



# Projet Fédérateur - EldyCare

---

ELDERLY CARE AND MONITORING

*Réalisé par :*

ALAMI IBN JAMAA Hamza

CHARIF Mehdi

HOUSNI Badreddine

OUHADI Yassine

*jury :*

BAINA Salah

BERRADA Bouchra

École nationale supérieure d'informatique et d'analyse des systèmes

Troisième année Génie Logiciel

Année Universitaire 2023-2024

## Résumé

---

Ce rapport présente Eldycare, une application mobile dédiée à la surveillance des personnes âgées. L'application offre des fonctionnalités de détection des chutes, d'envoi d'alertes en temps réel et de rappels. Elle utilise une architecture microservices et des technologies modernes. Le projet a suivi une approche de design thinking pour répondre aux besoins fonctionnels et non fonctionnels. L'application a été développée de manière robuste et évolutive, et contribue à améliorer la sécurité et le bien-être des personnes âgées. Des perspectives d'amélioration incluent l'intégration de nouvelles fonctionnalités et l'analyse des données de santé.



# Table des matières

<b>1</b>	<b>Introduction au contexte</b>	<b>1</b>
1.1	Présentation du contexte . . . . .	1
1.2	But du projet . . . . .	3
1.3	Conclusion . . . . .	3
<b>2</b>	<b>État de l'art</b>	<b>4</b>
2.1	Détection de chute . . . . .	4
2.1.1	Approche Algorithmique . . . . .	4
2.1.2	Approche ML . . . . .	5
2.2	Architecture Microservice . . . . .	5
2.2.1	Architecture dirigé par les évènements - EDA . . . . .	6
2.2.2	Websockets . . . . .	6
<b>3</b>	<b>Analyse et conception</b>	<b>7</b>
3.1	Design Thinking . . . . .	7
3.1.1	Désirabilité . . . . .	7
3.1.2	Faisabilité . . . . .	7
3.1.3	Viabilité . . . . .	7
3.2	Cahier de charge . . . . .	7
3.2.1	Besoins Fonctionnels . . . . .	8
3.2.2	Besoins non Fonctionnels . . . . .	8
3.2.3	Contraintes . . . . .	8
3.2.4	Diagramme Cas d'utilisation . . . . .	8
3.2.5	Méthodologie et organisation du projet . . . . .	10
3.3	Relation entre les utilisateurs . . . . .	11
3.4	Architecture Fonctionnelle . . . . .	12
3.5	Fonctionnalité d'envoi de rappels . . . . .	12
3.5.1	Diagramme de séquence pour l'envoi des rappels . . . . .	13
3.6	Fonctionnalité d'alerte . . . . .	13
3.6.1	Approche de détection de chute . . . . .	13
3.6.2	Diagramme de séquence pour l'envoi des alertes . . . . .	14
3.6.3	Notification de l'alerte . . . . .	14
3.7	Interface utilisateur . . . . .	15
3.8	Graphe de Navigation . . . . .	15
3.9	Smartphone UI . . . . .	16
3.9.1	Page d'inscription . . . . .	16
3.9.2	Page d'identification . . . . .	17
3.9.3	Pages dédiée aux contacts d'urgence . . . . .	18

3.9.4	Page dédiée aux personnes âgées . . . . .	19
3.9.5	Notification d'alerte . . . . .	20
3.10	Smartwatch UI . . . . .	21
3.11	Conclusion . . . . .	21
<b>4</b>	<b>Réalisation et résultats</b>	<b>22</b>
4.1	Outils utilisés . . . . .	22
4.1.1	Frameworks . . . . .	22
4.1.2	Devops . . . . .	23
4.1.3	Cloud . . . . .	24
4.2	Conteneurs docker . . . . .	25
4.3	Service de découverte eureka . . . . .	25
4.4	Service RabbitMQ . . . . .	26
4.5	Utilisation de WebSockets . . . . .	27
4.6	Pipeline CI/CD . . . . .	29
4.7	Intégration avec DockerHub . . . . .	30
4.8	Déploiement AWS . . . . .	30
4.9	Conclusion . . . . .	32

# Table des figures

1.1	Nombre d'arrêts cardiaques selon la tranche d'âge et le sexe. . . . .	2
1.2	Prévalence des blessures autodéclarées résultant d'une chute au cours des 12 derniers mois, selon l'âge et le sexe, population à domicile âgée de 65 ans ou plus, Canada, 2017-2018 . . . . .	2
1.3	Logo application . . . . .	3
2.1	Organigramme de la TBM . . . . .	5
2.2	Architecture dirigé par les évènements . . . . .	6
2.3	Communication Websocket . . . . .	6
3.1	Diagramme Cas d'utilisation . . . . .	9
3.2	Chronologie des sprints . . . . .	10
3.3	Tableau Kanban Trello . . . . .	10
3.4	Graphe d'utilisateurs . . . . .	11
3.5	Architecture Fonctionnelle . . . . .	12
3.6	Diagramme de séquence pour l'envoi des rappels . . . . .	13
3.7	Précision de la méthode algorithmique TBM et ML . . . . .	14
3.8	Diagramme de séquence pour l'envoi des alertes . . . . .	14
3.9	Graphe de Navigation . . . . .	15
3.10	Pages d'inscription . . . . .	16
3.11	Pages d'identification . . . . .	17
3.12	Pages dédiée aux contacts d'urgence . . . . .	18
3.13	dédiée aux personnes âgées . . . . .	19
3.14	Notification Eldycare . . . . .	20
3.15	Notification d'alerte en mode normal et étendu . . . . .	20
3.16	Interface SmartWatch . . . . .	21
4.1	Docker Containers . . . . .	25
4.2	Instances enregistrées avec Eureka . . . . .	25
4.3	Filtre d'authentification . . . . .	26
4.4	Inscription d'un nouvel utilisateur . . . . .	26
4.5	Mécanisme de diffusion des alertes . . . . .	27
4.6	Mécanisme de diffusion des rappels . . . . .	28
4.7	Github Jobs . . . . .	29
4.8	CICD Pipeline . . . . .	29
4.9	Repo DockerHub . . . . .	30
4.10	EKS Cluster . . . . .	31
4.11	la liste des pods . . . . .	31
4.12	la liste des services . . . . .	31

# Introduction Générale

---

Le rapport présenté ici concerne le projet de fin d'année portant sur l'application mobile Eldycare, dédiée à la surveillance des personnes âgées. Ce projet a été réalisé dans le cadre de notre troisième année de Génie Logiciel à l'École nationale supérieure d'informatique et d'analyse des systèmes.

L'objectif principal de notre projet Eldycare est de fournir une solution technologique pour améliorer la prise en charge et la sécurité des personnes âgées vivant à domicile. Nous sommes conscients des défis que rencontrent souvent les personnes âgées, notamment en matière de chutes accidentelles et de besoins d'assistance médicale. Notre application vise à répondre à ces problématiques en offrant un système de surveillance et de suivi adapté.

Dans ce rapport, nous aborderons différentes étapes de développement du projet Eldycare, allant de l'analyse et la conception à la réalisation et aux résultats obtenus. Nous présenterons également les outils et les technologies que nous avons utilisés tout au long du processus de développement.

Nous commencerons par une présentation du contexte du projet, en mettant en évidence les enjeux liés à la prise en charge des personnes âgées. Ensuite, nous passerons à l'état de l'art, où nous examinerons les différentes approches de détection des chutes et les architectures microservices existantes.

Nous poursuivrons avec l'analyse et la conception du projet, où nous décrirons le processus de design thinking que nous avons suivi, ainsi que les besoins fonctionnels et non fonctionnels identifiés. Nous présenterons également l'architecture fonctionnelle de notre application et les fonctionnalités clés, telles que l'envoi de rappels et la détection des alertes.

Ensuite, nous aborderons la phase de réalisation du projet, où nous expliquerons les outils et les technologies que nous avons utilisés, tels que les conteneurs Docker, le service de découverte Eureka et le service RabbitMQ. Nous décrirons également notre pipeline CI/CD et le déploiement sur AWS.

Enfin, nous conclurons ce rapport en résumant les résultats obtenus, en mettant en évidence les contributions de notre projet Eldycare dans le domaine de la surveillance des personnes âgées. Nous discuterons des limites de notre solution et des perspectives d'amélioration pour le futur.



# Chapitre 1

## Introduction au contexte

dans ce chapitre, nous allons présenter le contexte de notre projet fédérateur, ainsi que présenter notre projet ELDYCARE et son but.

### 1.1 Présentation du contexte

L'avènement de la digitalisation a révolutionné de nombreux aspects de notre vie quotidienne, et le domaine de la santé n'a pas fait exception à cette transformation majeure. À l'intersection de la technologie et des soins de santé, la digitalisation offre un potentiel sans précédent pour améliorer l'efficacité, l'accessibilité et la qualité des services médicaux. Cette évolution rapide englobe une multitude d'innovations, allant des dossiers médicaux électroniques à la télémédecine, en passant par les applications de suivi de la santé. Ainsi, cette révolution numérique façonne l'avenir des soins de santé en redéfinissant la manière dont les professionnels de la santé interagissent avec les patients, en facilitant l'accès aux informations médicales et en ouvrant la voie à de nouvelles approches diagnostiques et thérapeutiques.

La technologie a créé un tissu de connexions qui transcende les frontières physiques, et cette interconnexion constante revêt une importance capitale dans le domaine de la santé. La possibilité d'être toujours connecté avec autrui, en particulier en cas d'urgence et d'anomalie médicale. Les dispositifs intelligents, tels que les smartphones et les montres connectées, jouent désormais un rôle central en permettant une surveillance continue de la santé, la transmission rapide d'informations cruciales aux professionnels de la santé, et la mise en place de réponses rapides en cas de situations critiques.

Selon une étude "Out-of-hospital cardiac arrest in the Pilsen Region in 2018" [4], les personnes ayant plus de 50 ans sont plus susceptibles d'avoir un arrêt cardiaque (figure 1.1)

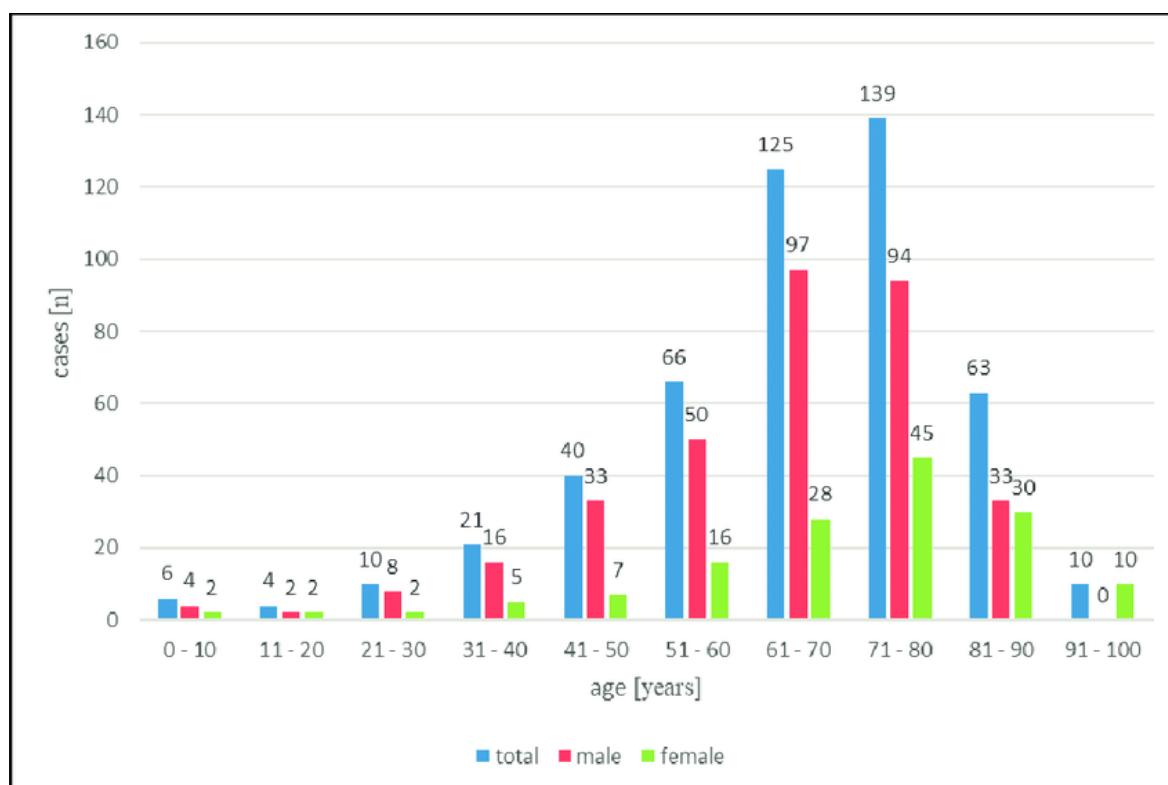


FIGURE 1.1 – Nombre d’arrêts cardiaques selon la tranche d’âge et le sexe.

Une autre étude ”Surveillance report on falls among older adults in Canada” [1] montre le pourcentage des blessures résultant d’une chute pour les personnes âgées de plus de 65ans (figure 1.2)

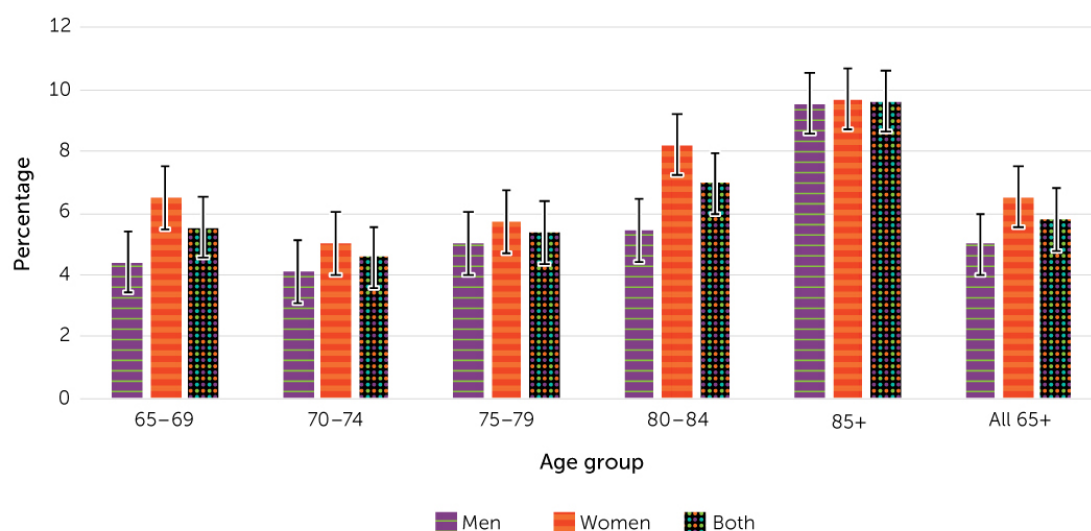


FIGURE 1.2 – Prévalence des blessures autodéclarées résultant d’une chute au cours des 12 derniers mois, selon l’âge et le sexe, population à domicile âgée de 65 ans ou plus, Canada, 2017-2018



## 1.2 But du projet

Notre application mobile *ELDYCARE* a pour but d'assurer une interconnectivité entre les personnes âgées, et leurs proches en cas d'anomalie médicale, et cela en liant les senseurs d'une smartwatch que la personne âgée va porter, avec notre application. Quelque soit l'emplacement du proche, si un problème survient chez la personne âgée, on recevra une alerte qui permettra non seulement d'identifier et appeler la personne, mais aussi de la localiser. Notre application donnera aussi la capacité au proche d'envoyer des rappels qui seront enregistré chez la personne âgée dans son calendrier.



FIGURE 1.3 – Logo application

## 1.3 Conclusion

Dans cette partie, nous avons présenté le contexte et le but de notre application *ELDYCARE*. La section suivante sera consacrée à l'état de l'art et les méthodologies utilisées dans notre application.

# Chapitre 2

## État de l'art

Dans ce chapitre, nous allons faire un état de l'art, notamment l'approche de détection de chute, l'architecture microservice, et l'architecture dirigée par les événements.

### 2.1 Détection de chute

L'un des défis de ce projet est la détection de chute de la personne âgée. La smart-watch est équipée de plusieurs capteurs de positions :

- Gyroscope
- Accéléromètre
- magnétomètre

Afin de détecter la chute, il faudrait traiter les informations brutes des capteurs. Selon un article publié par *de Quadros, Thiago and Lazzaretti, Andre Eugenio and Schneider* : "A Movement Decomposition and Machine Learning-Based Fall Detection System Using Wrist Wearable Device" [2], On peut utiliser 2 approches pour le traitement des signaux des capteurs de positions :

#### 2.1.1 Approche Algorithmique

Cette approche intitulé dans l'article "*Threshold based Method - TBM*" se base sur le calcul des composantes verticales de l'accélération, de vitesse et de déplacement. Il suffit de faire passer les signaux des capteurs par un filtre pass bas pour réduire le bruit, puis enlever l'influence de la gravité. D'ailleurs, si le mouvement est intense, les signaux du gyroscopes vont être utilisé dans les calculs. Suite a cela, nous aurons l'accélération qu'on utilisera pour extraire sa composante verticale. Cette composante va être intégrée une fois pour trouver la vitesse, puis une autre fois pour calculer le déplacement. ces valeurs vont être comparé a un seuil (figure 2.1).

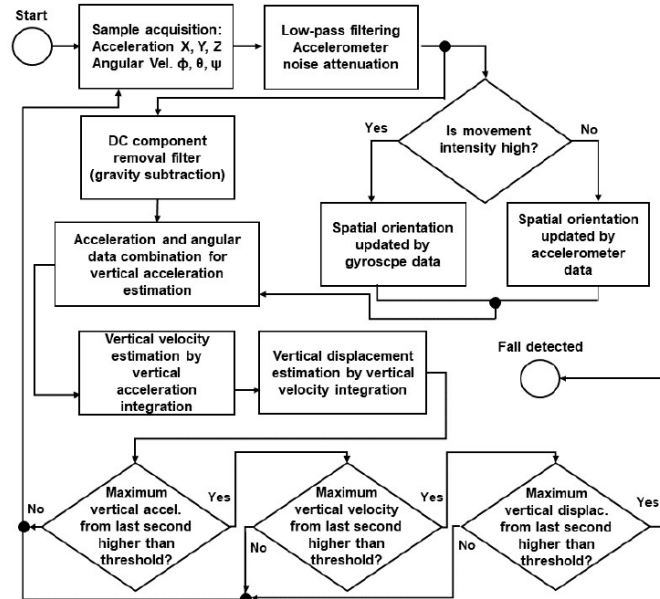


FIGURE 2.1 – Organigramme de la TBM

### 2.1.2 Approche ML

La deuxième approche est une approche machine learning en utilisant des algorithmes de classification, notamment *K-Nearest Neighbors*. Cette méthode est divisée en 2 étapes : L'extraction des caractéristiques depuis les données de capteurs traités (L'accélération, vitesse et déplacement et leurs composantes verticales), puis la classification.

## 2.2 Architecture Microservice

Une architecture microservices est une approche de conception logicielle où une application est décomposée en un ensemble de services autonomes et indépendants qui collaborent pour fournir la fonctionnalité globale de l'application. Chaque service dans cette architecture est un module distinct qui peut être développé, déployé et mis à l'échelle indépendamment des autres services.

Les principales caractéristiques d'une architecture microservices comprennent :

- Indépendance des Services
- Communication via des API
- Déploiement Indépendant
- Scalabilité Facilitée
- Technologie Variée
- Résilience
- Facilité de Développement
- Gestion Centralisée

### 2.2.1 Architecture dirigé par les évènements - EDA

L'architecture orientée événements (Event-Driven Architecture, EDA) est un style architectural dans lequel les composants d'un système communiquent entre eux en réagissant à des événements. Un événement représente un changement d'état ou une action significative qui se produit à l'intérieur du système ou dans son environnement. Ces événements sont souvent déclenchés par des actions utilisateur, des changements de données, des erreurs, etc.

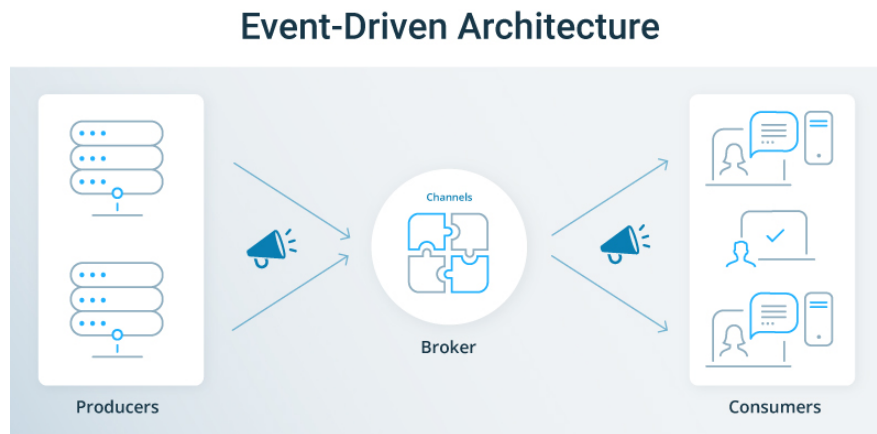


FIGURE 2.2 – Architecture dirigé par les évènements

### 2.2.2 Websockets

Les WebSockets sont un protocole de communication bidirectionnelle, en temps réel, qui permet une communication interactive entre un navigateur web et un serveur. Contrairement à la communication basée sur HTTP traditionnelle, qui suit le modèle de demande-réponse, les WebSockets permettent une communication en temps réel et bidirectionnelle, où le serveur peut envoyer des données au client sans attendre une demande explicite de ce dernier.

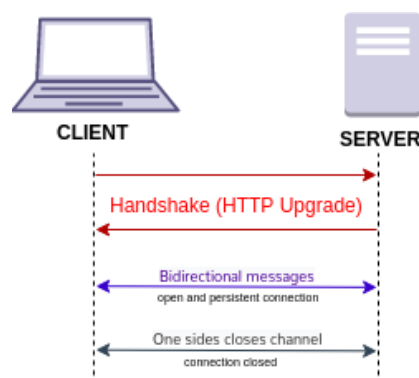


FIGURE 2.3 – Communication Websocket

# Chapitre 3

## Analyse et conception

Dans ce chapitre, nous allons Tout d’abord faire une analyse de Design Thinking du projet, et dresser un cahier de charge avec les besoins fonctionnels, non fonctionnels et les contraintes, ainsi que le diagramme cas d’utilisation et la méthodologie utilisé pour notre projet. Nous détaillerons l’architecture fonctionnelle de l’application et l’ensemble de ces fonctionnalités. Finalement, nous présenterons les maquettes des l’interface graphique.

### 3.1 Design Thinking

Nous allons en premier lieu analyser notre application Eldycare sous le thème de la santé, et plus précisément la surveillance des personnes âgées, sous trois axes :

#### 3.1.1 Désirabilité

Notre application est dans le contexte de la santé et répond à la problématique présentée. La valeur ajoutée consiste en la connexion constant entre les gens âgées et leurs proches en cas d’anomalie médicale ou de chute. Cette application est intéressante pour plusieurs utilisateurs comme les personnes vivants éloignés de leurs proches, ou ne sont pas a leurs disposition a 100% pour des raisons personnelles ou professionnelles.

#### 3.1.2 Faisabilité

Notre application est faisable vu qu’elle utilisera un ensemble de technologies préexistante, notamment une smartwatch contenant un ensemble de capteurs de position et de santé.

#### 3.1.3 Viabilité

Notre application à un bon potentiel vu le besoin, ainsi que les avancements technologiques qui rendent un traitement des requêtes en temps réel plus simple et rapide.

### 3.2 Cahier de charge

Dans ce qui suit, nous allons présenter l’ensemble des fonctionnalités, besoins et contraintes de notre application.

### 3.2.1 Besoins Fonctionnels

Les besoins fonctionnels de l'application sont comme suit :

- Surveiller la personne âgée à l'aide d'une Smart Watch
- Notifier contacts d'urgence associé à la personne au cas où une anomalie survient.
- Rappeler la personne âgée de faire des tâches (prendre des médicaments, aller chez le médecin...).
- Gestion des rappels par un relatif à l'aide d'une application.

### 3.2.2 Besoins non Fonctionnels

Parmi les besoins fonctionnels de l'application, on cite :

- Concevoir une interface intuitive pour les personnes quel que soit leur âge.
- Assurer un système résilient sans point de défaillance unique.
- Assurer un système performant.

### 3.2.3 Contraintes

- Assurer une connexion constante à internet
- Etant donné la conception d'une smart watch avec les capteurs nécessaires est impossible, nous allons simuler les données capturées pour les traiter.

### 3.2.4 Diagramme Cas d'utilisation

Afin d'illustrer ces besoins, nous présenterons le diagramme de cas d'utilisation du projet. Les acteurs de notre projet sont deux :

- Les personnes âgées (Elders)
- Les relatifs / contact d'urgence (relatives)

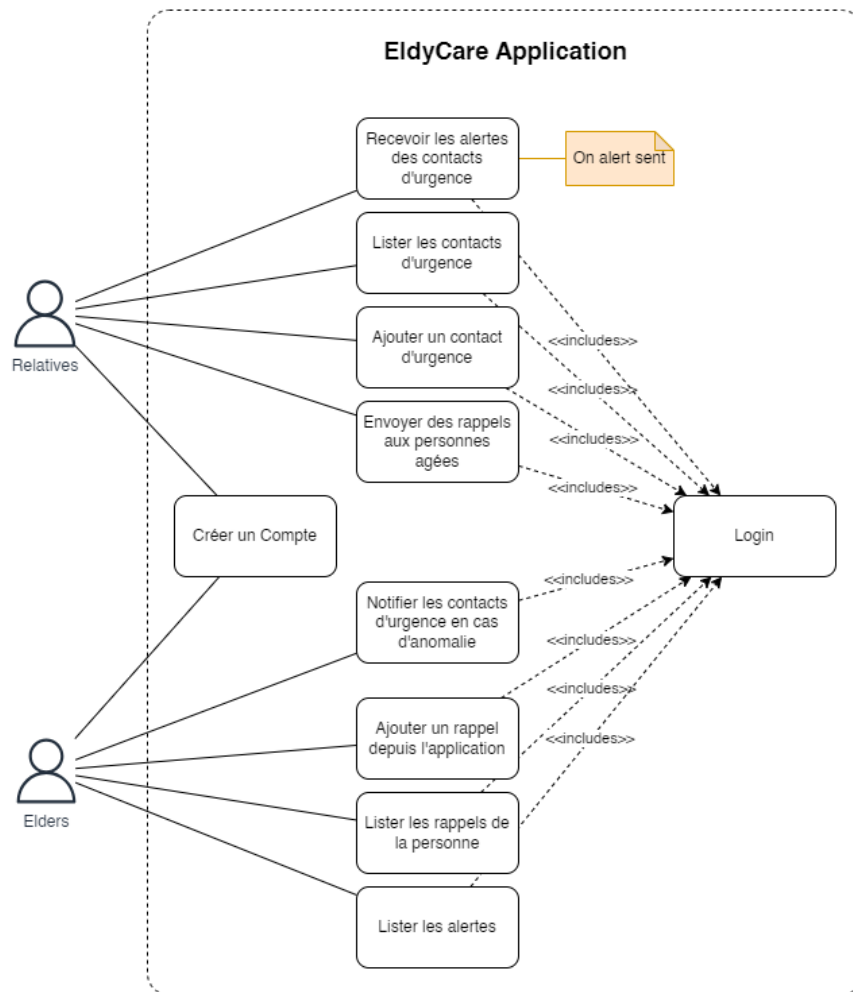


FIGURE 3.1 – Diagramme Cas d'utilisation

### 3.2.5 Méthodologie et organisation du projet

Afin d'organiser le déroulement et les tâches de notre projet, nous avons utilisé une méthodologie agile en combinaison avec la méthode Kanban. Nous avons effectué en total 4 sprints :

- Initialisation du projet : Architecturer et implémentation du projet : Front (interfaces utilisateurs, navigation de l'application mobile ...) et Back (implémentation des service, ajout de l'authentification ...)
- Ajout de le fonctionnalités d'alerte
- Ajout de le fonctionnalités de rappels
- Deployment de l'application sur AWS

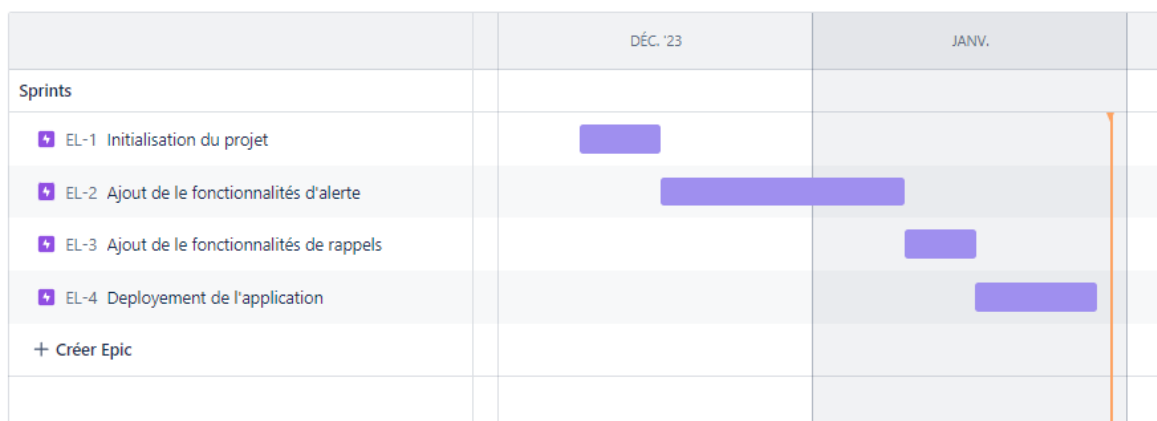


FIGURE 3.2 – Chronologie des sprints

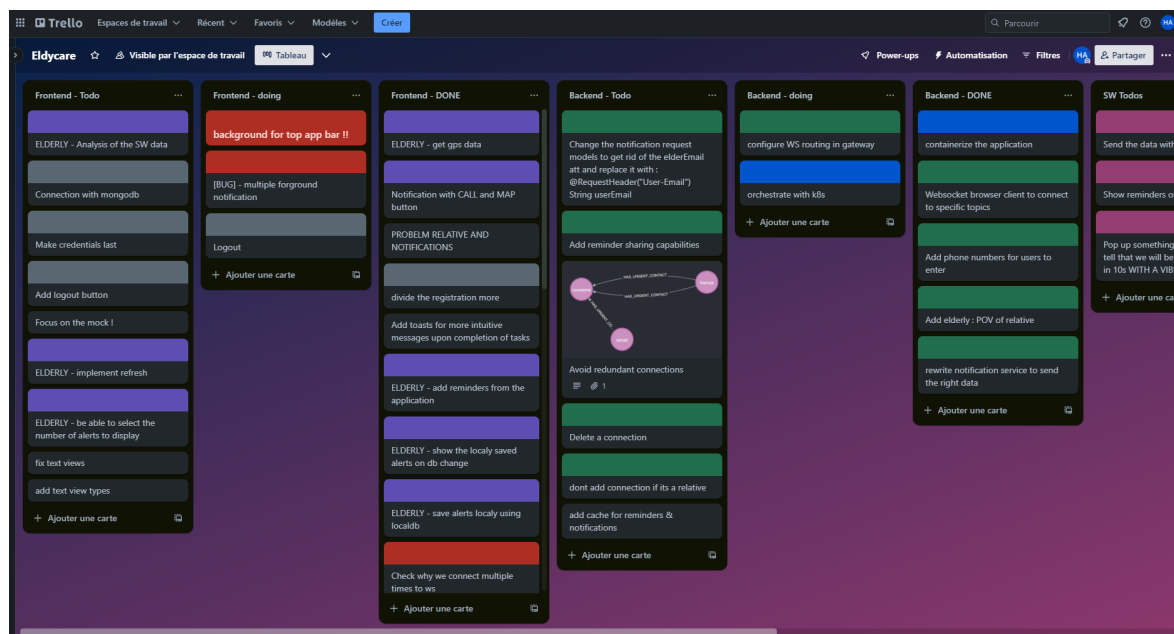


FIGURE 3.3 – Tableau Kanban Trello



### 3.3 Relation entre les utilisateurs

Comme décrit dans la section 3.2.4, notre application peut être utilisée par 2 profils d'utilisateurs. Le schéma ci contre modélise un exemple de relation entre les utilisateurs. chaque utilisateur possède des données propre a lui :

- Email
- Nom d'utilisateur
- Numéro de téléphone
- Type d'utilisateur

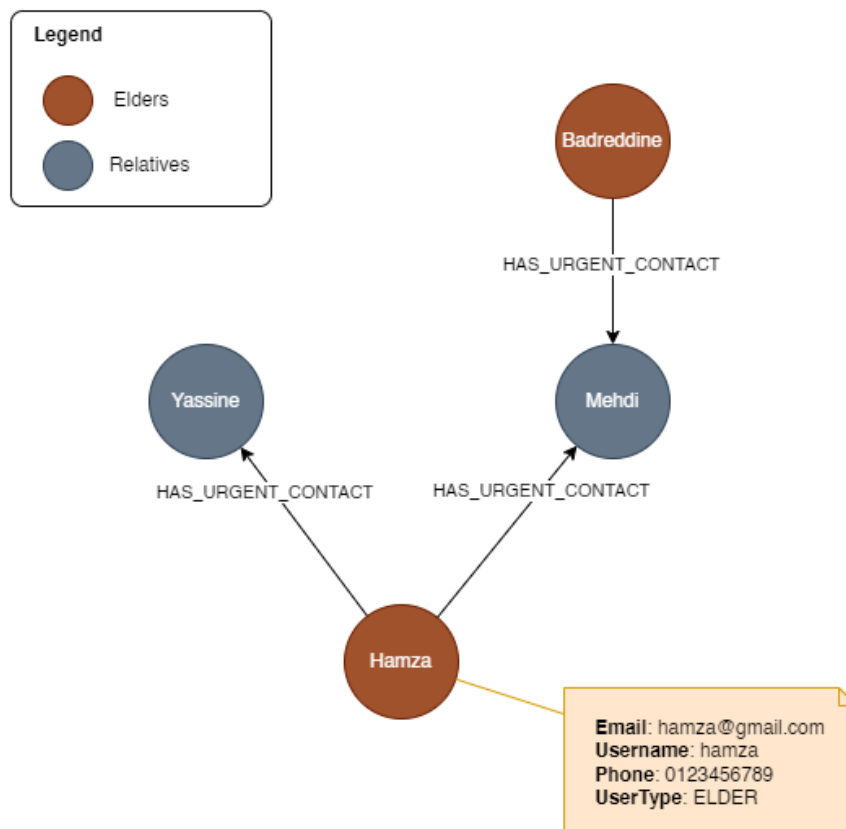


FIGURE 3.4 – Graphe d'utilisateurs

### 3.4 Architecture Fonctionnelle

Le diagramme 3.5 présente l'architecture fonctionnelle de notre application. Eldy-care est une application mobile avec la capacité de se connecter à la smartwatch de l'utilisateur pour détecter les signaux vitaux en cas de chute de l'utilisateur. Selon le type d'utilisateur, on aura 2 interfaces qui s'offre à nous, et pour les utilisateurs de type "Elder", on aura la communication avec la smartwatch en question. Étant donné une connexion internet, le téléphone mobile de l'utilisateur sera connecté au Backend de notre application, hébergé sur le cloud d'AWS. Nous avons opté pour une architecture en **microservice** vu les points positifs qu'elle apporte, notamment une bonne scalabilité, une meilleure expérience de développement et un couplage faible entre les services, nous laissant la possibilité d'effectuer des changements de manière rapide et simple.

Pour la communication inter-service, nous allons utiliser un broker / message-queue étant donné que notre application va suivre une architecture Event Driven. On note également l'utilisation des **websockets** pour la communication avec l'application mobile, afin d'assurer un traitement en temps réel.

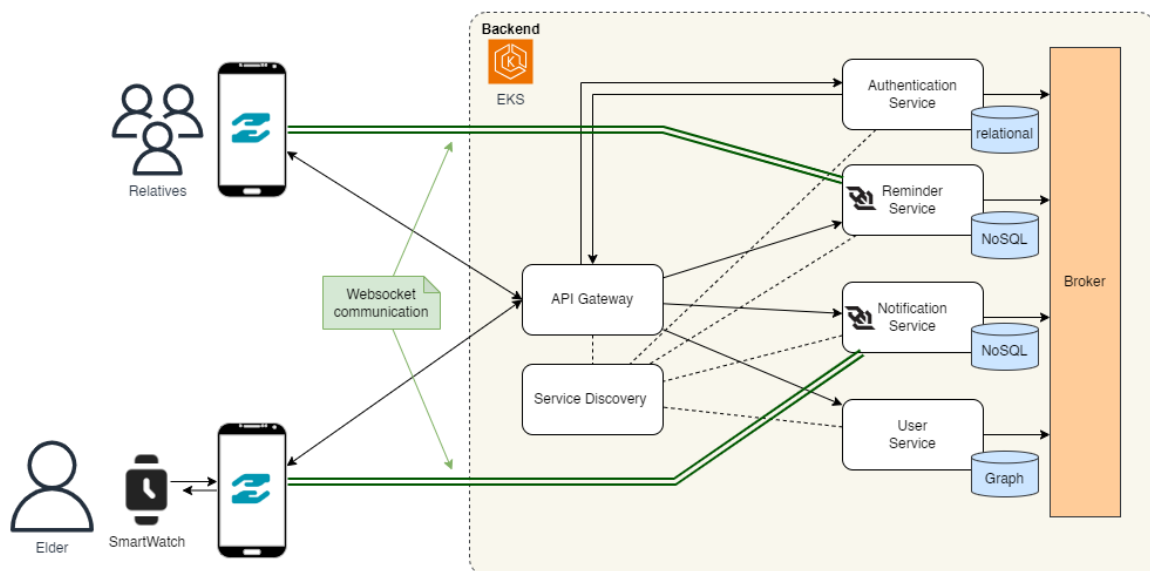


FIGURE 3.5 – Architecture Fonctionnelle

### 3.5 Fonctionnalité d'envoi de rappels

Parmi les fonctionnalités que notre application propose, nous avons la définition des rappels à distance.

A titre d'exemple, le proche prendra un rendez-vous pour la personne âgée, il pourra envoyer le rappel et ce dernier va être enregistré dans le calendrier de la personne en question. La personne âgée pourra voir la liste des rappels directement depuis l'application, et l'intégration avec la smartwatch lui permettra de toujours être notifié lorsque l'évènement approche.

### 3.5.1 Diagramme de séquence pour l'envoi des rappels

Voici un diagramme séquence décrivant le scénario d'envoi des rappels. L'interaction avec l'application mobile de la personne âgée doit être en temps réel, et donc utilisera les websockets comme technologie de communication.

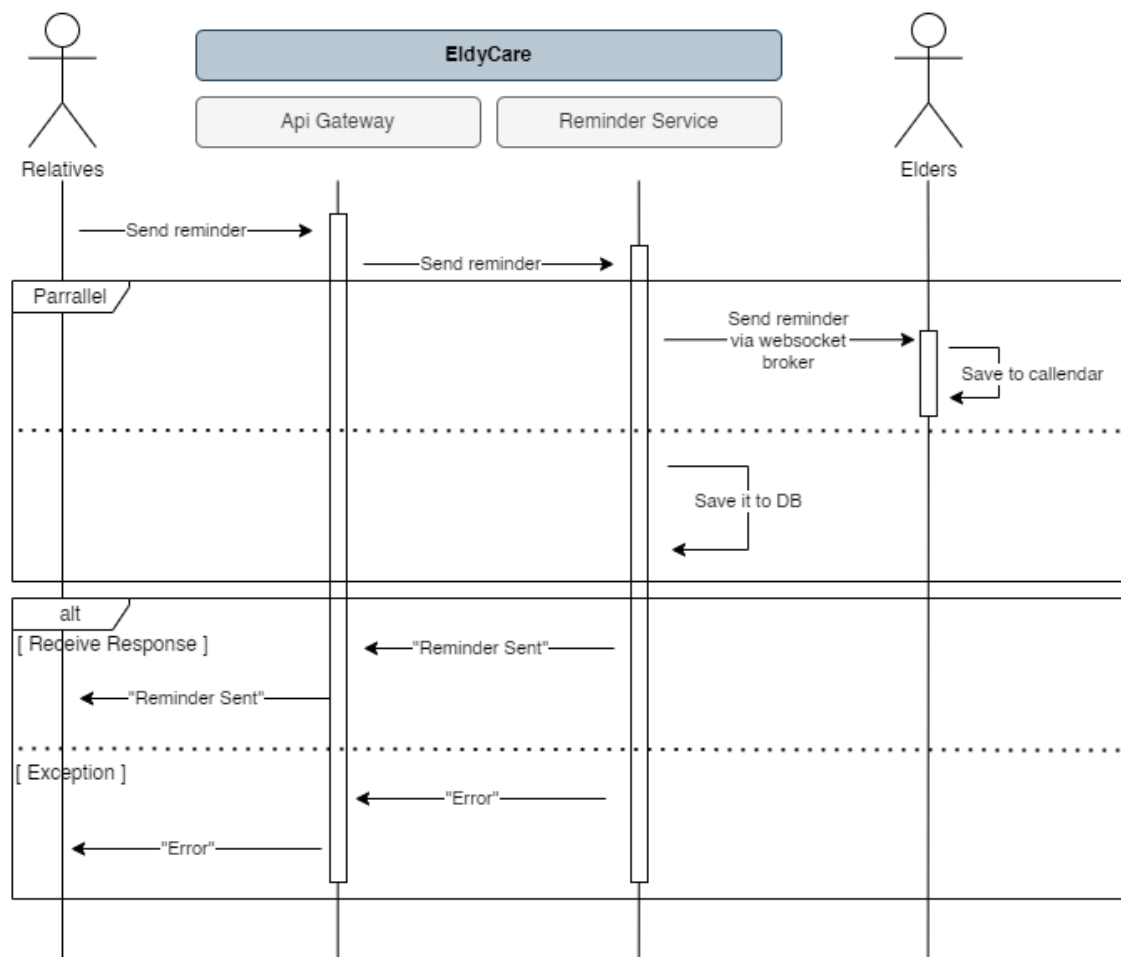


FIGURE 3.6 – Diagramme de séquence pour l'envoi des rappels

## 3.6 Fonctionnalité d'alerte

La fonctionnalité critique et principale de notre application est l'alerte des contacts d'urgence à l'issu d'un problème ou anomalie médicale, et ce grâce aux nombreux capteurs de position et de santé de la smartwatch. Dans cette itération du projet, on s'est focalisé sur l'aspect "Détection de chute".

### 3.6.1 Approche de détection de chute

Comme cité dans la section 2.1, nous avons 2 méthodes dont on pourrait choisir pour détecter la chute : Méthode algorithmique dite "Threshold based method TBM", et une méthode orienté ML. L'article [2] donne les précision des 2 méthode (figure 3.7) En comparant les précisions, on constate que la différence entre les deux méthodes

TABLE I  
EVALUATION OF DIFFERENT SIGNAL COMBINATION FOR TBM

Signal Combination	Sensitivity (%)	Specificity (%)	Accuracy (%)
(TA, TV)	<b>95.8</b>	<b>82.3</b>	<b>89.1</b>
(VA, TV)	91.7	82.3	87.0
(TA, VA, TV)	93.8	83.3	88.5
(TV)	86.5	80.2	83.3
(VA, VV, VD)	95.8	72.9	84.4

EVALUATION OF DIFFERENT SIGNAL COMBINATION FOR MLM

-	k-NN	LDA	LR	DT	SVM
Conf.	+Angles	(TA,VA,TV)	(VA,TV)	(VA,TV)	(VA,TV)
TP	96	95	94	94	94
TN	94	90	93	90	93
FP	2	6	3	6	3
FN	0	1	2	2	2
Sens.	100%	99.0%	97.9%	97.9%	97.9%
Spec.	97.9%	93.8%	96.9%	93.8%	96.9%
Accur.	<b>99.0%</b>	96.4%	97.4%	95.8%	97.4%

FIGURE 3.7 – Précision de la méthode algorithmique TBM et ML

n'est pas très grande, et étant donné que notre application doit être en temps réel, nous avons opté *d'utiliser la méthode algorithmique, implémenté directement sur la smartwatch*

### 3.6.2 Diagramme de séquence pour l'envoi des alertes

La figure ci contre illustre le diagramme séquence lors de la détection et l'envoi des alertes aux contacts d'urgence. Similairement à l'envoi des rappels, on utilisera les **websockets** comme technologie de communication.

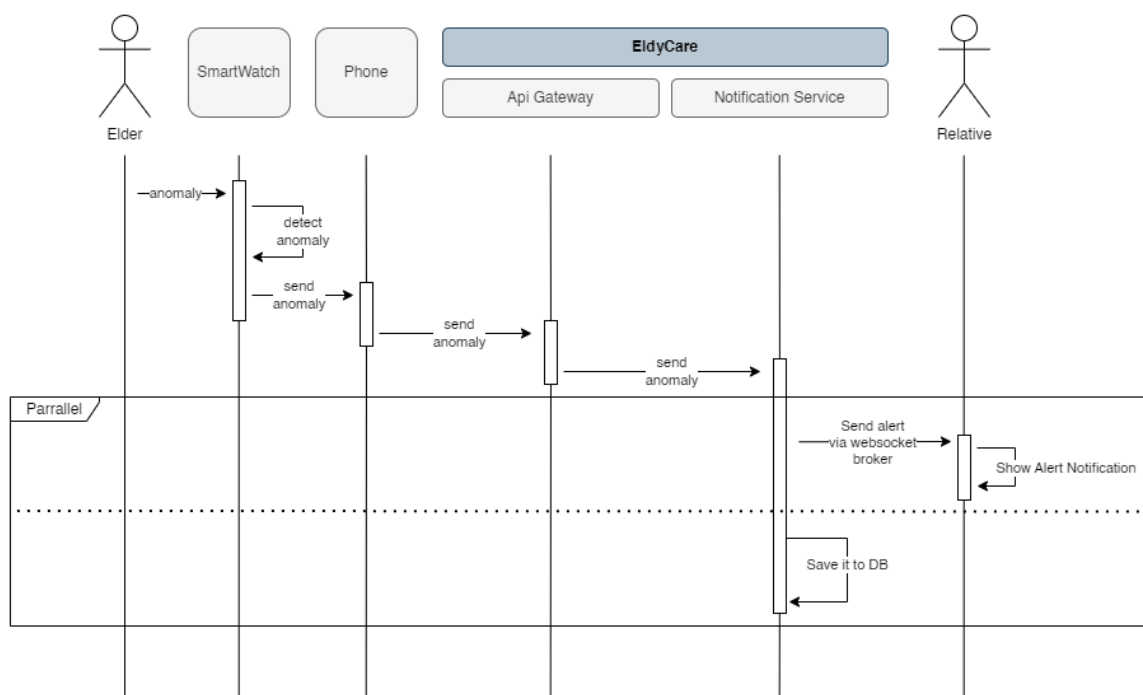


FIGURE 3.8 – Diagramme de séquence pour l'envoi des alertes

### 3.6.3 Notification de l'alerte

L'alerte chez le contact d'urgence doit être descriptive avec la capacité de :

- Savoir la cause de l'alerte
- Identifier la personne concernée

- Bouton d'action rapide pour appeler la personne
- Bouton d'action rapide pour localiser la personne lors du déclenchement de l'alerte

## 3.7 Interface utilisateur

La partie cliente de notre application ne sera réalisé que si on fait une conception détaillé de l'interface graphique de l'application mobile. Plusieurs caractéristiques doivent être présente dans notre conception :

- Une interface simple, minimale et intuitive pour être utilisable par toute tranche d'âge
- Un placement ergonomique et stratégique des boutons d'action
- Utilisation d'iconographie et d'image représentatives

Pour cela, on a utilisé un framework de conception de google : *"Material Design"* : créé par Google, c'est un langage visuel interactif servant de guide pour concevoir des interfaces graphiques. Inspiré d'objets réels comme le papier et l'encre, il se distingue du Flat Design. Utilisé par Google pour unifier le style graphique de ses applications, le Material Design offre une interface adaptable à tous les appareils et résolutions d'écran, améliorant l'expérience utilisateur (UX) et fournissant des ressources utiles aux développeurs et concepteurs [3].

## 3.8 Graphe de Navigation

La figure 3.9 illustre le flow de navigation de notre application mobile. on constate que après le login, nous aurons une redirection automatique vers les pages associé au type d'utilisateur identifié.

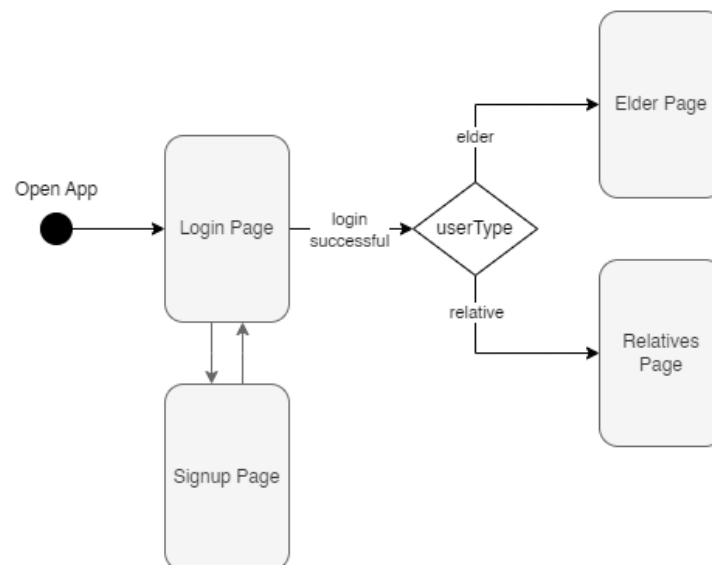


FIGURE 3.9 – Graphe de Navigation

## 3.9 Smartphone UI

### 3.9.1 Page d'inscription

La page d'inscription a été divisée en 3 parties que l'utilisateur va devoir remplir. Cette division est faite de telle sorte d'isoler les éléments selon le contexte (information de contact, information d'identification, information sur le type de l'utilisateur) et éviter d'encombrer l'interface. Nous avons aussi un bouton pour s'identifier si l'utilisateur a déjà un compte

The figure displays three sequential mobile app registration screens, each with a dark teal header and a white body.

- WHO ARE YOU:** Features an illustration of a woman with a 'HELLO' sign. It contains two input fields: 'Name' (with a 'Label' placeholder) and 'Phone Number' (with a 'Label' placeholder). A dark blue 'Next' button is centered below the fields. At the bottom, it asks 'Already have an account ?' with a 'Login' button.
- YOUR CREDENTIALS:** Features an illustration of a man carrying a large blue ring. It contains three input fields: 'Email' (with a 'Label' placeholder), 'Password' (with a 'Label' placeholder), and 'Confirm Password' (with a 'Label' placeholder). A dark blue 'Next' button is centered below the fields. At the bottom, it asks 'Already have an account ?' with a 'Login' button.
- WHICH ARE YOU:** Features two selectable options, each with an illustration: 'Close relative' (showing a family) and 'Elderly person' (showing an elderly couple). At the bottom, it asks 'Already have an account ?' with a 'Login' button.

FIGURE 3.10 – Pages d'inscription

### 3.9.2 Page d'identification

La page d'identification reste simple avec un champ pour l'email, et un champ pour le mot de passe. Nous avons aussi un bouton pour s'identifier si l'utilisateur à déjà un compte.

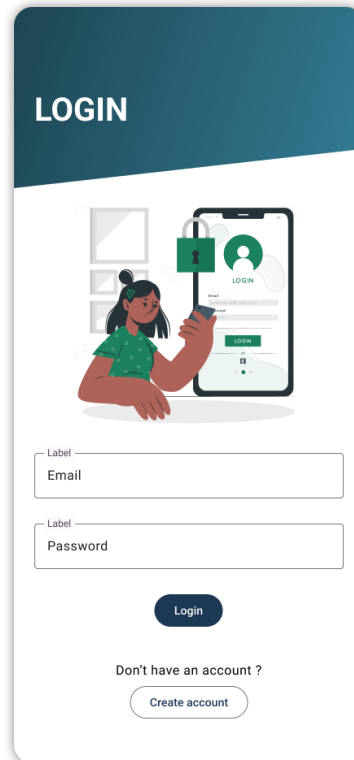


FIGURE 3.11 – Pages d'identification

### 3.9.3 Pages dédiée aux contacts d'urgence

Pour les proches / contacts d'urgence, nous aurons une interface simple, qui montre la liste des contacts d'urgence, et un bouton qui ouvre une fenêtre pour ajouter un contact d'urgence. L'utilisateur aura aussi la possibilité de cliquer sur l'un des contact pour lui envoyer un rappel grâce a un tiroir contenant les information de la personne, ainsi que la date, l'heure et le contenu du rappel.

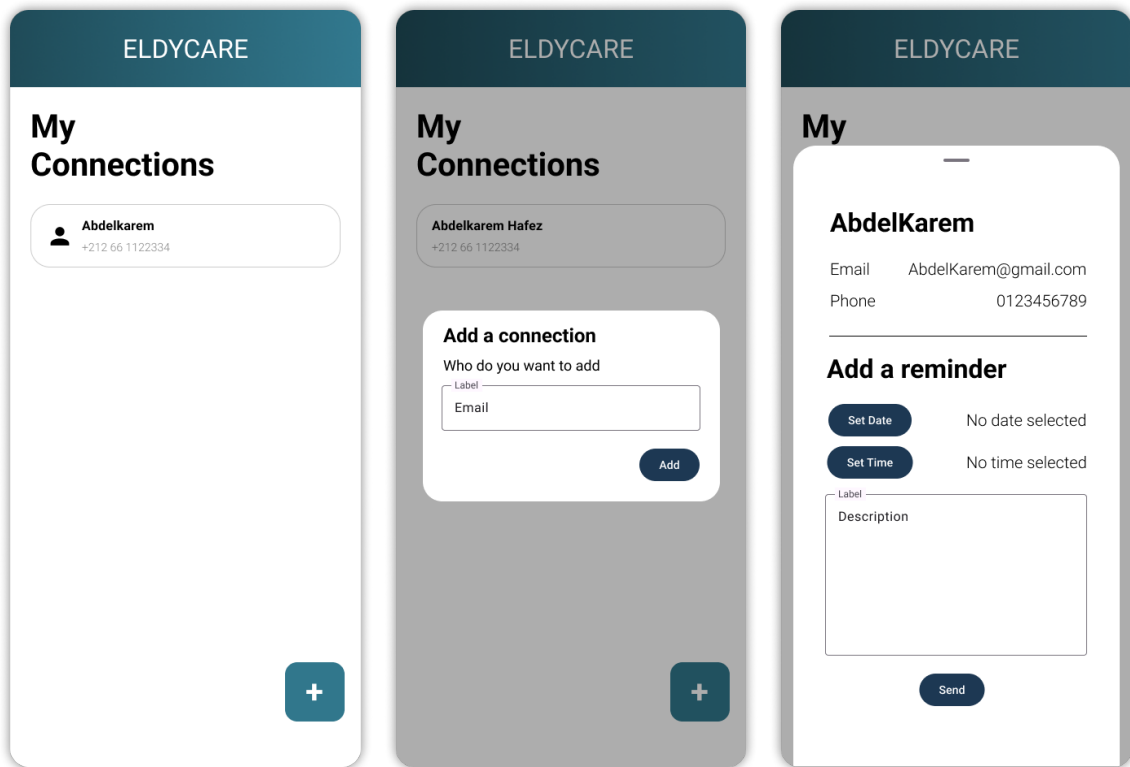


FIGURE 3.12 – Pages dédiée aux contacts d'urgence



### 3.9.4 Page dédiée aux personnes âgées

Pour les personnes âgées, nous aurons 2 pages que l'utilisateur peut y naviguer :

- Une page de rappel : listant l'ensemble des rappels relatifs à l'application, et la possibilité de créer un rappel directement dans l'application.
- Une page d'alerte : Listant l'ensemble des alertes et l'instant de leur occurrence.

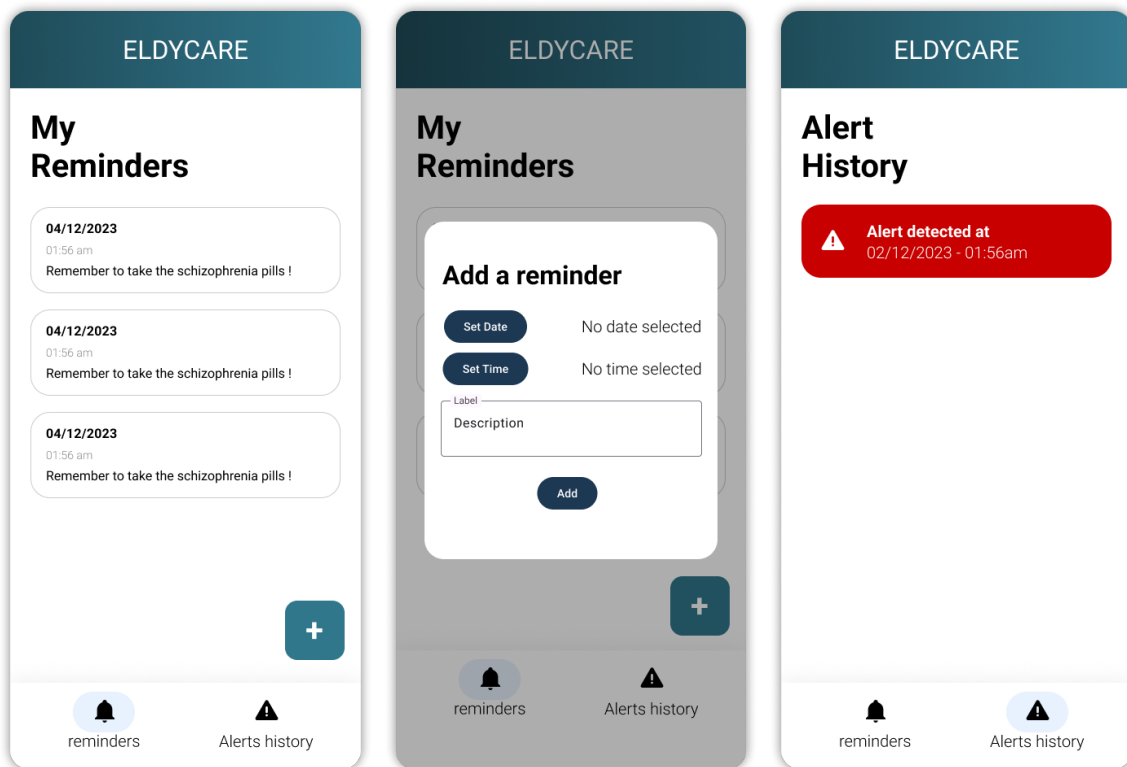


FIGURE 3.13 – dédiée aux personnes âgées

### 3.9.5 Notification d'alerte

Une fois connecté, notre application lance un processus en arrière plan. Pour s'assurer que le processus est effectivement démarré, nous aurons une notification fixe (figure 3.14). L'un des aspects importants de notre application sont les notifications d'alerte.

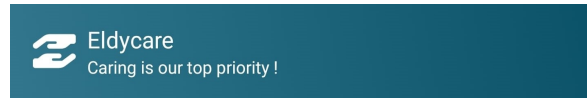


FIGURE 3.14 – Notification Eldycare

La notification aura 2 formes (figure 3.15) :

- Une forme normale : contenant une description courte de la cause de l'alerte
- Une forme étendue : contenant plus d'informations – instant de l'alerte, localisation, et la cause de l'alerte. 2 Boutons d'actions sont disponibles : Appeler la personne, et Localiser la personne en ouvrant Google Maps sur l'emplacement exact

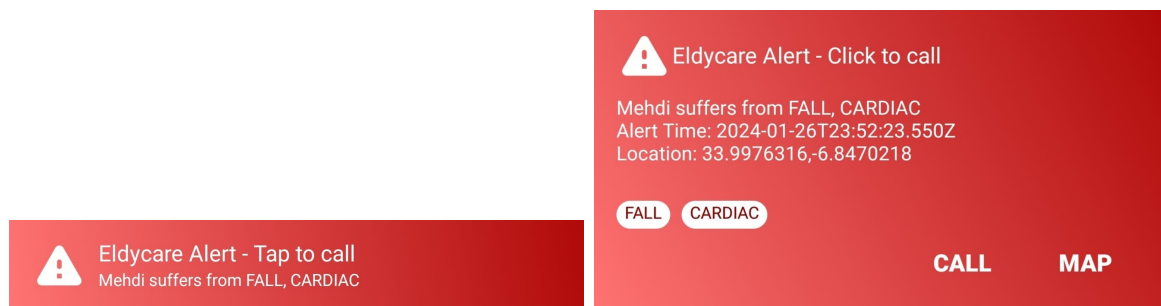


FIGURE 3.15 – Notification d'alerte en mode normal et étendu

## 3.10 Smartwatch UI

La smartwatch à comme but principal de surveiller les signaux vitaux de la personne et ses capteurs de position, et sera exécuté la majorité du temps en arrière plan. Par conséquent. Cependant, nous aurons un écran montrant le rythme cardiaque. D'autre part, les rappels seront géré automatiquement par l'application calendrier.

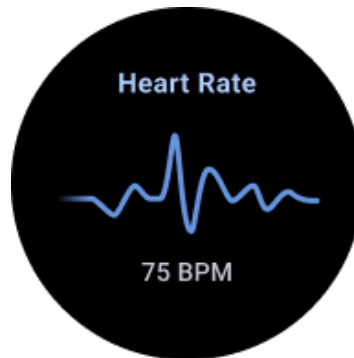


FIGURE 3.16 – Interface SmartWatch

## 3.11 Conclusion

Dans cette chapitre nous avons présenté le cahier de charge de l'application, l'architecture fonctionnelle de l'application, et l'ensemble des interfaces graphiques de notre application mobile. dans la section suivante, nous allons présenter nos résultats et réalisation.

# Chapitre 4

## Réalisation et résultats

### 4.1 Outils utilisés

Afin de réussir notre projet, nous avons utilisé plusieurs outils pour le développement de notre application

#### 4.1.1 Frameworks

##### **Spring Boot**

Spring Boot, un framework Java open-source, simplifie le développement d'applications en offrant des fonctionnalités prêtes à l'emploi et une approche convention-over-configuration. Son serveur d'application embarqué élimine la complexité du déploiement externe, faisant de Spring Boot un choix populaire pour le développement d'applications Java modernes.



##### **Android Jetpack Compose**

Android Jetpack Compose simplifie le développement d'interfaces utilisateur pour les applications Android. En adoptant une approche déclarative, cette bibliothèque open-source élimine la complexité du code XML. Jetpack Compose offre des fonctionnalités prêtes à l'emploi pour la gestion de l'état et la navigation, facilitant ainsi la création d'interfaces modernes et dynamiques sur Android.



### 4.1.2 Devops

#### Docker

Docker est une plateforme open-source automatisant le déploiement d'applications dans des conteneurs. Ces conteneurs fournissent un environnement léger et isolé, permettant aux développeurs de créer, déployer et exécuter des applications de manière cohérente, indépendamment de l'infrastructure sous-jacente. Docker simplifie ainsi le processus de gestion des dépendances logicielles, améliorant la portabilité et l'efficacité du développement d'applications.



#### Kubernetes

Kubernetes, souvent abrégé K8s, est une plateforme open-source d'orchestration de conteneurs, facilitant le déploiement, la gestion et l'évolutivité des applications conteneurisées. Il offre une automatisation robuste pour le déploiement, la mise à l'échelle, et la gestion des conteneurs, fournissant ainsi un environnement cohérent et efficace pour les applications distribuées. Kubernetes simplifie la gestion des ressources et garantit la disponibilité des applications, facilitant ainsi le déploiement sur des environnements cloud et hybrides.



#### Github Actions

GitHub Actions est un service d'automatisation intégré à GitHub, permettant aux développeurs de définir des workflows personnalisés pour automatiser les processus de build, de test et de déploiement directement depuis leur référentiel. Utilisant des fichiers YAML, cette fonctionnalité simplifie l'automatisation des tâches de développement.



GitHub Actions

### 4.1.3 Cloud

#### AWS

AWS (Amazon Web Services) est une plateforme de cloud computing proposant une variété de services pay-as-you-go pour le stockage, la computation, la base de données, et plus encore. Elle permet aux entreprises de déployer et gérer des applications de manière scalable et flexible sans investissements initiaux importants.



#### Amazon EKS

Amazon EKS (Elastic Kubernetes Service) est un service managé par AWS qui simplifie le déploiement, la gestion et l'évolutivité des applications conteneurisées en utilisant Kubernetes. EKS élimine la complexité opérationnelle liée à la gestion d'un cluster Kubernetes, offrant ainsi une solution scalable et flexible pour l'orchestration de conteneurs sur AWS.



#### Terraform

Outil d'infrastructure as code qui permet de définir, provisionner et gérer l'infrastructure cloud de manière efficace et reproductible.

Nous utilisons Terraform pour définir et provisionner notre infrastructure cloud sur AWS, notamment pour la configuration du cluster EKS. Dans notre configuration Terraform, nous spécifions des éléments tels VPC et les subnets.



## 4.2 Conteneurs docker

Pour assurer la portabilité, l'intégration facile et le déploiement efficace de nos services, nous avons opté pour l'utilisation de conteneurs Docker. Les conteneurs offrent un environnement isolé et standardisé, garantissant une cohérence entre les différentes étapes du cycle de vie de notre application.

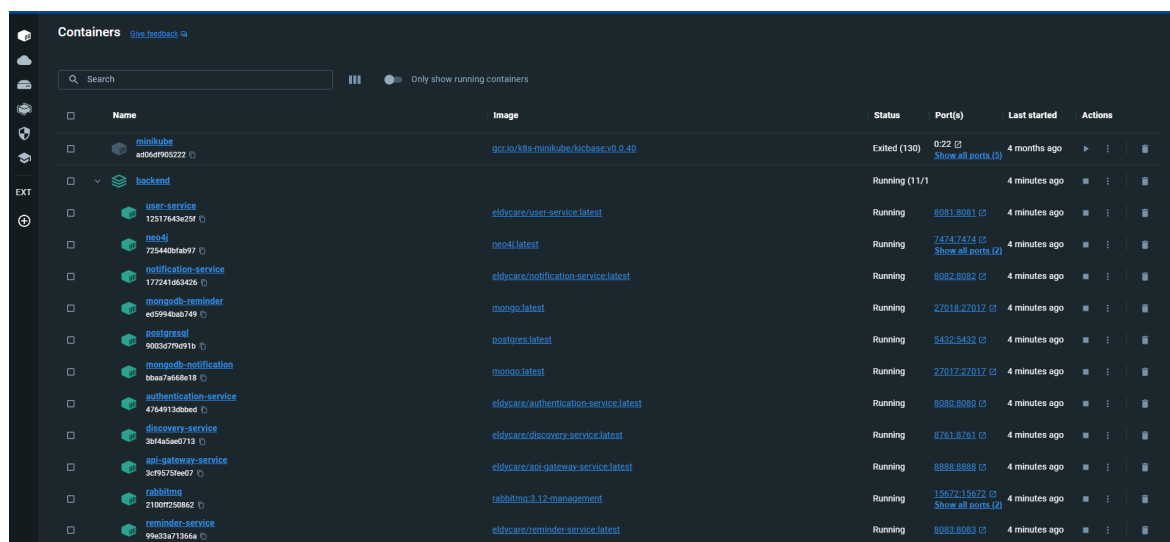


FIGURE 4.1 – Docker Containers

Nous avons utilisé des images DockerHub pour des bases de données telles que PostgreSQL, MongoDB et Neo4j, ainsi que pour message broker RabbitMQ. Cette approche nous permet de bénéficier de la facilité de gestion des dépendances et de la reproductibilité des environnements, éléments cruciaux dans un environnement de développement moderne.

Aussi chaque service que nous développons est conteneurisé à l'aide de Dockerfiles spécifiques. Ces Dockerfiles définissent l'environnement cible de notre service, garantissant une configuration homogène à chaque étape de notre pipeline de développement.

## 4.3 Service de découverte eureka

Nous avons mis en place un service de découverte Eureka pour faciliter la gestion dynamique des adresses IP des services au sein de notre architecture microservices. Eureka offre un moyen efficace de localiser et d'interagir avec les différents services, améliorant ainsi la résilience et la flexibilité de notre système.

Application	AMIs	Availability Zones	Status
AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">authentication-service-65b649765b-cw8mg:authentication-service:8080</a>
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">api-gateway-service-579f9cc545-r6mjw:gateway-service:8888</a>
NOTIFICATION-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">notification-service-bb4f658b9-5qjrb:notification-service:8082</a>
REMINDER-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">reminder-service-86b9865849-9dpwz:reminder-service:8083</a>
USER-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">user-service-96678dc88-m5t69:user-service:8081</a>

FIGURE 4.2 – Instances enregistrées avec Eureka

## 4.4 Service RabbitMQ

Notre architecture orientée événements repose également sur l'utilisation d'un service de messagerie RabbitMQ. Il joue un rôle central dans la communication asynchrone entre nos services, permettant un couplage plus lâche et une meilleure extensibilité. RabbitMQ contribue à la création d'un système robuste et réactif, capable de gérer efficacement les charges de travail variables.

Voici quelques cas d'utilisation de la communication entre les services à travers RabbitMQ :

- Filtrage des Services par Authentification dans le Service API Gateway : Le service API Gateway filtre les services par authentification, ce qui nécessite la validation du token JWT par le service d'authentification. C'est là que l'utilisation d'un message broker devient nécessaire.

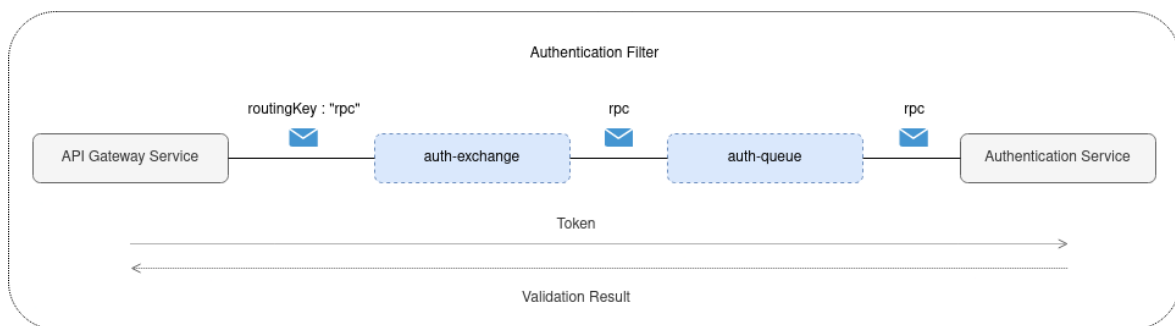


FIGURE 4.3 – Filtre d'authentification

- Création/Enregistrement d'un Nouvel Utilisateur : Lors de la création ou de l'enregistrement d'un nouvel utilisateur, les informations de l'utilisateur doivent également être sauvegardées dans le service utilisateur (user-service). Cela permet, par la suite, d'avoir la possibilité d'ajouter des personnes apparentées dans le cas d'une personne âgée ou d'ajouter cette dernière en tant que relative d'une personne âgée.

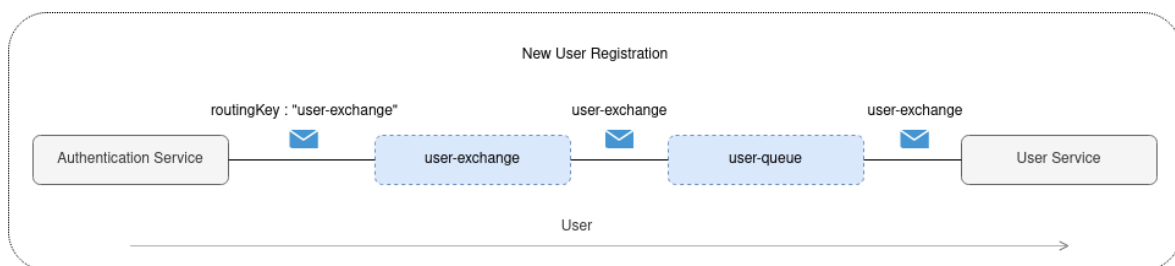


FIGURE 4.4 – Inscription d'un nouvel utilisateur



## 4.5 Utilisation de WebSockets

Après la mise en place du service RabbitMQ pour la communication asynchrone, nous avons également intégré le mécanisme de WebSockets pour la diffusion des alertes et des rappels. Cette approche a permis une communication en temps réel avec les personnes âgées et leurs proches.

### WebSockets dans le Service de Notification

Le service de notification utilise WebSockets pour envoyer des alertes en temps réel aux proches et aux personnes concernées lors de la détection d'une anomalie par la montre intelligente. Voici comment cela est réalisé :

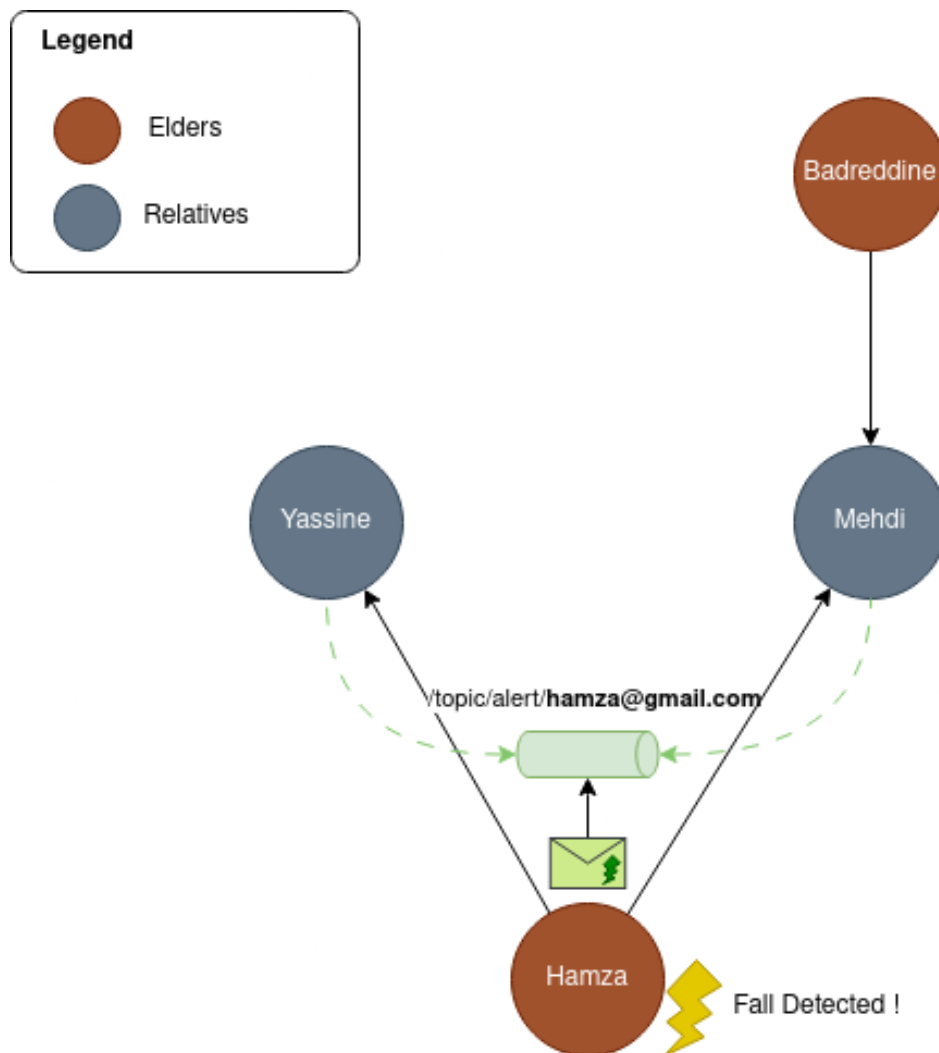


FIGURE 4.5 – Mécanisme de diffusion des alertes

### WebSockets dans le Service de Rappel

Le service de rappel utilise également WebSockets pour permettre aux proches de planifier des rappels personnalisés pour la personne âgée. Voici un aperçu de son utilisation :

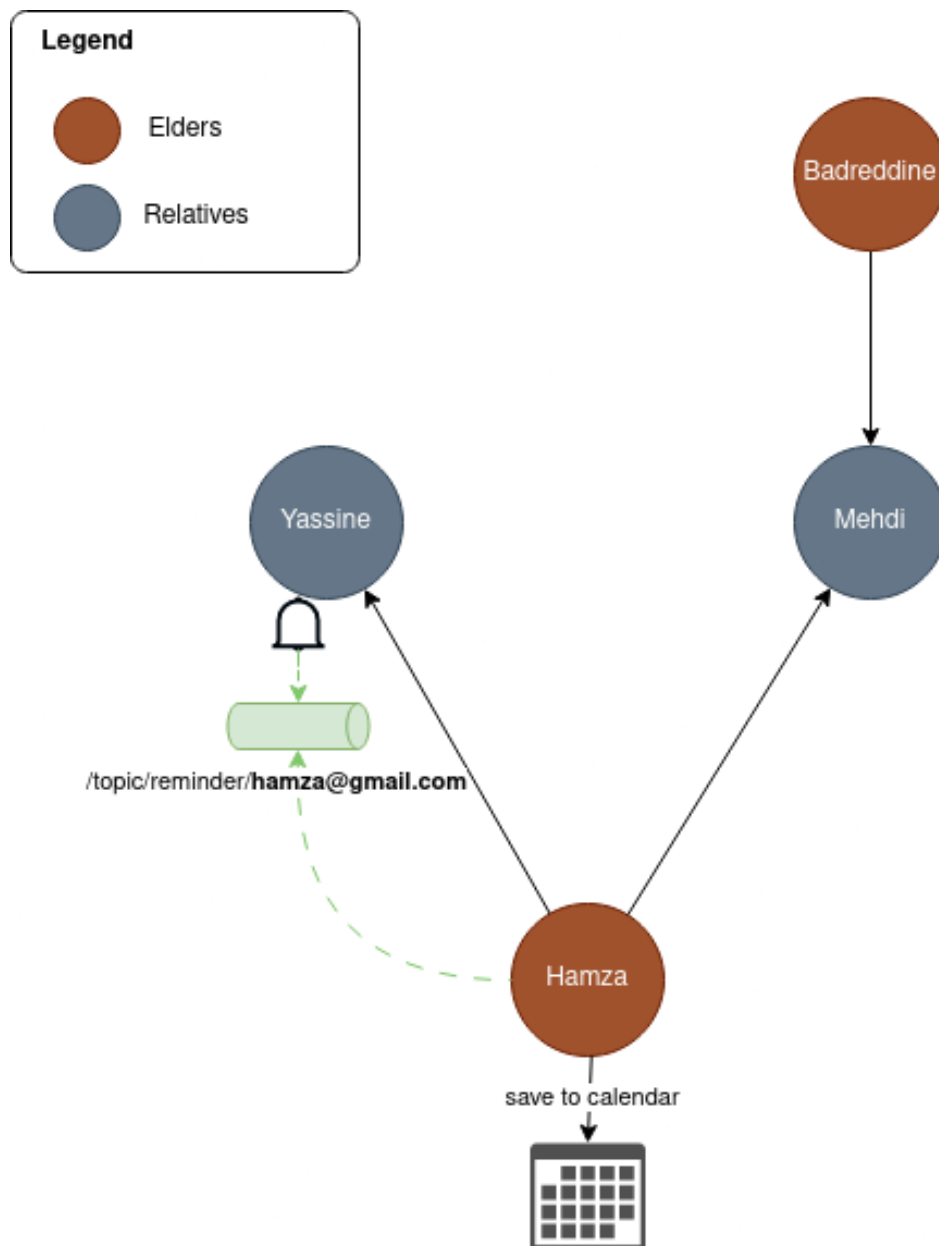


FIGURE 4.6 – Mécanisme de diffusion des rappels

Cette utilisation des WebSockets améliore considérablement la réactivité et la communication en temps réel entre le backend et l'application mobile, offrant ainsi une expérience utilisateur plus dynamique.

## 4.6 Pipeline CI/CD

Nous avons établi un pipeline CI/CD pour automatiser le processus d'intégration et de déploiement de nos services. Cette approche accélère le cycle de développement et assure la disponibilité continue de nouvelles fonctionnalités.

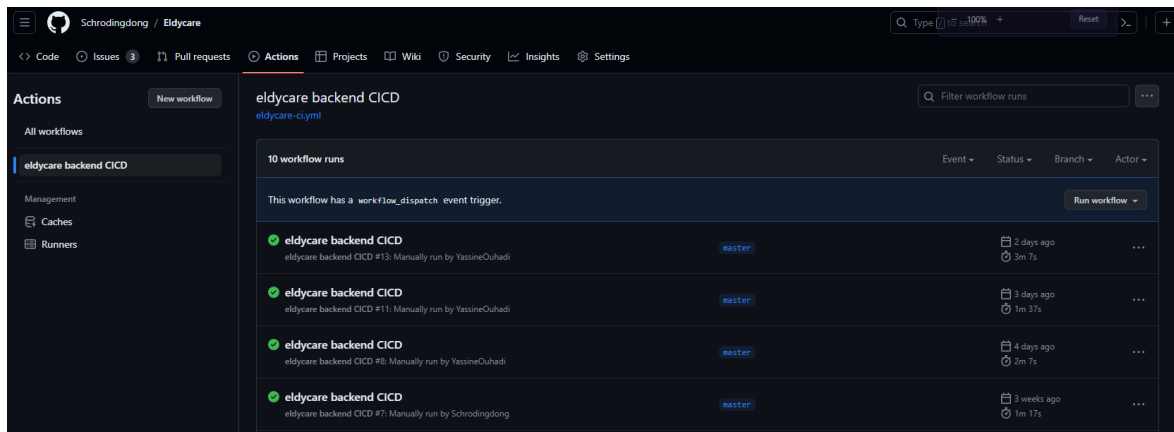


FIGURE 4.7 – Github Jobs

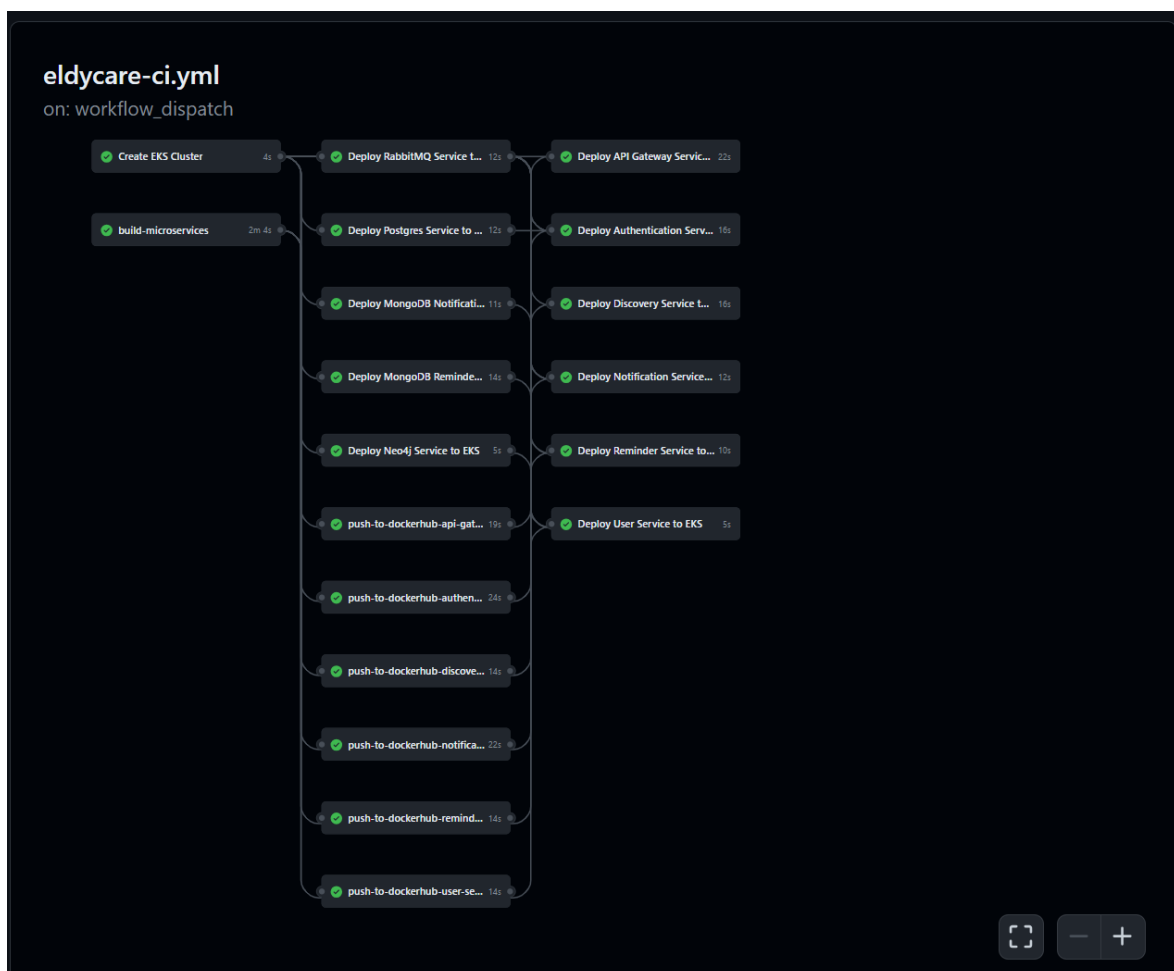


FIGURE 4.8 – CICD Pipeline

Pour l'implémentation de ce dernier, nous avons bénéficié des workflows de GitHub Actions pour construire des images Docker et les déployer dans DockerHub pour chaque service après chaque push dans la branche master. Cela facilite ensuite le déploiement vers le cluster EKS en AWS.

## 4.7 Intégration avec DockerHub

Notre pipeline CI/CD est intégré avec Dockerhub, poussant l'ensemble des artifacts des services à être enregistré et accessible à toute personne.

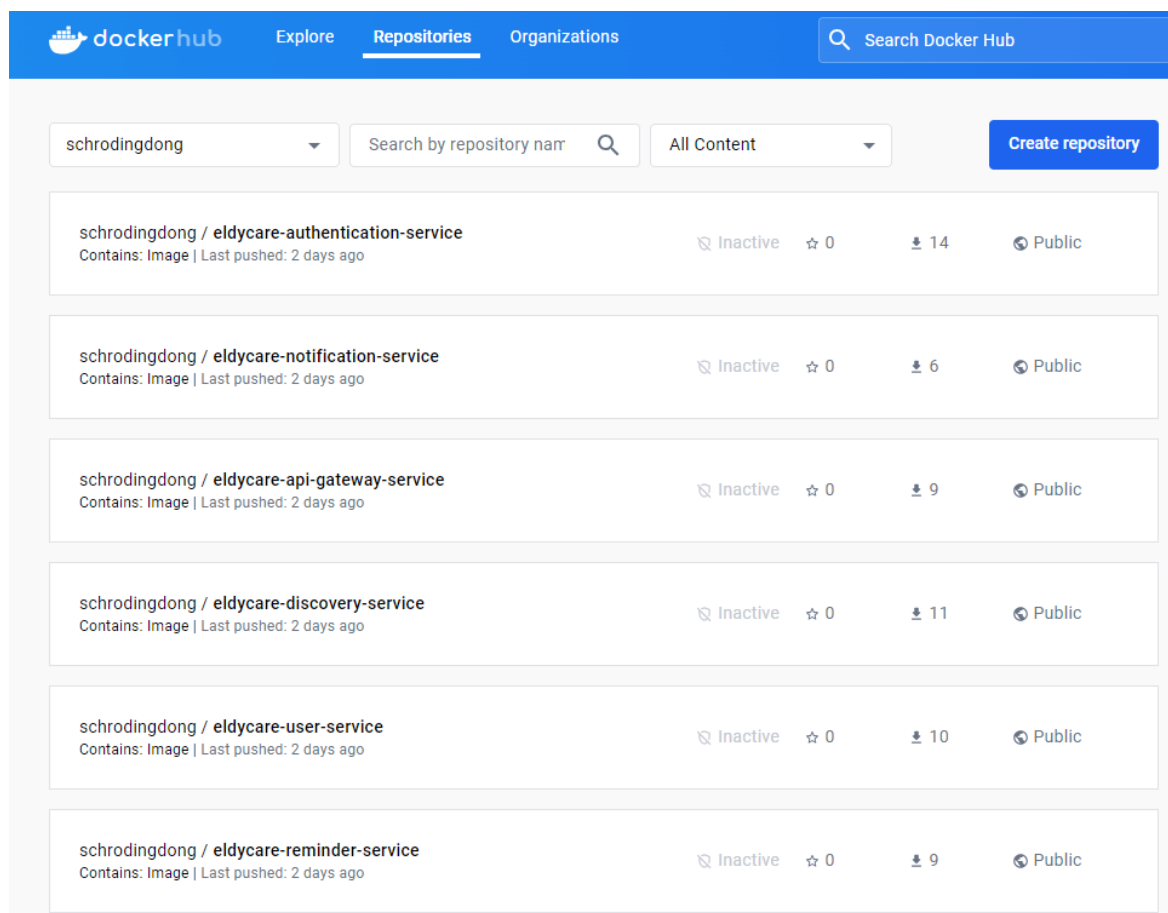
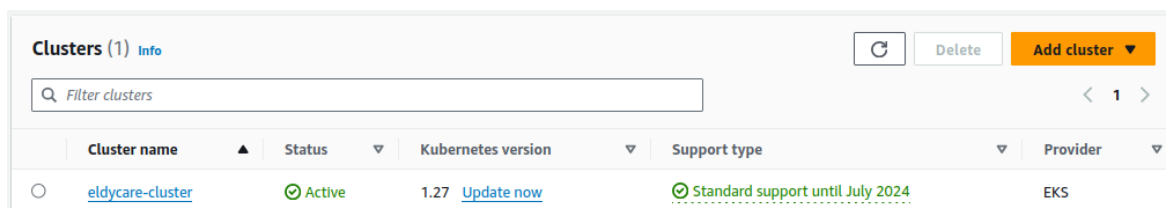


FIGURE 4.9 – Repo DockerHub

## 4.8 Déploiement AWS

Dans le cadre de notre stratégie de déploiement, nous avons choisi Amazon Web Services (AWS) comme plateforme cloud. Nous avons utilisé des outils tels qu'Elastic Kubernetes Service (EKS) pour orchestrer et déployer nos services sur des clusters Kubernetes gérés par AWS. Cela nous permet d'exploiter l'évolutivité et la fiabilité d'AWS pour prendre en charge nos charges de travail.

Nous avons créé notre cluster EKS dans la région AWS us-east-1, utilisant des outils tels qu'EKSctl et Terraform. L'utilisation de l'Infrastructure as Code avec Terraform nous a permis de décrire notre infrastructure de manière reproductible.



Cluster name	Status	Kubernetes version	Support type	Provider
<a href="#">eldycare-cluster</a>	Active	1.27 <a href="#">Update now</a>	Standard support until July 2024	EKS

FIGURE 4.10 – EKS Cluster

Chaque service que nous déployons est décrit dans un fichier YAML, suivant les meilleures pratiques de Kubernetes. Ces fichiers YAML décrivent les services et les déploiements nécessaires, garantissant ainsi une gestion efficace et reproductible de nos conteneurs Kubernetes dans EKS.

De plus, pour optimiser les coûts, nous avons configuré le déploiement de nos services avec une réplication minimale, fixant le nombre de réplicas à 1 pour chaque service. Cette approche nous permet de tirer parti de l'offre de Free Tier d'AWS.

NAME	READY	STATUS	RESTARTS	AGE
api-gateway-service-579f9cc545-r6mjw	1/1	Running	0	11h
authentication-service-65b649765b-cw8mg	1/1	Running	0	11h
discovery-service-67b865dfb9-vq2jb	1/1	Running	0	11h
mongodb-notification-service-5df8cf58cd-bpxfc	1/1	Running	0	11h
mongodb-reminder-service-587b7c7848-nqxln	1/1	Running	0	11h
neo4j-service-7649dc56f-x68rn	0/1	CrashLoopBackOff	135 (3m28s ago)	11h
notification-service-bb4f658b9-5qjrb	1/1	Running	0	11h
postgres-service-7cb97c697f-2fb2b	1/1	Running	0	11h
rabbitmq-service-86859474fb-x5knc	1/1	Running	0	11h
reminder-service-86b9865849-9dpwz	1/1	Running	0	7h20m
user-service-96678dc88-m5t69	1/1	Running	0	7h20m

FIGURE 4.11 – la liste des pods

Et voici la liste des services, les IPs externes et les ports pour chaque service :

api-gateway-service 8888:30943/TCP	LoadBalancer	10.100.115.20	ad3d4a38eb8a846409cd1a050d1b632a-875694907.us-east-1.elb.amazonaws.com
authentication-service 8084:31109/TCP	LoadBalancer	10.100.107.214	aaba7d257e0794c77ba0eee0795a137d-1527233917.us-east-1.elb.amazonaws.com
discovery-service 8761:30590/TCP	LoadBalancer	10.100.35.88	a91e37913a5a1461fb00bc3feb3cd31c-295397706.us-east-1.elb.amazonaws.com
kubernetes 443/TCP	ClusterIP	10.100.0.1	<none>
mongodb-notification-service 27017:30748/TCP	LoadBalancer	10.100.221.153	a0780ad88fa304f1ab0d4f21491604b8-570440562.us-east-1.elb.amazonaws.com
mongodb-reminder-service 27018:32144/TCP	LoadBalancer	10.100.95.35	acbdff77d4ad074c67a2e681be5ef669b-1460101538.us-east-1.elb.amazonaws.com
neo4j-service 7474:31259/TCP, 7687:31573/TCP	LoadBalancer	10.100.94.206	a66734dd0ce544ababc4a4ca4bf9e1da-57158815.us-east-1.elb.amazonaws.com
notification-service 8082:31783/TCP	LoadBalancer	10.100.117.218	ae7ba6d9644574962b055a7f46863349-1281813327.us-east-1.elb.amazonaws.com
postgres-service 5432:30760/TCP	LoadBalancer	10.100.114.252	abffafa79091a4550b83dbef255bf96a-1207429801.us-east-1.elb.amazonaws.com
rabbitmq-service 5672:32183/TCP, 15672:32701/TCP	LoadBalancer	10.100.152.86	a8333446795c24101be92667f2131ff6-1778000983.us-east-1.elb.amazonaws.com
reminder-service 8083:31757/TCP	LoadBalancer	10.100.114.30	a9ba17b44356247f6bd8ed636ab9afb-545525765.us-east-1.elb.amazonaws.com
user-service 8081:32430/TCP	LoadBalancer	10.100.103.63	ac85d210de90b4b14ade0c7c9b430647-1284120398.us-east-1.elb.amazonaws.com

FIGURE 4.12 – la liste des services

Nous effectuons ces déploiements bien sûr de manière automatisée grâce à notre pipeline GitHub Actions. Nous l'avons modifié pour déployer nos images DockerHub vers des pods dans le cluster EKS.

## 4.9 Conclusion

En conclusion, ces choix technologiques visent à garantir une architecture souple, évolutive et résiliente, prête à répondre aux défis d'un environnement informatique moderne.

## Conclusion Finale

---

En conclusion, notre projet Eldycare représente une avancée significative dans le domaine de la surveillance des personnes âgées. Grâce à notre application mobile, nous avons réussi à fournir une solution technologique qui permet la détection des chutes et l'envoi d'alertes en temps réel, tout en offrant une interface conviviale pour les utilisateurs.

Nous avons suivi une approche méthodique tout au long du processus de développement, en utilisant des outils et des technologies modernes tels que les microservices, les conteneurs Docker et le déploiement sur le cloud. Cela nous a permis de créer une application robuste, évolutive et facilement maintenable.

Cependant, il reste encore des axes d'amélioration possibles pour notre application Eldycare. Par exemple, nous pourrions intégrer de nouvelles fonctionnalités, telles que la détection d'autres situations d'urgence, l'analyse des données de santé ou encore l'intégration de dispositifs portables.

Aussi parmi les perspectives d'amélioration, nous envisageons une optimisation de l'architecture de déploiement sur AWS. Une transition vers des services avancés tels que AWS CloudWatch pour la gestion des logs, l'utilisation de la base de données DynamoDB pour le stockage de données, ainsi que l'intégration d'autres services AWS, permettrait d'accroître la performance, la fiabilité et la gestion opérationnelle de notre solution.

En résumé, notre projet Eldycare a été une expérience enrichissante qui nous a permis de mettre en pratique nos connaissances en génie logiciel et de développer une solution concrète pour répondre aux besoins des personnes âgées. Nous espérons que notre travail contribuera à améliorer la qualité de vie des personnes âgées et à assurer leur sécurité et leur bien-être.



# Bibliographie

- [1] Public Health Agency of Canada. Government of canada, Jun 2022.
- [2] Thiago de Quadros, Andre Eugenio Lazzaretti, and Fábio Kürt Schneider. A movement decomposition and machine learning-based fall detection system using wrist wearable device. *IEEE Sensors Journal*, 18(12) :5082–5089, 2018.
- [3] Usabilis. Material design, l’ui selon google!, April 2018.
- [4] Robin Šín, Ivana Vodehnalova, Denisa Ralbovska, Denisa Struncova, and Lenka Cechurova. Out-of-hospital cardiac arrest in the pilsen region in 2018. *Biomedical Papers*, 165, 01 2020.