objet vers la base de données et à la sauvegarde de celle-ci. Le déroulement est décrit par le pseudo-code 3.1

Listing 3.1 – Pseudo code de l'ajout d'un résident ou d'un responsable

```
actor = createEntity(world)
actor.function = resident or responsable
actor.activity = "unknown"
actor.db = new databaseObject()
fill_fields(actor.db)
actors.save(actor)
```

Connexion d'un résident ou d'un responsable

La connexion d'un résident ou d'un responsable consiste à la recherche de la clé de l'acteur correspondant au couple nom prénom avec le traitement d'erreurs. Cette clé permet de récupérer son acteur. La dernière étape consiste à passer le compte de l'acteur en status "en ligne". Elle est décrite par le pseudo-code 3.2

Listing 3.2 – Pseudo code de la connexion d'un résident ou d'un responsable

```
id = find_account(username, password)
if(password is inconrrect)
  throw new exception("password incorrect")
if(username not found)
  throw new exception("account not found")
actor = get_actor_resident(id)
actor.isOnline()
```

Mise à jour d'un résident ou d'un responsable

La mise à jour d'un résident ou d'un responsable consiste à la recherche de l'acteur représentant le numéro id puis à mettre à jour sa localisation et ses informations. Elle est décrite par le pseudo-code 3.3

Listing 3.3 – Pseudo code de la mise-à-jour d'un résident ou d'un responsable

```
actor = get_actor(id)
if(id not found)
  throw new exception("Accout not logged")
actor.updateLocation(x,y,z)
actor.updateProfile(profile)
```

Déconnexion d'un résident ou d'un responsable

La déconnexion d'un résident ou d'un responsable consiste à la recherche de l'acteur représentant le numéro id puis à rendre hors-ligne cet acteur. Elle est décrite par le pseudo-code 3.4 suivant.

Listing 3.4 – Pseudo code de la mise-à-jour d'un résident ou d'un responsable

```
actor = get_actor(id)
if(id not found)
  throw new exception("Accout not logged")
actor.isOffline()
```

Signalisation d'une alarme

La signalisation d'une alarme consiste à la recherche de l'acteur représentant le numéro id puis à la signalisation d'une alarme qui sera traitée par les observateurs. Elle est décrite par le pseudo-code 3.5 suivante

Listing 3.5 – Pseudo code de la mise-à-jour d'un résident ou d'un responsable

```
actor = get_actor(id)
responsable = actor.sendAlarmToObserver
responsable.sendAlarm(alarm)
```

3.1.3 Base de données

La sous-section de la base de données est séparée en deux parties. Premièrement se trouvera le diagramme EA établi, puis se trouvera le diagramme EER retranscrit.

Diagramme EA

Le diagramme EA représenté par la figure 3.2 a permis de reconnaître l'existence de 6 entités importantes et de 5 relations. Deux relations N-N ont été définies, Illness-Drug et Resident-Illness. Elles demanderont chacune l'établissement d'une table de relation supplémentaire dans le modèle relationnel.

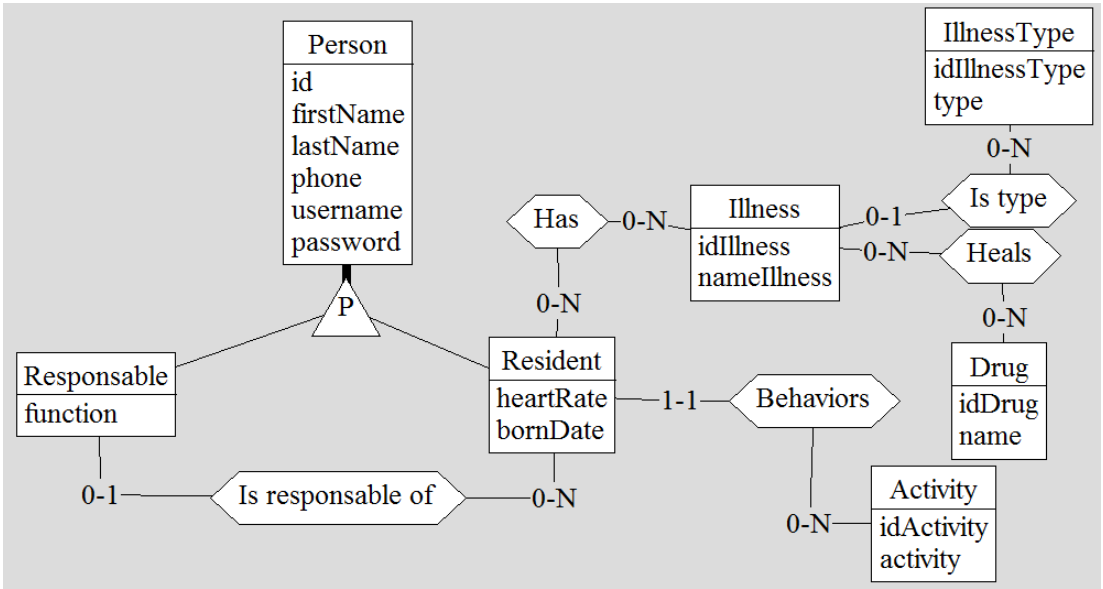

FIGURE 3.2 – Diagramme EA de la base de données

Ces six entités importantes sont décrites dans le tableau 3.1 suivant.

Resident	Informations du résident
Activity	Activité possible du résident
Illness	Maladies du résident
IllnessType	Types de maladie
Drug	Médicaments pour la maladie
Responsable	Information du responsable

TABLE 3.1 – Entités du diagramme EA

Diagramme EER

Le diagramme EER final représenté par la figure 3.3 ne contient dorénavant que 5 entités importantes. En effet, pour des raisons de simplicité, la fusion des entités Resident et Responsable a été envisagée. Les deux relations N-N se sont transformées en une table de relation.

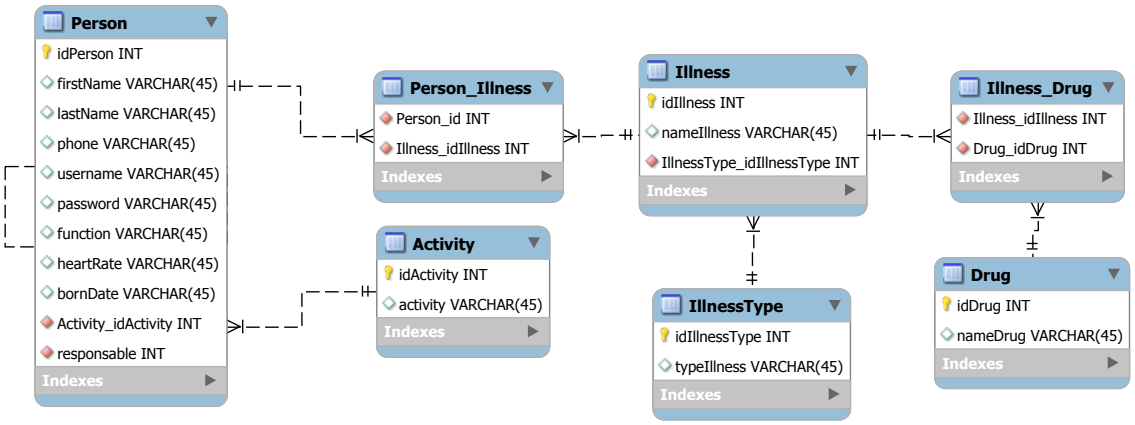

FIGURE 3.3 – Diagramme EER de la base de données

Illness La table Illness contient les maladies et leurs types. Les champs suivants sont définis :

idIllness	Numéro d'identification
nameIllness	Nom de la maladie
IllnessType_idIllnessType	Référence au type de la maladie

TABLE 3.2 – Table des maladies

Drug La table Drug contient les médicaments. Les champs suivants sont définis :

idDrug	Numéro d'identification
nameDrug	Nom du médicament

TABLE 3.3 – Table des médicaments

IllnessType La table IllnessType contient les types des maladies. Les champs suivants sont définis :

idIllnessType	Numéro d'identification
typeIllness	Type de maladie

TABLE 3.4 – Table des types de maladies

Activity La table Activity contient les activités définies. Les champs suivants sont définis :

idActivity	Numéro d'identification
activity	Nom de l'activité

TABLE 3.5 – Table des activités

Person La table Person contient les résidents et les responsables. Les champs suivants sont définis :

idPerson	Numéro d'identification
firstName	Prénom
lastName	Nom de famille
phone	Numéro de téléphone
username	Nom d'utilisateur
password	Mot de passe
function	Fonction du responsable
heartRate	Rythme cardiaque du résident
bornDate	Date de naissance du résident
Activity_idActivity	Activité du résident
responsable	Numéro du responsable du résident

TABLE 3.6 – Table des personnes

3.2 Serveur de communication

La conception du serveur de communication contiendra le détail des méthodes de l'architecture REST.

3.2.1 Objets JSON

Les objets JSON définis sont séparés en deux dossiers. Un dossier nommé "input" contenant les objets réceptionnés dans le corps des opérations du serveur. Le second, nommé "output" contenant les objets qui seront transmis aux clients.

Objets en entrées

Les objets en entrés sont les suivants :

- Alarm
- AlarmEdit
- InfoResident
- InfoResponsable
- Connection

Alarm L'objet Connection contient les informations de connexion d'un résident ou d'un responsable envoyées au serveur REST.

(String) username	Nom d'utilisateur
(String) password	Mot de passe
(String) token	Token d'authentification FCM
(String) phone	Numéro de téléphone

TABLE 3.7 – Information des alarmes

AlarmEdit L'objet Connection contient les informations de connexion d'un résident ou d'un responsable envoyées au serveur REST.

(int) id	Id du responsable
(int) idAlarme	Id de l'alarme (Id du responsable)
(String) response	Réponse booléenne oui/non

TABLE 3.8 – Informations de validation d'alarmes

InfoResident L'objet InfoResident contient les informations d'update d'un résident envoyé au serveur REST.

id	Numéro d'identification
heartRate	Rythme cardiaque
activity	Activité
longitude	Longitude
latitude	Latitude

TABLE 3.9 – Information de mise-à-jour d'un résident

InfoResponsable L'objet InfoResponsable contient les informations d'update d'un responsable envoyées au serveur REST.

id	Numéro d'identification
longitude	Longitude
latitude	Latitude
disponibility	Disponibilité (true/false)
phone	Numéro de téléphone

TABLE 3.10 – Informations de mise à jour d'un responsable

Connection L'objet Connection contient les informations de connexion d'un résident ou d'un responsable envoyées au serveur REST.

username	Nom d'utilisateur
password	Mot de passe
token	Token d'authentification FCM
phone	Numéro de téléphone

TABLE 3.11 – Informations de connexion

3.2.2 Méthodes de l'architecture REST

Les opérations de l'architecture REST des résidents et des responsables ont été pour la plupart séparées. Cela facilitera les possibilités d'extension par la suite.

Sommaire des opérations

Les détails des méthodes de l'architecture REST sont séparés en une partie résident et une partie responsable.

Opérations des responsables Les opérations des responsables sont listés dans le tableau 3.12.

[POST] login/responsable	Login des responsables
[POST] update/responsable	Informations des responsables
[POST] logout/responsable	Déconnexion d'un responsable
[PUT] alarm/take	Acceptation d'une alarme
[PUT] alarm/validate	Validation d'une alarme
[GET] residents	Récupération des résidents
[GET] resident/id	Récupération d'un résident
[POST] message/responsable	Envoi de messages sur un canal
[POST] message/responsable/new	Création d'une question pour un résident
[POST] resident/(id)/drugs	Ajout de médicaments au résident
[PUT] resident/(id)/drugs	Modification des périodes d'un médicament
[DELETE] resident/(id)/drugs	Suppression d'un médicament
[GET] drugs	Récupération des médicaments disponibles

TABLE 3.12 – Analyse des opérations REST pour les responsables

Opérations pour les administrateurs Les opérations des administrateurs sont listés dans le tableau 3.13.

[POST] users/new	Ajout d'un utilisateur
[GET] users	Récupération des utilisateurs

TABLE 3.13 – Opérations pour les administrateurs

Les opérations des responsables sont définies dans le tableau suivant :

[POST] login/resident	Login des résidents
[POST] update/resident	Informations des résidents
[POST] logout/resident	Déconnexion d'un résident
[POST] alarm	Envoi d'alarme
[POST] message/resident/new	Création d'un canal de message
[POST] message/resident	Envoi de messages sur un canal

TABLE 3.14 – Analyse des opérations REST pour les résidents

Connexion d'un résident ou d'un responsable

Deux méthodes ont été créées pour le login, une pour les résidents et une pour les responsables. Elles utilisent le verbe POST avec la même structure détaillée dans le listing 3.6.
[POST] login/resident : Login des résidents
[POST] login/responsable : Login des responsables

Listing 3.6 – Structure d'envoi à la connexion

```
{
  "username": "",
  "password": "",
  "token": "", // FCM token to send push notifications
  "phone": ""  // Phone number
}
```

Des informations sur le profil des utilisateurs sont retournées au login. Elles contiennent les informations utilisées par les applications mobiles concernées.

Réception résident	**Réception responsable**
`{ "id": 1, "firstname": "", "lastname": "", "bornDate": 1, "illness": [{"name": "", "type": ""}], "drugs": [{"name": "", "periods": [{"hour": 1, "minute": 1}]}] }`	`{ "id": 1, "firstname": "", "lastname": "", "function": "" }`

TABLE 3.15 – Communication JSON pour la connexion

Informations

Deux opérations permettent aux résidents et au corps médical d'envoyer leurs informations servant d'acquittement. Le champ "disponibility" est un number, ce qui permettrait de traiter les résidents occupés mais qui pourraient être dérangés par une alarme urgente (par exemple s'ils boivent un café). Ces deux méthodes sont les suivantes :
[POST] update/resident : Informations des résidents
[POST] update/responsable : Informations des responsables

Informations des résidents	**Informations des responsables**
`{ "id": 1, "heartRate": 1, "activity": "", "longitude": 1, "latitude": 1 }`	`{ "id": 1, "longitude": 1, "latitude": 1, "disponibility": 1, // 1: disponible "phone": "" }`

TABLE 3.16 – Communication JSON pour l'envoi d'informations au serveur

Alarmes

Les méthodes des alarmes permettent d'informer le serveur d'une alarme ou d'en valider. L'ajout d'alarme se passe par le verbe POST et la modification par le verbe PUT.
Le JSON envoyé pour la prise et la validation d'une alarme utilise la même structure. Ces trois méthodes sont les suivantes :
[POST] alarm : Envoi d'alarme
[PUT] alarm/take : Acceptation d'une alarme
[PUT] alarm/validate : Validation d'une alarme

Déclenchement alarme	**Modification alarme**
`{ "id": 1, "type": "", "value": "", "longitude": 1, "latitude": 1, "phone": "" }`	`{ "id": 1, // id responsable "idAlarm": 1, // id resident "response": true }`

TABLE 3.17 – Communication JSON pour le déclenchement et la validation d'alarmes

Lors de la réception d'une alarme, les informations suivantes sont envoyées en FCM :

Information d'une alarme	**Fin de l'alarme**
{ "data": { "id": 1, "type": "", "value": "" } }	{ "data": { "id": 1, "status": true //finished } }

TABLE 3.18 – Envoi des informations des alarmes par FCM

Récupération des résidents

La route /resident retourne tous les résidents dont le responsable correspondant à l'id est responsable.
[GET] /residents/idResponsable : Récupération des résidents d'un responsable
[GET] /resident/idResident : Récupération du résident ayant l'id ou de tous les résidents

Les résidents responsables	**Recherche d'un résident**
{ "residents": [{ "id": 1, "firstname": "", "lastname": "", "phone": "", "bornDate": 1, "longitude": 1, "latitude": 1, "heartRate": 1, "activity": "", "location": "", "illness": [{"name": "", "type": ""}], "drugs": [{"name": "", "periods": [{ "hour": 1, "minute": 1}]}] }] }	{ "firstname": "" "lastname": "", "phone": "", "bornDate": 1, "longitude": "", "latitude": "", "heartRate": 1, "activity": "", "location": "", "illness": [{"name": "", "type": ""}], "drugs": [{"name": "", "periods": [{ "hour": 1, "minute": 1}}] }] }

TABLE 3.19 – Récupération des résidents

Déconnexion

La route /logout permet la déconnexion d'un utilisateur **[POST] /logout** : Déconnexion
Les paramètres sont définis selon la structure simple "x-www-form-urlencoded", comme il n'est pas nécessaire d'utiliser le json.

```
"id": 0
```

Envoi de message

L'envoi de messages est effectué depuis le serveur en utilisant le système de notification Push FCM. Pour faciliter l'implémentation des applications mobiles des résidents et des responsables, des opérations REST ont été effectués afin que la demande d'envoi passe par le serveur.
Pour des raisons de simplicité, le même JSON est utilisé pour la création du canal et l'envoi de messages. Le champ "idResponsable" n'est pas utilisé à la création du canal.
Le JSON défini est le suivant :

Listing 3.7 – Création du canal ou envoi d'un message

```
{
  "idResident": 1, // id of the resident
  "idResponsable": 1, // id of the responsable, or a dummy number
  "message": ""
}
```

Création du canal de diffusion La création du canal de diffusion est effectuée une seule fois par le résident qui commence une discussion, sans connaître le responsable. L'opération est la suivante :
[POST] /message/resident/new : Création d'un canal de message par les résidents

Le JSON utilisé est indiqué dans le listing 3.7

Création d'une question La création d'une question par un responsable contient directement le numéro du résident, la question ainsi que des réponses possibles. L'opération est la suivante :
[POST] /message/responsable/new
Les JSONs utilisés sont indiqués dans les extraits 3.7.

Demande d'une question	**Réception d'une question par FCM**

`{`
`  "idResident": 1,`
`  "idResponsable": 1,`
`  "responses": [`
`  {"response": ""}, ...]`
`  "question": "" //question`
`}`
	`{`
	`  "data": {`
	`  "idResident": 1,`
	`  "idResponsable": 1,`
	`  "responses": [`
	`  {"response": ""}, ...]`
	`  "question": "" // question`
	`  }`
	`}`

Table 3.20 – Création d'une question par les responsables

Envoi de nouveaux messages dans le canal de diffusion L'envoi d'un message dans un canal de diffusion est effectué grâce aux opérations suivantes :
[POST] /message/resident : Envoi d'un message dans un canal par les résidents
[POST] /message/responsable : Envoi d'un message dans un canal par les responsables

Le JSON utilisé est indiqué dans le listing 3.7

Réponses FCM La structure JSON indiqué en 3.8 sera envoyée aux applications mobiles à chaque message reçu.

Listing 3.8 – Réception d'un message par FCM

```
{
  "data": {
        "idResident": 1,
        "idResponsable": 1,
        "message": ""
  }
}
```

Édition des médicaments d'un résident

L'édition des médicaments utilise quatre opérations qui sont les suivantes :
[GET] /drugs : Récupération de tous les médicaments disponibles)
[POST] /resident/(id)/drugs : Ajout d'un médicament
[PUT] /resident/(id)/drugs : Modification des périodes d'un médicament
[DELETE] /resident/(id)/drugs : Suppression d'un médicament

Récupération des médicaments disponibles La récupération des médicaments retourne le JSON suivant :

```
[{
  "idDrug": 1
  "name": "",
}, ... ]
```

Ajout d'un médicament L'ajout d'un médicament consiste à l'envoi du JSON suivant :

```
{
  "id": 1,  // id responsable
  "idDrug": 1,
  "periods": [{"hour": 1, "minute": 1}, ...]
}
```

Modification d'un médicament La modification d'un médicament consiste à l'envoi du JSON suivant :
Pour faciliter les opérations, le responsable modifie seulement le médicament nécessaire.

```
{
  "id": 1,  // id responsable
  "idDrug": 1,
  "periods": [{"hour": 1, "minute": 1}, ...]
}
```

Suppression d'un médicament La suppression d'un médicament consiste à l'envoi du JSON suivant :
Pour faciliter les opérations, le responsable précise seulement le médicament nécessaire au lieu de renvoyer toute la liste.

```
{
  "id": 1,  // id responsable pour les permissions
  "idDrug": 1
}
```

Ajout d'un résident

L'ajout d'un résident se passe par l'opération /users/resident/new.

```
{
  "firstName": "",
  "lastName": "",
  "bornDate": "",
  "username": "",
  "password": "",
  "illness": [{"name": "", "type": ""}],
  "drugs": [{"name", "periods": [{"hour": 1, "minute": 1}]}]
  "idResponsable": 1
}
```

Ajout d'un responsable

L'ajout d'un résident se passe par l'opération /users/responsable/new.

```
{
  "firstName": "",
  "lastName": "",
  "bornDate": 123,
  "username": "",
  "password": "",
}
```

3.3 Traitement de la concurrence, des alarmes et des messages

Le traitement des alarmes et des messages est effectué en utilisant un moniteur. Toutes les actions modifiant l'état de la concurrence y passent.

3.3.1 Définition des conditions

Les conditions suivantes seraient utilisées dans le moniteur.

- responseResident : Cette condition permet de bloquer la déconnexion d'un résident. Elle est utilisée quand un résident doit répondre à une question d'un responsable.
- responseResponsable : Cette condition permet de bloquer la déconnexion d'un responsable. Elle est utilisée quand un responsable doit choisir de prendre une alarme ou de la valider.
- availableResponsable : Cette condition permet de traiter les attentes de recherche d'un responsable. Elle est utilisée pour les résidents qui recherchent un responsable, mais qui doivent attendre car les responsables sont occupés ou que la demande n'est pas prioritaire.
- takeAlarm : Cette condition est utilisée par les alarmes qui attendent que l'alarme soit acceptée par le responsable auquel elle a été envoyée. Le responsable a 90 secondes pour l'accepter, sinon une nouvelle personne sera recherchée.
- ValidateAlarm : Cette condition est utilisée par les alarmes qui attendent d'être validée.

3.3.2 Conception des opérations

Les opérations définis sont les suivantes :

Mise à jour d'un résident

La mise à jour d'un résident consiste à mettre à jour sa position, son activité et son rhytme cardiaque. Le timer de 30 secondes traitant le timeout est remis à 0 à chaque update.

```
actor = residents.find(id)
actor.updateLocation(location)
actor.updateActivity(activity)
actor.updateHeartRate(heartRate)

actor.resetUpdateTimer()
```

Mise-à-jour d'un responsable

La mise à jour d'un résident consiste à mettre à jour sa position, son activité et son rhytme cardiaque. Le timer de 30 secondes traitant le timeout est remis à 0 à chaque update.

```
actor = responsable.find(id)
actor.updateLocation(location)
actor.updateDisponibility(disponibility)

actor.resetUpdateTimer()
```

Déconnexion d'un résident

La déconnexion d'un résident désactivera les alarmes qui ne sont pas encore traitées, les alarmes qui sont en cours de traitement devront attendre la réponse des responsables.

```
actor = residents.find(id)
alarms = actor.getAlarms()
alarms.clearNew()
actor.startDisconnection()
while(!actor.canDisconnect())
  responseResident.wait()
```

Déconnexion d'un responsable

La déconnexion d'un responsable attend que toutes les alarmes reçues par le responsable soient validées.

```
actor = responsables.find(id)
actor.startDisconnection()
while(!actor.canDisconnect())
  responseResponsable.wait()
```

Timeout résident

Un timeout d'un résident qui a lieu lorsque son status n'a pas été mis à jour durant les 30 dernières secondes envoie une alarme.

```
actor = residents.find(id)
alarm = createAlarm("TimeOut", "30")
actor.sendAlarm(alarm) // Call the alarm operation, explained later
```

Timeout responsable

Un timeout d'un responsable qui a lieu lorsque son status n'a pas été mis-à-jour durant les 30 dernières secondes le rend indisponible. Toutes les alarmes qui lui étaient affectées sont basculées à un autre résident en relançant l'algorithme de recherche.

```
actor = responsables.find(id)
actor.disponibility = -1
for( alarm a in actor.alarms)
  a.changeResponsable()
```

Envoi d'un message sans connaître le destinataire

Le résident peut envoyer un message sans connaître le destinataire. À ce moment, un algorithme va rechercher un responsable disponible qui est son responsable direct, ou le plus proche s'il n'est pas disponible et lui envoyer le message.

```
actor = residents.find(id)

responsable = actor.getResponsable()
if(responsable.disponibility < 0)
  // responsable direct unavailable
  responsable = findResponsable()
  while (no responsable found)
    waitResponsable.await()
    responsable = findResponsable()

actor.sendMessage(message, responsable.Token)
```

Envoi d'un message en connaissant le destinataire

Le résident ou le responsable va pouvoir directement envoyer un message au destinataire. Si cet utilisateur est en train de se connecter, elle sera bloquée dans sa déconnexion.

```
actor = residents.find(id)
actor.sendMessage(message, destination)
```

Envoi d'une question à un résident

Le responsable va pouvoir envoyer une question à un résident. L'envoi d'une question va bloquer la déconnexion jusqu'à ce qu'elle soit répondue.

Ajout d'alarme

La conception de l'algorithme de traitement des alarmes contiendra un diagramme d'état suivi d'un.

Diagramme d'états Six états ont été défini dans le diagramme d'état de la figure 3.4. Ils correspondent à la description suivante :

1. Start : Etat de démarrage où la priorité de l'alarme est calculée. Si une alarme existe déjà pour ce résident, la priorité sera additionnée.
2. Work : Etat de processing. Si des opérations sont prioritaires, l'alarme sera déplacée dans la file d'attente. Si l'alarme est prête, elle sera envoyée au responsable choisi.
3. Wait in queue : Attente dans la file. Si l'alarme est choisie, elle sera redirigée dans l'état de processing.
4. Sent : Après qu'une alarme soit prête, elle est envoyée au responsable défini. Depuis cet état, elle peut être soit refusée ou passée en timeout, ce qui la redirigera à l'état de processing. Si ce n'est pas le cas, elle peut aussi être confirmée par le responsable ce qui la passera à l'état done.
5. Validating : Après que le responsable a accepté l'alarme, il doit soit valider la résolution de l'alarme ce qui la conclura, soit la refuser ce qui recherchera une nouvelle personne.
6. Done : Après qu'une alarme soit terminée, elle sera retirée du système. Une entrée de log sera ajoutée dans la base de données afin de garder un historique de l'alarme et des personnels impliqués.

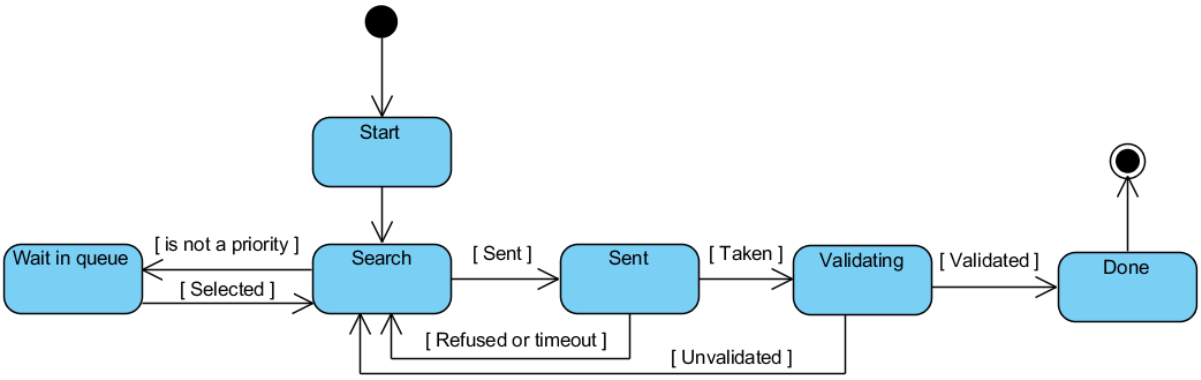

FIGURE 3.4 – Diagramme d'état sur les alarmes

Pseudo-code de l'envoi d'alarme Dans le pseudo-code 3.9 est démontré le fonctionnement de l'opération d'envoi d'alarmes. Deux conditions ont été définies, une lors de l'attente de trouver un responsable et une pour l'attente d'une réponse.

Listing 3.9 – Pseudo-code de l'opération du moniteur d'envoi d'alarmes

```
// Start state
computePriority(alarm)

while(alarm is not done)
  while (more urgent alarms exists)
    // Wait in queue State
    waitResponsable.await()

  // Work State
  responsable = findResponsable()
  while (no responsable found)
    waitResponsable.await()
    responsable = findResponsable()

  // Sent state
  sendAlarmToResponsable(alarm, responsable)
  maxDate = 90000  //Max timeout of 90s
  while(no response received)
    waitResponse.await(maxDate)

  // Validating state
  while( no response received )
    waitValidating.await( )

// Done state
responsable.disponibility = 1
waitResponsable.signal()
```

Acceptation d'une alarme

Le responsable ayant reçu une alarme doit l'accepter. S'il l'accepte, le thread d'envoi d'alarme va attendre l'acceptation, s'il refuse sa disponibilité va passer en bas de la liste et l'algorithme va recherche un autre responsable. Le pseudo-code effectué est détaillé dans l'extrait 3.10.

Listing 3.10 – Pseudo-code de l'opération du moniteur de l'acceptation d'une alarme

```
alarm = findAlarm()
alarm.take = true
waitResponse.signal()
```

Validation d'une alarme

Le responsable ayant accepté une alarme doit la valider. S'il la valide, le thread d'envoi d'alarme va se terminer, s'il la invalide, l'algorithme va rechercher un autre responsable. Le pseudo-code effectué est détaillé dans l'extrait 3.11.

Listing 3.11 – Pseudo-code de l'opération du moniteur de validation d'alarme

```
alarm = findAlarm()
alarm.vaidate = true
waitResponse.signal()
```

3.4 Application de monitoring

Au niveau de la conception de l'application de monitoring seront détaillées les maquettes effectuées.

3.4.1 Maquettes

La maquette principale de l'application consiste à la représentation des différents rôles du système suivant leur position dans une carte de la résidence. Il sera possible de filtrer les types d'éléments affichés (résidents, corps médical et lieu). Il sera aussi possible de consulter sous forme de liste les entités statiques, comme ce qui est dessiné sur la gauche. En cliquant sur un lieu, la zone correspondante deviendra colorée.
Afin de faciliter l'ergonomie de l'application de monitoring, il a été pensé de proposer un champ de recherche permettant directement d'effectuer une recherche d'un résident ou d'un lieu.

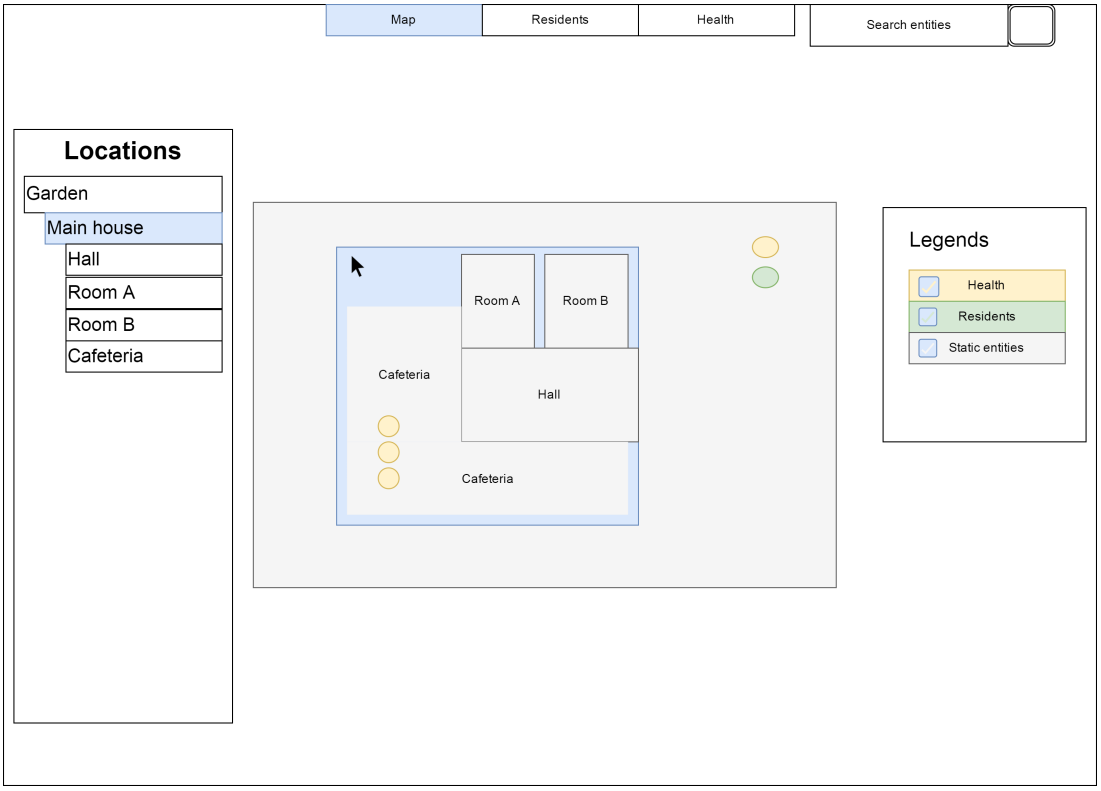

FIGURE 3.5 – Maquette de l'application de monitoring

Il existera aussi une vue permettant d'afficher la liste des résidents et la liste du corps médical. Cette liste contiendra aussi toutes les informations essentielles de la personne.

3.5 Diagramme de séquences

Le diagramme de séquence défini dans la figure 3.6 a permis de définir le fonctionnement de la mise à jour des informations dans l'application de monitoring.

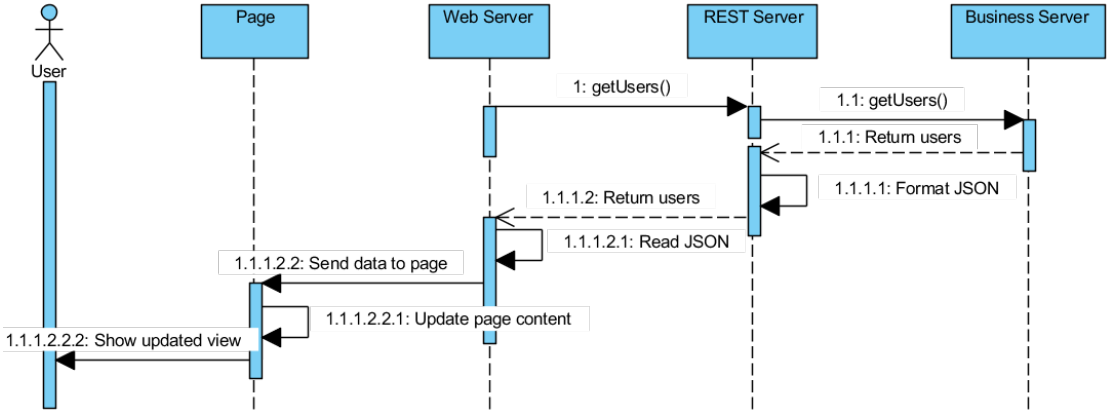

FIGURE 3.6 – Diagramme de séquence du rafraîchissement de l'application de monitoring

3.6 Résumé

La conception a permis la création d'un diagramme de classes de l'infrastructure serveur, la conception des opérations du serveur REST avec la structure des données, la conception de la base de données ainsi que de la conception de l'algorithme et de l'application de monitoring.

3.7 Continuité

Le prochain chapitre traitera de l'implémentation du projet.

4. Implémentation

Dans ce chapitre est traitée l'implémentation des différentes tâches qui sont l'infrastructure de l'environnement intelligent, le serveur de communication, l'algorithme de traitement des alarmes et l'application graphique de monitoring des entités.

Sommaire

4.1 Infrastructure de l'environnement intelligent

Le serveur représentant l'environnement intelligent a été implémentée selon les détails effectués dans la conception. Des difficultés au niveau de la localisation ont été trouvées : les téléphones testés par les deux autres projets qui sont RMMA et HMMA n'ont pas la même précision. De ce fait, une marge de différence a été ajoutée pour chacune des entités du bâtiment. Suite à cette marge, les deux applications se retrouvent en grande partie localisées dans le bâtiment, malgré une marge d'erreur d'environ une salle dans certains cas.

Il a aussi été repéré que les salles plus proches de la tour de Télécommunications sont bien mieux localisées depuis les téléphones portables, ceci dû à la non-présence de bâtiments adjacents.

Pour faciliter l'implémentation, seule la partie des entités de la librairie uMove a été utilisée. En effet, en utilisant le serveur de communication HTTP REST, les entités n'effectue pas d'actions d'eux-mêmes, ce qui est la raison pour laquelle les couches Senseurs et Sendget n'ont pas été utilisées. Le traitement des actions comme les alarmes ou l'envoi de messages passent par un moniteur. De ce fait, la couche des observateurs n'est pas non plus utilisée.

4.2 Serveur de communication

Le serveur de communication a été facilement mis-en-place par la librairie Spring qui a facilité son développement. Grâce à cette librairie, il est maintenant possible d'effectuer un .jar permettant de lancer le serveur sur n'importe quelle machine équipée de Java. Spring ne demandant pas de serveur d'hébergement, comme Glassfish ou Tomcat, il n'y aura pas de prérequis d'installation.

4.2.1 Opérations

Les opérations REST ont été implémentées comme définies dans la conception. Plusieurs changements ont été effectuées par rapport aux propositions initiales. Tout d'abord, certains champs

contiennent en suffixe le nom de l'objet, ce qui évite les problèmes de parsing du JSON, "nameIllness" par exemple.

Dans les cas où les demandes ne correspondent pas aux informations attendues, des exceptions sont jetées. C'est par exemple le cas lorsqu'un client souhaite établir une connexion avec un mot de passe invalide ou un compte qui n'existe pas.
Dans le chapitre suivant concernant les tests se trouveront les JSONs envoyés et reçus effectués.

4.3 Algorithme de traitement des alarmes et des messages

L'algorithme de traitement des alarmes et des messages a été implémentée comme défini dans la conception. Un moniteur a été utilisé pour faciliter le traitement de la concurrence au lieu d'utiliser les observateurs de la librairie uMove.

Le manque d'information et de temps n'a malheureusement pas permis d'effectuer un "vrai" système de calcul de priorités. Pour le moment, l'algorithme défini que les alarmes de rhytmes cardiaques sont prioritaires par rapport aux chutes, qui sont eux-aussi prioritaires par rapport aux alertes de sorties du lieu de vie, ainsi qu'à ceux des activités.

En résumé, le résultat est le suivant : `activité < sortie < chute < rhytme cardiaque`.

4.4 Application Web de monitoring des entités

L'application graphique de monitoring des entités a été effectuée en tant qu'application Web. Le framework Angular 2 a été utilisée. La librairie Leaflet permettant une meilleure gestion de la carte du monde n'a malheureusement pas pu être utilisée dans le temps imparti ; le monde est représenté dans un SVG en utilisant des rectangles, cercles et autres polygones.

Les choix des formes ont été définis de manière à ne pas obliger les utilisateurs à consulter la couleur. Cela permet aux daltoniens de pouvoir utiliser entièrement l'application.

4.4.1 Connexion d'un administrateur

La première étape d'un administrateur, démontrée par la figure 4.1 après lancement d'une application Web consiste à la connexion d'un administrateur. Dans le cas où la connexion est validée, l'utilisateur sera redirigé vers l'application Web principale.

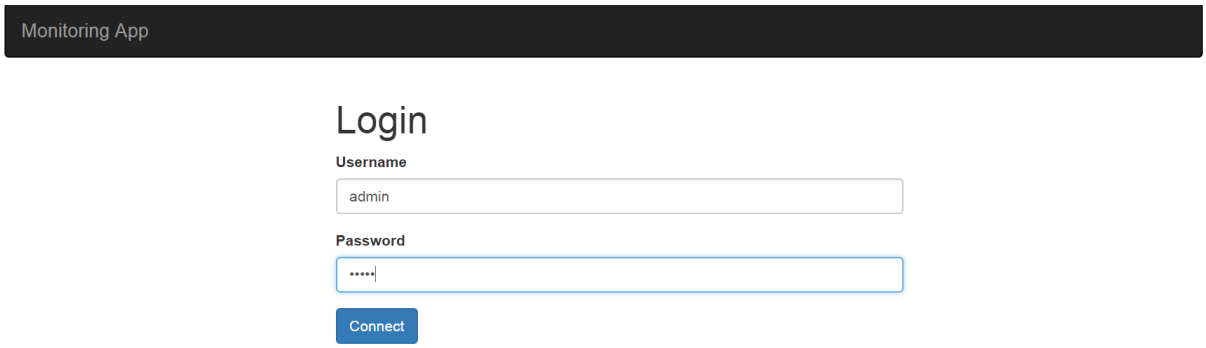

FIGURE 4.1 – Connexion d'un administrateur

4.4.2 Le menu de l'application

Le menu affichée dans la figure 4.2 visible après connexion contient les options suivantes :

- Home : Permet la redirection sur la page d'accueil
- Residents : Permet la redirection sur la page des résidents
- Staff : Permet la redirection sur la page des responsables
- Recherche : Permet l'exécution d'une recherche (page d'accueil uniquement) d'un acteur se trouvant sur la carte.
- Logout : Permet la déconnexion de l'application Web.

FIGURE 4.2 – Menu de l'application

4.4.3 Page d'accueil

La page d'accueil visible dans la figure 4.3 permet la consultation de la carte comme la base a été définie dans la conception. Deux différences sont à noter : Premièrement, les formes de la légende ont été modifiées : une étoile est utilisée pour la représentation des responsables et un point d'exclamation est utilisé pour le signalement d'un résident en état d'alarme.

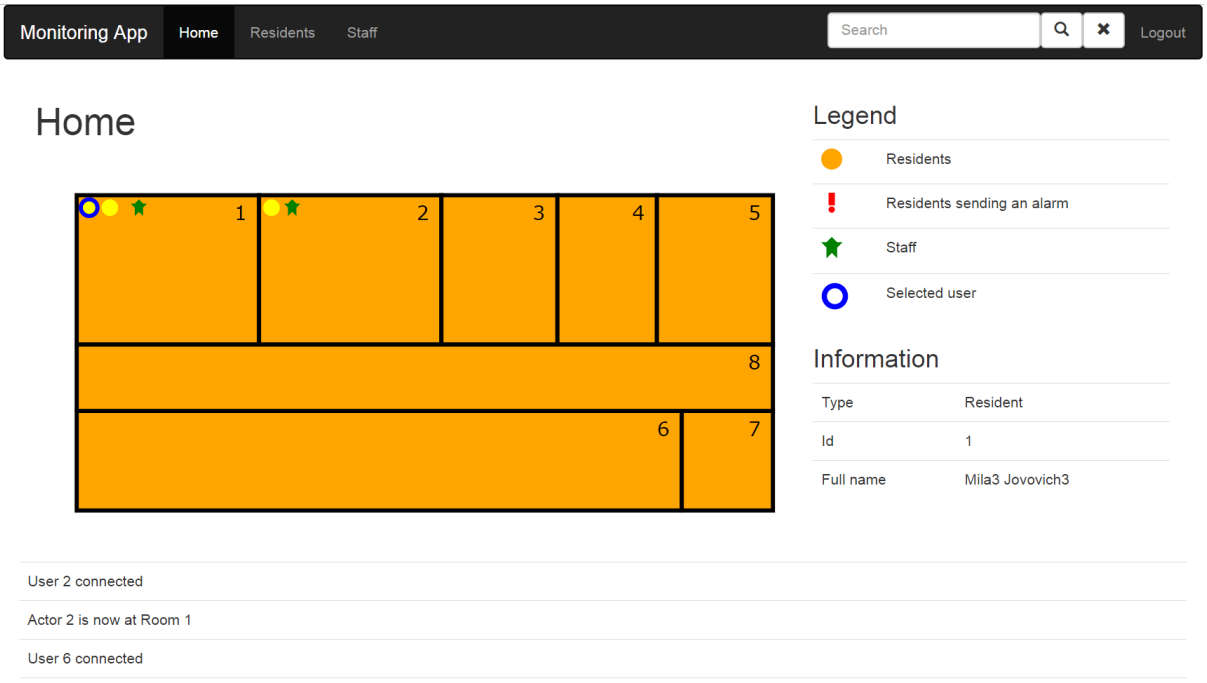

FIGURE 4.3 – Page d'acuceil de l'application

Sur le bas de la page se trouvent aussi les dix derniers messages de logs enregistrés sur le serveur.

4.4.4 Page des résidents

La page des résidents permet de consulter la liste des ceux-ci et d'en ajouter. La liste des résidents comprend les informations de l'id, du prénom, du nom et de l'activité de la personne. Un bouton Show details permet de consulter le profil de la personne, comme le démontre la figure 4.4. L'ajout des résidents est représenté par la figure 4.5. Elle contient un formulaire contenant les informations nécessaires à la création d'un résident.

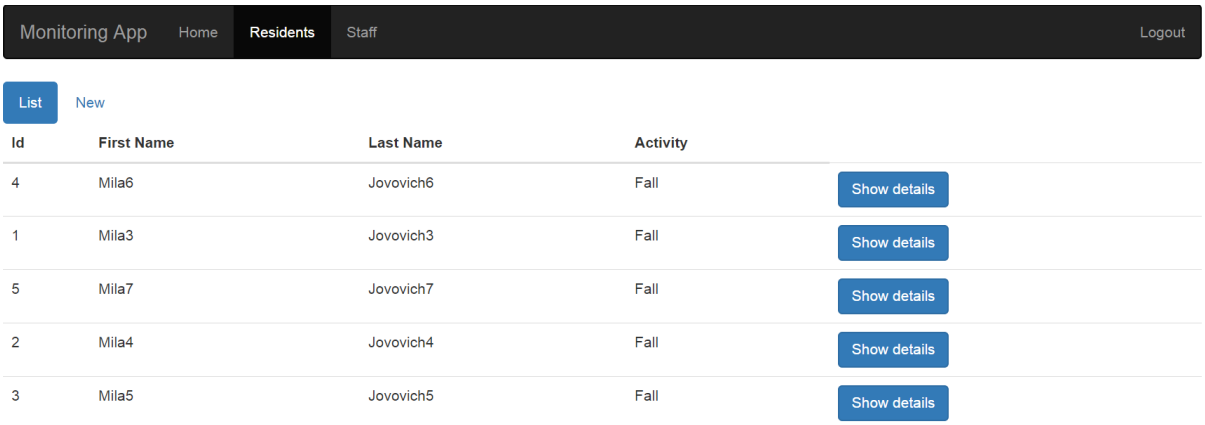

FIGURE 4.4 – Liste des résidents

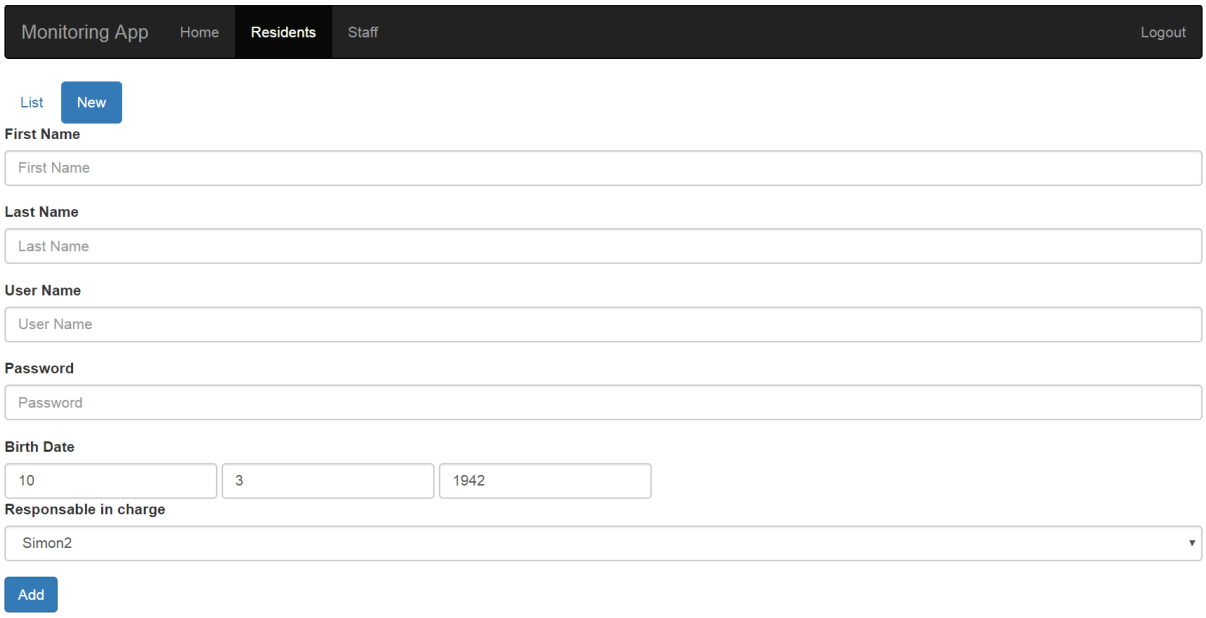

FIGURE 4.5 – Ajout d'un résident

4.4.5 Page des responsables

La page des responsables permet de consulter la liste de ceux-ci et d'en ajouter. La liste des responsables comprend leur id, leur prénom, leur nom et leur disponibilité. Un bouton Show details permet de consulter le profil de la personne, comme démontrée par la figure 4.6. L'ajout des responsables est représenté par la figure 4.7. Elle comprend un formulaire contenant les informations nécessaires à la création d'un responsable.

FIGURE 4.6 – Liste des responsables

FIGURE 4.7 – Ajout d'un responsable

4.5 Synthèse

L'implémentation du projet comprend les informations de mise en place des différents composants nécessaires au projet. Quelques différences se trouvent par rapport aux informations définies dans l'analyse et la conception.

4.6 Continuité

Dans le chapitre suivant seront détaillés les tests effectués pour chacun des composants du système, que sont l'infrastructure serveur et le serveur de communication, l'algorithme de traitement des alarmes ainsi que de l'application graphique de monitoring des entités.

5. Tests

Ce chapitre contiendra les tests effectués sur le serveur de communication et ses opérations utilisant l'infrastructure uMove, sur l'algorithme de traitement des alarmes et sur l'application graphique de monitoring des entités physiques du système.

Sommaire

5.1 Tests fonctionnels

Les tests fonctionnels effectués sont séparés pour les opérations du serveur de communication et les scénarios globaux ainsi que ceux de l'application Web de monitoring.

5.1.1 Tests des opérations du serveur de communication

Un test a été effectué pour chacune des opérations REST effectuées.

Connexion

Les tests de connexion ont été séparés pour chacun des types d'acteurs du système, que sont les résidents et les responsables.

Connexion d'un responsable La connexion du responsable s'appelle en appelant l'opération suivante. **localhost :8080/login/responsable** Un exemple de JSON envoyé est le suivant :

```
{
  "username": "1",
  "password": "1",
  "token": "058190xjjsd",
  "phone": "025 543 24 32"
}
```

Ce json retournera le résultat suivant :

```
{
  "id": 6,
  "firstname": "Simon1",
  "lastname": "Belmont1",
  "function": "doctor"
}
```

Connexion d'un résident La connexion du résident s'appelle en appelant l'opération suivante. **localhost :8080/login/resident**
Un exemple de JSON envoyé est le suivant :


```
{
  "username": "3",
  "password": "3",
  "token": "058190xjjsd",
  "phone": "025 543 24 32"
}
```


Ce JSON retournera le résultat suivant :


```
{
  "id": 2,
  "firstname": "Mila1",
  "lastname": "Jovovich1",
  "bornDate": 1497872126000,
  "illness": [
  {
    "nameIllness": "Alzheimer0",
    "type": "Cerebrales 0",
    "drugs": [{"name": "Aspegic 0",
      [{"hour": 10, "minute":20},{"hour":14, "minute":30}]}]
  }]
}
```


Cas d'exceptions Les cas suivants sont bien traités, pour les résidents et pour les responsables :

Description du cas	**HTTP Code**	OK/nOK
Nom d'utilisateur incorrect	Erreur 404	OK
Mot de passe incorrect	Erreur 404	nOK 412.
Utilisateur déjà connecté	Erreur 400	OK
Sans erreur	HTTP 200 OK	OK

TABLE 5.1 – Traitements des cas pour le login

Envoi d'alarme

L'opération d'envoi d'alarme est fonctionnelle. La logique métier de l'observateur n'est pas encore implémenté.

```
{
  "id": 2,
  "type": "Fall",
  "value": 60,
  "longitude": 30,
  "latitude": 30,
  "phone": "025 543 24 32"
}
```

Les exceptions suivantes sont traitées :

Description du cas	**HTTP Code**	OK/nOK
Id incorrect	Erreur 404	OK
Utilisateur non connecté	Erreur 404	OK
Sans erreur	HTTP 200 OK	OK

TABLE 5.2 – Traitements des cas pour l'envoi d'alarme

Validation d'une alarme

L'opération pour la validation d'une alarme a été implémentée. Les tests suivants y ont été effectués :

```
{
  "id": 4,
  "idAlarme": 2,
  "response": true
}
```

Les exceptions suivantes sont traitées :

Description du cas	**HTTP Code**	OK/nOK
Id incorrect	Erreur 404	OK
Utilisateur non-connecté	Erreur 404	OK
Alarme non existante	pas implémentée	nOK
Sans erreur	HTTP 200 OK	OK

TABLE 5.3 – Traitements des cas pour l'envoi d'alarme

Déconnexion d'un résident et d'un responsable

L'opération pour la déconnexion d'un résident a été implémentée. Les tests suivants ont été effectués :
Paramètre : id = 2
Les exceptions suivantes sont traitées :

Description du cas	**HTTP Code**	OK/nOK
Id incorrect	Erreur 404	OK
Utilisateur non-connecté	Erreur 404	OK
Sans erreur	HTTP 200 OK	OK

TABLE 5.4 – Traitements des cas pour la déconnexion d'un résident

Mise-à-jour

L'opération de mise à jour est aussi séparée pour chacun des deux acteurs du système.

Mise à jour d'un responsable L'opération de mise à jour d'un responsable a pu être validée.
L'appel à cette opération est faite avec un POST vers l'adresse : /update/responsable
Le JSON envoyé est le suivant :

```
{
        "id": 6,
        "longitude": 30,
        "latitude": 30,
        "disponibility": 0,
        "phone": "82520492"
}
```

Mise-à-jour d'un résident L'opération de mise à jour d'un résident a pu être validée.
L'appel à cette opération est faite avec un POST vers l'adresse : /update/resident
Le JSON envoyé est le suivant :

```
{
        "id": 5,
        "heartRate": 100,
        "activity": "Normal",
        "longitude": 30,
        "latitude": 30
}
```

Validation des erreurs

Les exceptions suivantes sont traitées, pour les résidents et pour les responsables

Description du cas	**HTTP Code**	OK/nOK
Id incorrect	Erreur 404	OK
Utilisateur non-connecté	Erreur 404	OK
Sans erreur	HTTP 200 OK	OK

TABLE 5.5 – Traitements des cas pour la mise à jour d'un résident et d'un responsable

5.1.2 Scénario de tests du serveur

Afin de tester plus en détails le serveur métier, des scénarios de tests ont été effectués pour chacune des opérations importantes.

Tests d'envoi de messages

Les tests d'envoi de messages ont pu déterminer que l'envoi de messages fonctionnait globalement, mis à part pour le même cas à problèmes définis dans les tests d'envoi d'alarme qui est lorsqu'un résident envoie une alarme lorsqu'il n'y a pas de responsables connectés. Les autres cas prévus fonctionnent.

TABLE 5.6 – Tests fonctionnels sur l'envoi de messages

Description du cas	Résultat attendu	OK/n-OK
Connexion d'un résident et connexion d'un responsable. Le résident envoie un message	Le responsable doit recevoir le message du résident	OK
Connexion d'un résident, envoi d'un message. Puis plus tard, connecter un responsable	Le responsable doit recevoir le message	n-OK
Connexion de deux résidents et un d'un responsable. Les deux résidents envoient un message.	Le résident recevra les messages	OK
Connexion d'un résident et d'un responsable. Le responsable envoie une question à un résident.	Le résident doit pouvoir répondre à la question et le responsable devra recevoir la réponse.	OK

Tests des alarmes

Les tests d'envoi d'une alarme ont permis de constater que le cas d'un responsable qui se connecte après l'envoi d'une alarme ne repasse pas par le moniteur. Ce cas n'a pas encore été traité. Le cas de déplacer un responsable en bas de liste quand il refuse une alarme n'a pas été traité non plus. Les autres cas prévus fonctionnent.

Description du cas	Résultat attendu	OK/n-OK
Connexion d'un responsable, mise à jour du status à disponible, connexion d'un résident et envoi d'une alarme.	Le responsable doit recevoir l'alarme	OK
Connexion d'un résident, envoi d'une alarme. Puis plus tard, connecter un responsable et mettre à jour son status à disponible	Le responsable doit recevoir l'alarme	n-OK
Connexion de deux responsables, mise à jour de leur status à disponible, connexion de deux résidents et les deux envoient une alarme	L'algorithme doit assigner un responsable à chaque résident	OK
Connexion de deux responsables, mise à jour de leur status à disponible, connexion d'un seul résident et les deux envoient une alarme	L'algorithme va assigner le responsable à un résident, et après que l'alarme soit validée assigner le résident à la seconde alarme	OK
Connexion d'un responsable, mise à jour de son status à disponible. Connexion d'un résident qui envoie une alarme. Le responsable refuse de prendre l'alarme.	Comme il n'y a qu'un responsable, l'alarme sera à nouveau envoyée au responsable	OK
Connexion de deux responsables, mise à jour de leurs status à disponible. Connexion d'un résident qui envoie une alarme. Le premier responsable refuse de prendre l'alarme.	L'alarme est envoyée au second responsable	n-OK
Connexion de deux responsables, mise à jour de leurs status à disponible. Connexion d'un résident qui envoie une alarme. Le premier responsable refuse de valider l'alarme.	L'alarme est envoyée au second responsable	n-OK

5.1.3 Scénario de tests de l'application de monitoring

Les scénarios de tests de l'application de monitoring ont permis de définir qu'il y a un problème concernant l'ajout d'un responsable. Ce problème n'a pas été corrigé.
La consultation des administrateurs ainsi que leurs ajouts n'ont pas été implémentés dans l'application Web.

TABLE 5.7 – Scénario de tests de l'application de monitoring

Description du cas	Résultat attendu	OK/n-OK
Tentative de connexion sans nom utilisateur ni/ou mot de passe	Le bouton ne peut pas être pressé	OK
Connexion avec un compte invalide	Erreur de connexion	OK
Connexion avec un compte valide	Redirection vers la page d'accueil	OK
Les informations de logs et la position des acteurs sont mis-à-jour	Le rafraîchissement est visible	OK
Utilisation de la barre de recherche	La barre de recherche permet la recherche de nom ou de prénom	OK
Déconnexion d'un utilisateur	Redirection vers le login	OK
Consultation de la liste des résidents	Liste des utilisateurs affichée	OK
Ajout d'un résident	Le nouveau résident peut-être ajouté	OK
Consultation de la liste des responsables	Liste des responsables affichée	OK
Ajout d'un responsable	Le nouveau responsable a été ajouté	n OK
Consultation de la liste des adminsitrateurs	Liste des administrateurs affichée	n-OK
Ajout d'un administrateur	Le nouvel administrateur a été ajouté	n-OK

5.2 Tests utilisateurs

Les tests utilisateurs suivants ont été effectués pour l'application de monitoring.

5.2.1 Premier test

Les résultats du premier test ont permis de constater que la barre de recherche permettant d'effectuer uniquement des recherches par "nom" et "prénom" n'est pas très clair d'utilisation.

Description du cas	OK/nOK	Constat
Connexion de l'administrateur admin/admin	OK	
Chercher le résident qui se trouve dans la salle 2	nOK	Le premier réflexe a été d'écrire "salle 2" dans la barre de recherche. Son second réflexe était correct.
Chercher dans la liste de résidents celui ayant l'id 4	OK	
Ajouter un résident	OK	
Déconnexion de l'application Web	OK	

5.2.2 Second test

Ce second test utilisateur a permis de constater qu'une barre de recherche dans les listes de résidents et responsables serait important.

Description du cas	OK/nOK	Constat
Connexion de l'administrateur admin/admin	OK	
Chercher dans la liste de résident celui ayant l'id 4	OK	
Ajouter un résident	nOK	Le premier réflexe a voulu être d'utiliser la barre de recherche (indisponible pour la liste des résidents et responsables). Son deuxième réflexe était correct.
Déconnexion de l'application Web	OK	

5.3 Synthèse

Les différents tests ont été commentés dans ce chapitre. Certains tests ont permis de définir qu'il n'y avait encore des cas qui n'ont pas été traités, notamment au niveau de la concurrence. Concernant les tests utilisateurs de l'application Web, il est possible de constater que malgré quelques difficultés rencontrées par les utilisateurs, l'application semble facile d'apprentissage. Dû au haut niveau du domaine visé pour le moment (rôle administrateur), cela semble satisfaisant.

5.4 Continuité

Le prochain chapitre servira la conclusion du projet.

6. Infrastructure

Dans ce chapitre sont décrites les informations sur l'infrastructure, comprenant les informations sur les projets effectués.

6.1 Serveur métier

Le serveur métier est effectué en JavaEE, avec Gradle pour charger les dépendances.

6.1.1 Dépendances

Les dépendances du serveur métier sont définies dans Gradle, comme représentées dans le code 6.1. Il s'y trouve les dépendances à uMove, au logiciel Firebase, à Spring, ainsi que pour les accès à la base de données.

Listing 6.1 – Dépendances Gradle

```
dependencies {
  // spring components
  compile("org.springframework.boot:spring-boot-starter-web")
  testCompile('org.springframework.boot:spring-boot-starter-test')
  compile('org.springframework.boot:spring-boot-starter-data-jpa')

  // connectors
  compile ('mysql:mysql-connector-java')

  // other components
  testCompile('com.jayway.jsonpath:json-path')
  testCompile group: 'junit', name: 'junit', version: '4.12'
  compile("com.google.firebase:firebase-admin:5.1.0")
  compile files("library/uMove.jar"))
}
```

6.2 Serveur Web

Le serveur Web utilisé pour la mise en place de l'application de monitoring est un NodeJS. La solution Angular 2 nommé Angular-Cli a été installé grâce à NPM.

```
npm -g install angular/cli
```

Compilation et exécution du serveur Web La commande |ngserve| permet la compilation et l'exécution du serveur Web Angular2.

6.3 Outils de développement

Les outils de développement principalement utilisés pour le projet sont les suivants :

JetBrains - WebStorm IDE utilisé pour le développement de l'application Web Angular 2.

JetBrains - IntelliJ IDE utilisé pour le développement de l'application Java EE.

7. Conclusion

Sommaire

7.1 Travail effectuées

Les objectifs définis dans le cahier des charges ont été atteints.

7.2 Problèmes rencontrés

Les problèmes rencontrés principaux sont les suivants :

Problème d'utilisation de Maven et Spring Au début du projet, j'ai rencontré un problème de compilation des projets utilisant Maven et Spring. Après recherche, je me suis rendu compte qu'il y avait un problème de compilation sur les postes utilisant le pilote de Nvidia 378.49. Cette version de pilote était composé d'une bogue qui faisait passer la compilation par le logiciel Cuda, qui n'arrivait pas à le compiler. Après désinstallation du pilote graphique, les logiciels fonctionnaient correctement.

7.3 Modifications prévues pour la défense

Pour la défense sont prévues les modifications et corrections suivantes :

- Correction des problèmes énoncés dans les tests.
- Implémentation de l'ajout et de la consultation de la liste des administrateurs
- Amélioration de la carte du monde présentée dans l'application Web, avec l'ajout d'un jardin et des salles ayant une identité propre (salon, salle à manger, toilettes, etc.)

7.4 Perspectives d'extension

Les perspectives d'extension suivantes seraient possibles et permettraient l'amélioration du projet.

7.4.1 Serveur de l'environnement intelligent

- Remplacer la localisation dans le bâtiment par des capteurs aux portes
- Implémenter le timeout des résidents et responsables s'ils ne mettent pas à jour leur status au bout d'une certaine période

7.4.2 Serveur de communication REST

— Utiliser un serveur WebSocket au lieu d'un serveur REST. Un serveur de WebSocket permettre de diminuer la charge du réseau en n'utilisant pas du long-polling, ce qui permettrait au serveur d'annoncer les changements aux clients, qui sont les applications mobiles et l'application Web de monitoring. De ce fait, il n'y aurait pas besoin d'utiliser le cloud FCM de Google pour la communication.
— Utiliser le mode de transport de HTTPS avec un système de Token

7.4.3 Algorithme de traitement des alarmes et des messages

— Effectuer du Machine Learning sur un grand dataset de données pour calculer les priorités de manière très précises suivant les types d'alarmes et leurs conséquences.

7.4.4 Application Web de monitoring

— Ajouter des bruitages lorsque les résidents envoient des alarmes ou se déplacent
— Implémenter des droits d'accès différents pour les responsables, qui pourraient uniquement consulter les utilisateurs mais ne pas pouvoir en ajouter.

7.5 Conclusion personelle

Ce projet m'a permis d'apprendre à utiliser Spring, JPA, ainsi que de revoir les matières apprises durant les cours de Systèmes d'Informations et d'Ihm effectués à la Haute-Ecole d'Ingéniérie et d'Architecture de Fribourg HEIA-FR.
J'aurais personnellement préféré pouvoir appronfondir l'élaboration des différentes tâches du projet, ce qui n'a malheureusement pas pu être le cas avec les 4 tâches fournies. Par exemple, j'aurais bien apprécié pouvoir établir l'application Web en plusieurs semaines au lieu de n'avoir qu'une seule semaine à disposition à la fin du projet.

Je tiens particulièrement à remercier le soutien de mes superviseurs et mon expert durant ce projet. Je remercie également les deux personnes d'avoir accepté d'effectuer un test utilisateur ainsi que les diverses personnes qui m'auraient aidées à effectuer le projet. Je remercie aussi mes deux collègues des projets RMMA et HMMA pour l'excellente coopération effectuée en groupe.

7.6 Déclaration d'honneur

Je, soussignée, William Greppi, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toutes autres formes de fraudes. Toutes les sources d'information utilisées et les citations d'auteurs ont été clairement mentionnées.

Lieu et date : Fribourg, le 14.07.17
Signature : William Greppi

7.7 Contenu du CD

Le contenu du CD est le suivant :

- **Code**
  - SEMS-App : contenant le projet Angular 2 de l'application Web.
  - SEMS-Server : contenant le projet JavaEE du serveur métier.
- **Documentation**
  - PV : contient les procès-verbaux effectués
  - Rapport : contient les différentes versions du rapport
  - Cahier des charges : contient les différentes versions du cahier des charges

8. Références

Pour consulter davantages d'informations, les références de ce projet sont définis dans l'énumération suivante.

1. Thèse de doctorat de M. Bruegger :
   `https://doc.rero.ch/record/24442/files/BrueggerP.pdf`
2. Rapport de projet de M. Meuwly :
   `https://multidoc.eia-fr.ch/record/1762/files/Report_SmartEnvironmentManager.pdf`
3. Le projet HMMA de Aurélien Etienne
4. Le projet RMMA de Rafic Galli
5. Patient Data Viewer :
   `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6139641`
6. Mobile Health :
   `http://www.sciencedirect.com/science/article/pii/S1532046415001136`

9. Sources

Les sources des éléments utilisées dans ce rapport sont définies dans l'énumération suivante.

1. Listing java en latex :
   `https://tex.stackexchange.com/questions/115467/listings-highlight-java-annotations`
2. Logo JavaFX :
   `https://lankydanblog.files.wordpress.com/2017/01/javafx.png`
3. FCM :
   `https://developer.xamarin.com/guides/android/application_fundamentals/notifications/firebase-cloud-messaging/Images/03-downstream.png`
4. Logo Angular 2 :
   `https://angular.io/assets/images/logos/angular/angular.png`

10. Glossaire

Dans ce chapitre se trouvent les différentes informations du glossaire.

10.1 Abbréviations

Les abbréviations importantes utilisées dans le rapport sont les suivantes :
— HTML : HyperText Markup Language
— FCM : Firebase Cloud Messaging
— HMMA : Health Mobile Monitoring App
— RMMA : Resident Mobile Monitoring App

10.2 Accronymes

Les accronymes techniques importants utilisés dans le projet sont les suivants :

Spring Spring est la librairie utilisée pour implémenter facilement et rapidement un serveur REST en Java.

Angular 2 Angular 2 est un framework simplifiant le développement d'application Web.

Serveur REST Un serveur REST est un modèle de serveur permettant l'exécution de méthodes distantes standardisées.

uMove uMove est la librairie développée codée par M. Bruegger représentant le concept d'un système d'un environnement intelligent.

Table des figures

Listings