**Table des matières**

**Remerciements**

**Résumé**

**Abstract**

**Introduction Générale**

* Contexte
* Problématique
* Objectifs du projet
* Méthodologie adoptée
* Organisation du mémoire

# Chapitre 1 : Étude bibliographique et état de l’art

## Les chutes chez les personnes sensibles : enjeux et conséquences

Les chutes représentent un problème de santé publique majeur, en particulier chez les personnes sensibles, notamment les personnes âgées, les patients atteints de maladies chroniques, ou encore les personnes en situation de handicap. Elles sont l'une des principales causes de blessures graves, de perte d'autonomie et parfois même de décès prématuré.

Selon l'Organisation Mondiale de la Santé (OMS), chaque année, environ 28 à 35 % des personnes âgées de 65 ans et plus chutent au moins une fois. Ce taux augmente avec l’âge et les facteurs de risque, atteignant plus de 40 % chez les plus de 80 ans. Ces chutes peuvent entraîner des conséquences physiques (fractures, traumatismes crâniens), mais aussi psychologiques (peur de tomber à nouveau, isolement social).

**Facteurs de risque des chutes**

Les causes des chutes sont souvent multifactorielles :

* **Physiologiques** : faiblesse musculaire, troubles de l’équilibre, baisse de la vision ou de l’audition.
* **Pathologiques** : maladies neurologiques (Parkinson, AVC), troubles cognitifs, hypotension orthostatique.
* **Environnementaux** : sols glissants, mauvaise éclairage, absence de rampes ou de barres d’appui.
* **Médicamenteux** : prise de certains traitements comme les sédatifs, antihypertenseurs, ou diurétiques.

**Conséquences sociales et économiques**

Outre l’impact sanitaire, les chutes ont des conséquences économiques importantes : hospitalisations, soins de longue durée, rééducation, voire aménagement du domicile ou placement en institution. Le coût humain est également élevé, affectant la qualité de vie de la personne et de ses proches.

**Importance de la détection rapide**

La rapidité de prise en charge après une chute est cruciale pour minimiser les séquelles. Une détection tardive, surtout chez les personnes vivant seules, peut aggraver la situation : hypothermie, déshydratation, complications médicales. C’est pourquoi la mise en place de systèmes intelligents de détection de chute constitue aujourd’hui un enjeu de prévention, d’intervention rapide, et de protection des personnes vulnérables.

## Technologies existantes de détection de chute

La détection automatique des chutes est un domaine en pleine évolution, tirant parti des avancées en électronique embarquée, capteurs intelligents et intelligence artificielle. Plusieurs solutions ont vu le jour, avec des approches différentes en matière de fiabilité, d'autonomie et d'intégration dans le quotidien des personnes sensibles.

**1.2.1 Systèmes mécaniques et manuels**

Les premières approches utilisaient des boutons d’alerte manuelle, portés en pendentif ou en bracelet. Lorsqu’une chute se produit, la personne doit appuyer sur le bouton pour alerter un proche ou les secours.  
Cependant, ces dispositifs dépendent entièrement de la capacité de la personne à réagir, ce qui limite leur efficacité en cas d’inconscience ou de désorientation.

**1.2.2 Capteurs embarqués**

Les dispositifs modernes intègrent des capteurs de mouvement (accéléromètre, gyroscope) capables de détecter les variations brusques d'accélération et d’orientation caractéristiques d’une chute :

* **Accéléromètres** : mesurent les accélérations linéaires sur les axes X, Y et Z.
* **Gyroscopes** : détectent les rotations du corps.
* **Magnétomètres** : parfois utilisés pour compléter la mesure de position.

Certains dispositifs sont intégrés dans des montres intelligentes, bracelets connectés ou ceintures intelligentes, capables d’envoyer une alerte automatiquement.

**1.2.3 Vision par ordinateur**

Des recherches plus récentes s’intéressent à l’utilisation de caméras et d’algorithmes de vision par ordinateur pour détecter une chute.  
Bien que cette approche puisse offrir une bonne précision, elle présente plusieurs limitations :

* Respect de la vie privée,
* Nécessité d’un environnement contrôlé (pièce fixe, bonne luminosité),
* Coût d’équipement élevé.

**1.2.4 Intelligence artificielle et apprentissage automatique**

Certaines solutions exploitent le machine learning ou le deep learning pour reconnaître des motifs spécifiques liés aux chutes à partir de données de capteurs.  
Ces techniques permettent d’augmenter la fiabilité du système, mais demandent une phase d’apprentissage importante, des ressources de calcul, et une base de données d’entraînement.

**1.2.5 Limites des systèmes existants**

Malgré leur diversité, les solutions actuelles font face à plusieurs défis :

* **Faux positifs et faux négatifs**,
* **Autonomie limitée** (problème de batterie),
* **Portabilité et confort d’utilisation**,
* **Prix élevé pour les utilisateurs à faible revenu**.

Ces limitations motivent le développement de nouvelles solutions simples, efficaces, peu coûteuses et adaptées à un contexte local, comme celui proposé dans le cadre de ce projet.

## Les microcontrôleurs et objets connectés dans la santé

L’avènement des **technologies embarquées** et de l’**Internet des Objets (IoT)** a considérablement transformé le secteur de la santé. Grâce aux microcontrôleurs et aux capteurs connectés, il est désormais possible de concevoir des dispositifs intelligents capables de surveiller, analyser et transmettre des données médicales en temps réel.

**1.3.1 Rôle des microcontrôleurs dans les dispositifs médicaux**

Les microcontrôleurs, comme l’ESP32 utilisé dans ce projet, sont de petits systèmes informatiques autonomes, capables de :

* Collecter des données à partir de capteurs,
* Exécuter des traitements légers en local,
* Gérer des communications sans fil (Wi-Fi, Bluetooth, GSM),
* Réagir à des événements en temps réel.

Leur faible coût, leur consommation énergétique réduite et leur flexibilité en font un choix idéal pour les projets de santé connectée.

**1.3.2 Applications des objets connectés en santé**

Les objets connectés sont utilisés dans de nombreuses applications médicales :

* **Surveillance cardiaque** (fréquence, pression artérielle),
* **Détection de mouvements anormaux** (tremblements, chutes),
* **Suivi de l’activité physique**,
* **Aide aux personnes âgées ou malades à domicile**,
* **Téléassistance et alertes médicales**.

Dans le contexte de la détection de chute, les objets connectés permettent non seulement de détecter l’accident, mais aussi de **réagir rapidement** en **alertant les proches** ou **les services de secours**, tout en **transmettant la position GPS** de la victime.

**1.3.3 Avantages des solutions embarquées dans la santé**

* **Accessibilité** : technologies économiques et adaptables.
* **Portabilité** : systèmes légers, portés sur le corps ou intégrés aux vêtements.
* **Réactivité** : détection et envoi d’alerte quasi instantanés.
* **Automatisation** : pas besoin d'intervention humaine pour déclencher l’alerte.

**1.3.4 Limites et défis**

Malgré leurs nombreux avantages, l’utilisation des objets connectés en santé présente aussi des défis à relever :

* **Fiabilité des capteurs**,
* **Sécurité des données** (confidentialité des informations personnelles),
* **Autonomie énergétique**,
* **Robustesse dans des environnements variés**.

Ces éléments sont à prendre en compte lors de la conception d’un système fiable, sécurisé et réellement utile pour les utilisateurs finaux.

## Choix technologiques du projet : motivations

Le choix des composants et des technologies utilisés dans ce projet repose sur plusieurs critères essentiels : fiabilité, coût, simplicité d’intégration, consommation énergétique, et accessibilité des composants sur le marché local. L’objectif est de concevoir un système efficace, réactif, mais également abordable et facile à reproduire.

**1.4.1 Microcontrôleur ESP32**

Le ESP32 a été choisi comme microcontrôleur principal pour ses nombreuses fonctionnalités :

* Double cœur et bonne capacité de traitement pour gérer plusieurs capteurs.
* Connectivité Wi-Fi et Bluetooth intégrée.
* Faible consommation d’énergie, idéale pour les systèmes portables.
* Large communauté et documentation riche.

Un second ESP32 est également utilisé côté "suivi", pour recevoir les alertes et déclencher un signal sonore (buzzer) en cas de chute détectée.

**1.4.2 Capteur MPU6050**

Le MPU6050 combine un accéléromètre et un gyroscope 3 axes, permettant de détecter :

* Les mouvements brusques,
* Les chocs ou pertes d’équilibre,
* Les changements de position caractéristiques d'une chute.

Ce capteur est petit, précis, peu coûteux, et facilement intégrable via une communication I2C.

**1.4.3 Module GPS NEO-6M**

Le NEO-6M est utilisé pour déterminer la position géographique de la personne en cas de chute. Il a été choisi pour :

* Sa précision,
* Son faible coût,
* Sa compatibilité avec l’ESP32.

Il permet de transmettre les coordonnées GPS exactes lors d’une alerte.

**1.4.4 Module GSM SIM800L**

Pour l’envoi d’alertes par SMS, le module SIM800L est utilisé. Il permet de :

* Transmettre un message contenant la position GPS,
* Fonctionner dans les zones sans connexion Internet,
* Être alimenté par batterie avec une gestion énergétique optimisée.

Ce choix garantit que les alertes peuvent être envoyées même en absence de Wi-Fi ou de réseau mobile de données.

**1.4.5 Boutons de contrôle**

Deux boutons physiques sont intégrés :

* *Un bouton d’urgence* : permet à l’utilisateur d’envoyer manuellement une alerte.
* *Un bouton d’annulation de fausse alerte* : permet d’annuler un SMS si la détection est erronée.

Ces boutons offrent un contrôle direct et simple au porteur du dispositif.

**1.4.6 Application mobile de suivi**

Une **application mobile** est développée pour permettre :

* Le suivi en temps réel de la position de la personne,
* L’affichage des alertes de chute,
* Une interface simple pour les aidants.

Cette application complète le dispositif matériel pour offrir une **solution complète et interactive**.

# Chapitre 2 : Conception du système de détection de chute

## Architecture générale du système

Le système de détection de chute développé dans ce projet repose sur une architecture distribuée, combinant capteurs embarqués, modules de communication, une application backend et une application mobile. Cette architecture permet une détection automatique de chute, un envoi d'alerte par SMS, un suivi en temps réel, ainsi que la centralisation de l’historique des événements.

**2.1.1 Composants principaux du système**

L’architecture se compose de trois grands blocs :

**a) Terminal embarqué sur la personne**

Ce module, porté par la personne sensible, est responsable de la détection des chutes :

* **ESP32 principal** : microcontrôleur chargé de la gestion des capteurs et modules.
* **MPU6050** : capteur de mouvement pour détecter les chutes.
* **NEO-6M** : module GPS pour localiser la personne.
* **SIM800L** : pour l’envoi de SMS en cas d’alerte.
* **Boutons physiques** : pour déclencher ou annuler manuellement une alerte.
* **Connexion Wi-Fi ou GPRS** : pour envoyer les données vers le backend.

**b) Serveur Backend**

Le serveur backend centralise les données et permet :

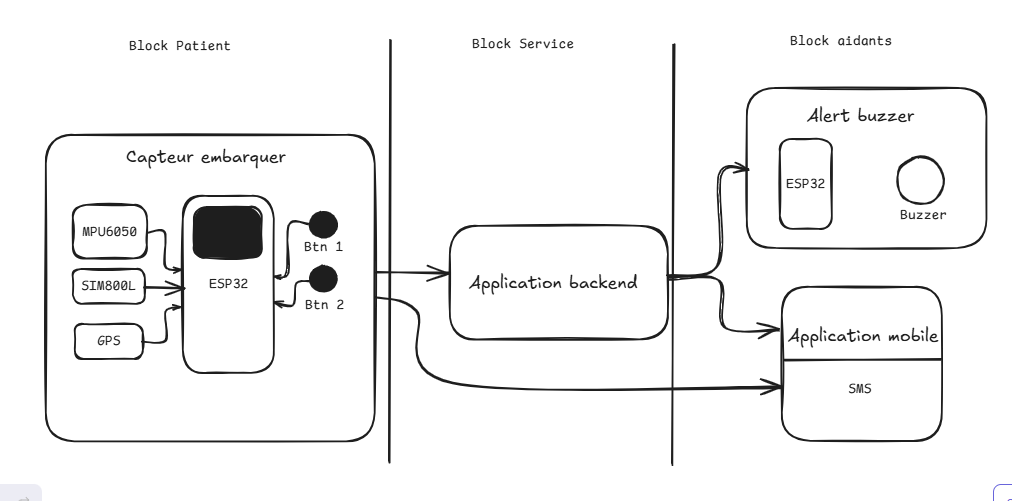
* L’enregistrement des alertes dans une base de données (date, position, état, etc.),
* La mise à disposition des données GPS en temps réel à l’application mobile,
* L’envoi de notifications aux aidants via l’API.

**c) Application mobile de suivi**

L’application permet aux aidants de :

* Visualiser en temps réel la position GPS de la personne suivie,
* Recevoir les alertes instantanément,
* Consulter l’historique des chutes et des alertes,
* Être notifiés même à distance.

**2.1.2 Schéma simplifié de l’architecture**

****

**2.1.3 Fonctionnement global**

1. **Surveillance** : L’ESP32 collecte les données de mouvement.
2. **Détection de chute** : En cas d’anomalie, l’ESP32 récupère la position GPS.
3. **Transmission** :
   * Envoie un SMS d’alerte contenant la position.
   * Envoie également la position et l’alerte au serveur backend via HTTP ou MQTT.
4. **Traitement** :
   * Le backend enregistre l’alerte dans une base de données.
   * Il transmet les informations à l’application mobile en temps réel.
5. **Visualisation** : L’application affiche la position actuelle et l’historique des alertes.

Cette architecture assure une communication fluide et fiable, tout en garantissant une traçabilité des événements.

# Chapitre 3 : Communication entre les modules et développement logiciel

## Programmation des microcontrôleurs (ESP32)

Avant d’entamer le développement du projet de détection de chute, il était essentiel de maîtriser le fonctionnement du microcontrôleur principal utilisé : l’**ESP32**. Cette phase d’apprentissage a débuté par des exercices simples permettant de se familiariser avec les entrées/sorties numériques, ainsi que les fonctionnalités réseau de la carte.

* **Programme de base : clignotement de LED (Blink)**

Le premier test réalisé consiste à allumer et éteindre une LED à intervalles réguliers à l’aide d’un programme simple, appelé "Blink".

**Matériel utilisé :**

* 1 LED bleue
* 1 résistance de 200 ohms
* Connexion entre la broche **GPIO 21** de l’ESP32 (sortie digitale) et la **cathode** de la LED
* La **masse** (GND) du microcontrôleur connectée à l’anode via la résistance

*(Insérer ici un schéma de câblage si disponible)*

**Fonctionnement du programme :**  
Le programme, écrit sur l’IDE Arduino, utilise une boucle (loop()) pour envoyer un signal haut (HIGH) sur la broche GPIO 21, ce qui allume la LED, puis un signal bas (LOW) pour l’éteindre, avec un délai de 1000 ms entre chaque changement d’état.

**Outils utilisés :**

* IDE Arduino pour l’édition, la compilation et le téléversement du code
* Bibliothèque par défaut incluse pour la gestion des ports GPIO
* Ce test simple permet de valider que la carte fonctionne correctement et que le téléversement via USB est opérationnel.
* **Connexion à un réseau Wi-Fi**

L’ESP32 possède un module Wi-Fi intégré, ce qui en fait un composant adapté aux projets d’**IoT (Internet of Things)**. La connexion au réseau est donc une étape fondamentale pour assurer la communication avec des services distants, tels qu’un serveur ou une application mobile.

**Bibliothèque utilisée:**  
La bibliothèque **WiFi.h**, incluse dans les outils ESP32 pour Arduino, permet de gérer les connexions sans fil de façon simple et efficace.

**Procédure :**

* Installation de la bibliothèque ESP32 dans l’IDE Arduino (via le gestionnaire de cartes)
* Utilisation d’un exemple de code fourni (template) pour tester la connexion
* Modification du code pour inclure les identifiants du réseau Wi-Fi personnel
* Téléversement du programme sur la carte
* Vérification de la connexion à partir du moniteur série (affichage de l’adresse IP obtenue)

Une fois la carte connectée à un réseau Wi-Fi, elle peut envoyer ou recevoir des données à distance, ce qui ouvre la voie à l’intégration de la détection de chute avec un service de notification ou une application mobile.

Ces premiers exercices m'ont permis de mieux comprendre :

* la **manipulation des broches GPIO**, notamment en sortie numérique pour le contrôle de composants simples comme une LED ;
* le fonctionnement du **module Wi-Fi** intégré à l’ESP32 pour établir une connexion réseau ;
* ainsi que l'utilisation de l'**environnement de développement Arduino IDE**, aussi bien pour la rédaction du code que pour le téléversement et le débogage via le moniteur série.

Cette base de connaissances a été indispensable pour la suite du projet, notamment pour l’interfaçage avec les capteurs, les modules de communication et le développement logiciel embarqué.

## Simulation d’un bouton avec l’ESP32

Dans le cadre du projet, l’utilisation de **boutons poussoirs** est indispensable, notamment pour déclencher des alertes manuellement ou valider certaines actions côté utilisateur. Avant d’intégrer cette fonctionnalité au système complet, il a été nécessaire de tester et de comprendre le fonctionnement d’un bouton connecté à l’ESP32.

* **Matériel utilisé**
* 1 bouton poussoir
* 1 LED bleue
* 1 résistance de **10 kΩ** (pour le **pull-down**)
* 1 résistance de **220 Ω** (pour la LED)
* 1 carte ESP32
* Fils de connexion
* Breadboard
* **Principe de fonctionnement**

Le circuit est monté de manière à ce que l'appui sur le bouton envoie un signal **HAUT (HIGH)** sur une broche GPIO de l’ESP32. Par défaut, sans appui, la broche est maintenue à l'état **BAS (LOW)** grâce à la résistance de pull-down de 1 kΩ.

* La LED est branchée sur une autre broche GPIO, comme dans le test précédent.
* L’état de la LED dépend d’une **variable booléenne** dont la valeur change à chaque appui sur le bouton.
* **Code et logique de programmation**

Le programme, téléversé via l’Arduino IDE, réalise les actions suivantes :

1. Lecture de l’état du bouton à chaque cycle de la boucle loop()
2. Inversion de la valeur de la variable booléenne si un appui est détecté (détection de front descendant ou montant)
3. Allumage ou extinction de la LED en fonction de la variable

Ce test permet de valider la **lecture d’entrée digitale** avec un **événement utilisateur physique** (le bouton), et de déclencher une action simple (allumer ou éteindre une LED).

* **Résultat et perspectives**

Après exécution du programme, le système a fonctionné comme prévu : chaque pression sur le bouton alternait l’état de la LED.

Ce test constitue une base fonctionnelle pour l’**intégration de boutons dans le projet final**, notamment pour la **simulation d’un bouton d’urgence**, qui activera à la place une alerte ou une autre fonctionnalité (buzzer, envoi de SMS, etc.).

## Le MPU6050 avec l’ESP32

Le **MPU6050** est un composant clé du projet, car il permet de détecter les mouvements et donc les chutes potentielles. Ce capteur intègre un **accéléromètre** (mesure de l’accélération linéaire sur 3 axes) ainsi qu’un **gyroscope** (mesure de la rotation angulaire), ce qui en fait un choix idéal pour une solution de détection embarquée.

Avant d’exploiter ses données dans une logique de détection de chute, il a été nécessaire de réaliser un test d’interfaçage et de lecture de ses valeurs.

* **Matériel utilisé**
* 1 capteur MPU6050
* 1 carte ESP32
* Câbles de connexion
* **Bibliothèque et environnement**

Pour simplifier l’intégration du capteur, la bibliothèque **Adafruit\_MPU6050.h** a été utilisée. Elle est régulièrement mise à jour, bien documentée, et adaptée à l’environnement Arduino. Elle offre une abstraction simple des fonctions de lecture de données du capteur.

* **Connexion et protocole**

Le MPU6050 communique via le protocole I2C. Il possède deux broches principales pour cela :

* SDA (données) → connectée à GPIO 21 sur l’ESP32
* SCL (horloge) → connectée à GPIO 22 sur l’ESP32  
  (*À noter : tu avais mentionné GPIO 32 dans ton texte, mais GPIO 22 est la broche I2C SCL par défaut sur l’ESP32.*)

L’alimentation se fait en 3.3 V ou 5 V selon la version du module, et la masse (GND) est reliée à celle de l’ESP32.

* **Fonctionnement du programme de test**

Un exemple de programme fourni avec la bibliothèque Adafruit a été utilisé pour tester la communication avec le capteur et lire les valeurs d’accélération. Ce programme réalise les actions suivantes :

1. Initialisation de la communication I2C et du capteur MPU6050
2. Lecture des valeurs d’accélération selon les axes X, Y et Z
3. Affichage en boucle de ces valeurs sur le moniteur série, toutes les 1000 ms

* **Résultat du test**

Le test s’est révélé concluant : à chaque mouvement du capteur, les valeurs d’accélération changent instantanément sur le moniteur série. Cela prouve que :

* le module est bien reconnu par la carte ESP32 ;
* la communication I2C fonctionne correctement ;
* les données brutes d'accélération peuvent être récupérées et utilisées pour détecter des chutes.

## Détection de chute avec le MPU6050

Une fois le capteur **MPU6050** opérationnel et correctement interfacé avec l’ESP32, l’étape suivante consiste à développer une **logique de détection de chute**, fondée sur l’analyse des données d’accélération fournies par l’accéléromètre intégré.

* **Définition de la chute (approche physique)**

Avant d’implémenter une détection logicielle, il est important de comprendre ce que représente une **chute** d’un point de vue physique.

Une chute peut être décrite comme :

une **descente rapide** d’un objet (accélération soudaine vers le bas),

suivie d’un **arrêt brutal** lorsqu’il touche le sol ou un autre obstacle.

Cela correspond donc à :

une **variation rapide de l’accélération** (souvent sur l’axe vertical, Z),

suivie d’un **pic d’impact**, détectable par une valeur d'accélération élevée.

* **Comment simmuler cette definition de chute sur le programme**

L’accéléromètre du MPU6050 mesure les accélérations sur les trois axes : X, Y, et Z. Pour une chute verticale, **l’axe Z** est le plus pertinent, car il est aligné avec la gravité terrestre.

En temps normal, un objet immobile sur une table présente une accélération d’environ **1 g** (9,8 m/s²) sur l’axe Z.

Lors d’une chute suivie d’un impact, cette valeur peut atteindre un **pic brutal**, supérieur à **2 g, 5 g, voire 10 g**, selon la violence de l’impact.

Dans le programme, une **condition est ajoutée** :

Si l’accélération sur l’axe Z dépasse une valeur seuil, ici fixée à **12 g**, alors un message "Chute détectée" est affiché sur le moniteur série.

Cette valeur seuil a été définie de manière empirique, en simulant plusieurs chutes pour observer les valeurs maximales atteintes par le capteur.

* **Résultat**

Après téléversement du programme dans la carte ESP32 :

Le système affiche correctement le message d’alerte "Chute détectée" dans le **moniteur série**, à chaque fois qu’une chute réelle ou simulée est effectuée avec l’appareil.

La détection est **réactive et cohérente** avec les mouvements violents.

Ce test confirme que l’accéléromètre du **MPU6050** peut être utilisé efficacement pour la détection de chute, base fonctionnelle du projet.

## Le SIM800L avec l’ESP32

Le **module SIM800L** joue un rôle central dans la **communication à distance** de l’appareil, notamment parce qu’il est destiné à être porté par une personne pouvant subir une chute. Ce module GSM/GPRS permet plusieurs types de communication :

* **Envoi et réception de SMS**
* **Appels vocaux**
* **Transmission de données via GPRS (data mobile)**

Dans le cadre de ce projet, les fonctionnalités les plus pertinentes sont :

* l’**envoi de SMS** pour alerter les proches du patient,
* et la **transmission de données GPS ou d’état** par le biais de la connexion mobile si nécessaire.

Le test présenté dans cette partie se concentre spécifiquement sur l'utilisation du **SIM800L pour envoyer un message d’alerte par SMS**, en lien avec la détection de chute.

* **L’envoi de SMS**

Le but cette test c’est de pouvoir envoyer une message à partir du composant .

* Principe de fonctionnement

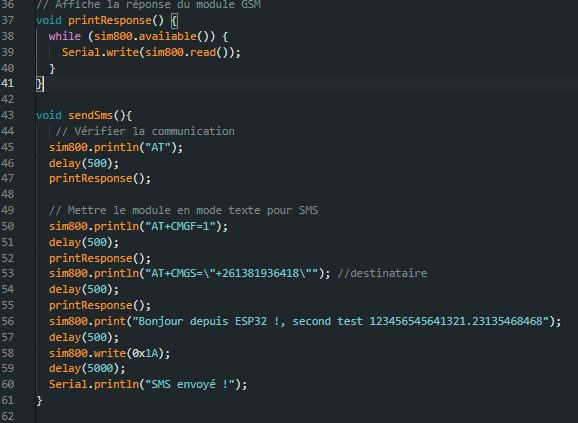
Le **SIM800L** communique avec l’ESP32 via le protocole **UART** (Universal Asynchronous Receiver Transmitter), qui permet une transmission série de données. Pour cela, les connexions suivantes doivent être établies :

* La broche **TX** du SIM800L → sur une **broche RX** de l’ESP32
* La broche **RX** du SIM800L → sur une **broche TX** de l’ESP32
* Bibliothèque utilisée : HardwareSerial.h

L’ESP32 dispose de plusieurs ports UART matériels. Pour activer l’un d’eux (autre que le port série par défaut), on utilise la bibliothèque **HardwareSerial.h**, qui permet de créer une instance UART personnalisée (Serial1, Serial2, etc.).

* **Exemple de code utilisé**

Le programme suivant initialise la communication avec le module, passe en mode texte, configure le numéro du destinataire, et envoie le SMS :



* Résultat

Après avoir téléversé et exécuté le programme :

* Le module SIM800L a bien répondu aux commandes **AT**,
* Le message SMS a été envoyé au numéro défini,
* Le moniteur série affiche "SMS envoyé !" à la fin de l’opération.

Ce test valide l’envoi de SMS à partir de l’ESP32 via le SIM800L, ce qui constitue une **fonction clé du projet** pour la transmission des alertes de chute.

## Le Gps Neo avec l’ESP32

Dans ce projet, le module **GPS Neo-6M** est utilisé pour fournir la **position géographique précise** du porteur du dispositif lors d’une situation d’urgence, comme une chute. Cette information est essentielle pour permettre aux proches ou aux secours de localiser rapidement la personne concernée.

* Principe de localisation par GPS

La **géolocalisation par GPS** fonctionne grâce à un réseau de satellites en orbite autour de la Terre. Le module GPS, en tant que récepteur, capte les signaux envoyés par plusieurs satellites. En mesurant le **temps de transmission** de ces signaux et en les croisant, le module est capable de calculer :

* la **latitude**
* la **longitude**
* parfois aussi l’altitude et la vitesse

Ces **coordonnées GPS** permettent de localiser précisément le dispositif sur la carte.

* Connexion matérielle et communication

Le **module GPS Neo-6M** communique avec l’ESP32 via le protocole **UART**, de la même manière que le module SIM800L. Il faut donc :

* connecter la broche **TX** du GPS à une **broche RX** de l’ESP32
* connecter la broche **RX** du GPS à une **broche TX** de l’ESP32 (même si cette broche est parfois inutilisée, selon la configuration)

L’alimentation du module GPS se fait généralement en **5V** ou **3.3V** selon les modèles, et il est important de connecter aussi la **masse (GND)** au microcontrôleur.

* Bibliothèque utilisée : TinyGPS++

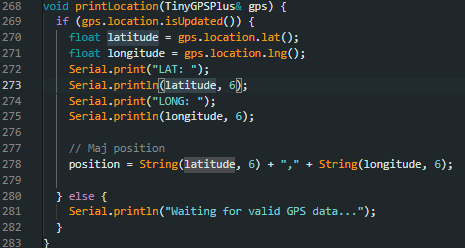
Pour faciliter l’extraction et la lecture des données GPS, la bibliothèque **TinyGPS++** a été utilisée. Elle est :

* **légère**
* **bien documentée**
* et parfaitement adaptée aux microcontrôleurs comme l’ESP32

Elle permet d'accéder facilement aux informations telles que :

* la latitude (gps.location.lat())
* la longitude (gps.location.lng())
* la précision (gps.hdop.hdop()), etc.

Le programme de base consiste à récupérer les coordonnées GPS et à les afficher en temps réel sur le **moniteur série**.



* Résultat du test

Après exécution du programme :

* Le **moniteur série** affiche bien des valeurs de **latitude** et **longitude**.
* Ces valeurs sont des **nombres réels** (décimaux), qui ne sont pas directement interprétables sans une carte.
* Pour vérifier la précision des données, les **coordonnées ont été copiées manuellement dans Google Maps**.

En les collant dans la barre de recherche de **Google Maps**, la position exacte du dispositif est apparue sur la carte, **confirmant le bon fonctionnement du module GPS** et de sa communication avec l’ESP32.

Ce test montre que le **GPS capte bien les signaux satellites** et fournit des **informations fiables**, ce qui est indispensable pour une alerte géolocalisée en cas de chute.

## Envoi de la position GPS par SMS

L’objectif principal du projet, côté utilisateur final (les proches du patient), est de recevoir un **message d’alerte contenant la localisation précise** du porteur du dispositif en cas de chute ou de situation d’urgence.

Après avoir validé séparément :

* l’envoi de **SMS** avec le module **SIM800L**,
* la réception des **coordonnées GPS** avec le module **Neo-6M**,

l'étape suivante consiste à **combiner ces deux fonctionnalités** dans un seul programme, afin d'envoyer automatiquement la position GPS par SMS lors d’un événement déclencheur.

* Montage et interconnexion des modules

L’ESP32 dispose de **trois ports UART**, dont l’un est déjà réservé pour la communication série USB (Moniteur série). Il reste donc **deux ports UART disponibles** pour la connexion avec les modules externes :

* **SIM800L** → connecté aux broches **GPIO 16 (RX)** et **GPIO 17 (TX)**
* **GPS Neo-6M** → connecté aux broches **GPIO 1 (TX)** et **GPIO 2 (RX)**

Les deux modules sont **alimentés par une source commune**, en veillant à respecter les niveaux de tension (3.3V ou 5V selon le modèle) et à éviter les surcharges.

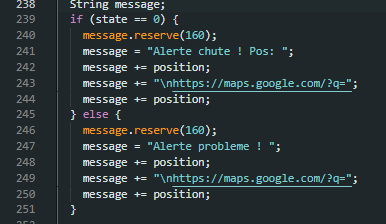
* Fonctionnement du programme

Le programme combine les logiques précédemment testées :

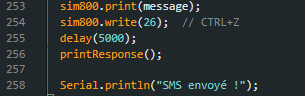
* Le module **GPS** capte la position géographique (latitude et longitude)
* Ces coordonnées sont **stockées dans une variable texte**
* Le module **SIM800L** est ensuite utilisé pour **envoyer ces données par SMS**
* L’action est **déclenchée par l’appui sur un bouton poussoir**

Cela permet de simuler une alerte manuelle ou une chute, et de tester l’enchaînement complet : détection → localisation → envoi de l’alerte.

* Exemple de logique en pseudo-code



Enregistrer la position dans la variable « message »



Envoyer le « message » vers la destinataire.

* Résultat du test

Lors du test :

* À chaque **pression sur le bouton**, un **SMS contenant les coordonnées GPS** a été reçu sur le téléphone configuré.
* Le **lien Google Maps** généré dans le message permet d’afficher directement la position sur une carte.
* Le fonctionnement des deux modules en simultané a été **stable** et **réactif**, démontrant que l’ESP32 peut gérer plusieurs communications UART sans conflit.

Ce test valide donc la **chaîne complète de détection et d’alerte par SMS avec géolocalisation**, cœur du projet.

## Intégration du bouton d’urgence et du buzzer distant

Le système de détection de chute ne doit pas seulement envoyer un message au proche du patient. En effet, un simple SMS peut être **insuffisant** : si le destinataire ne consulte pas immédiatement son téléphone ou s’il n’a pas de réseau au moment critique, l’alerte peut ne pas être prise en compte à temps.

Pour cela, un **système d’alarme à distance** a été ajouté. Il permet de déclencher une **alerte sonore (buzzer)** sur un autre appareil connecté, contrôlé **à distance via Internet**, pour attirer rapidement l’attention.

* Principe de fonctionnement

Lorsque le dispositif détecte une chute ou lorsqu’un **bouton d’urgence** est pressé :

* L’ESP32 utilise le **module SIM800L** pour envoyer une **requête HTTP** à un **service backend** développé avec **Spring Boot**.
* Ce service agit comme un **point de relais**, recevant l’information et la redistribuant à tous les appareils connectés.
* Les autres appareils, connectés via **WebSocket**, reçoivent cette alerte en temps réel et déclenchent **l’activation du buzzer**.
* Pourquoi utiliser Spring Boot ?

Le framework **Spring Boot** a été choisi pour plusieurs raisons :

* Il **simplifie la création d’API REST** (requêtes HTTP)
* Il prend en charge **WebSocket nativement**
* Il est **fiable**, **scalable** et **facile à déployer**
* Il est **adapté aux communications en temps réel**

L’application Spring boot sur le projet à pour role d’intermediaire entre les deux appareil et l’application mobile. Elle reçoi une requette http qui active l’alarme de l’autre appareil et enregistre les historique des alert reçu.

* Intégration dans le programme

Une **fonction a été ajoutée** dans le code de l’ESP32 pour effectuer une requête HTTP vers le serveur Spring Boot, en utilisant le **SIM800L comme modem GPRS**.

Cela nécessite une **connexion réseau mobile (data)**, et l’envoi de **commandes AT spécifiques** pour établir une session HTTP.

* Exemple de logique générale
* Résultat

Lors du test :

* Une **pression sur le bouton d’urgence** a bien déclenché une **requête HTTP**
* Le **serveur Spring Boot** a bien réceptionné la requête
* Les appareils clients connectés par **WebSocket** ont reçu l’alerte instantanément
* Le **buzzer** a été activé à distance

Ce système d’alerte **complète efficacement le SMS**, en assurant une **notification sonore immédiate** sur un autre terminal, même si le téléphone n'est pas consulté.

## Développement de l’application mobile pour les historiques

Pour faciliter la localisation du patient et consulter l’historique des alertes, une **application mobile** a été développée. Elle permet de :

* Coller directement les messages reçus
* Extraire automatiquement les coordonnées GPS
* Lancer la localisation dans des applications tierces comme Google Maps
* Afficher les alertes précédentes
* **Choix technologique**

Plusieurs solutions existent pour développer une application mobile :

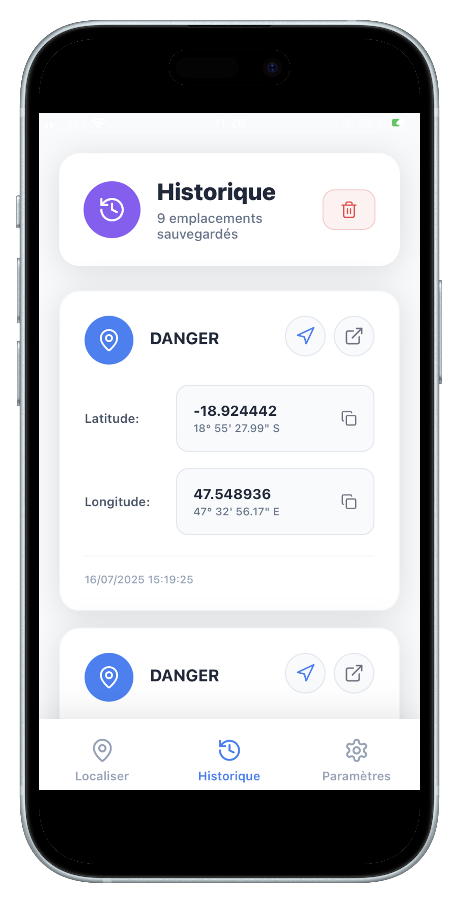
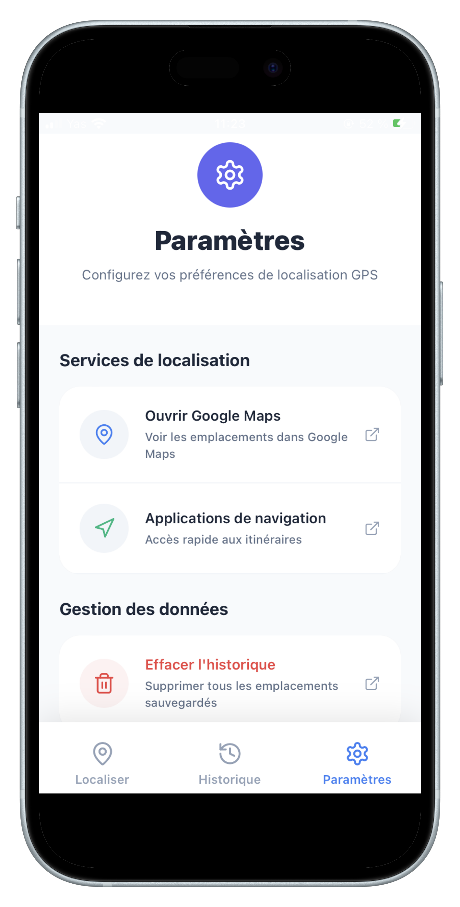
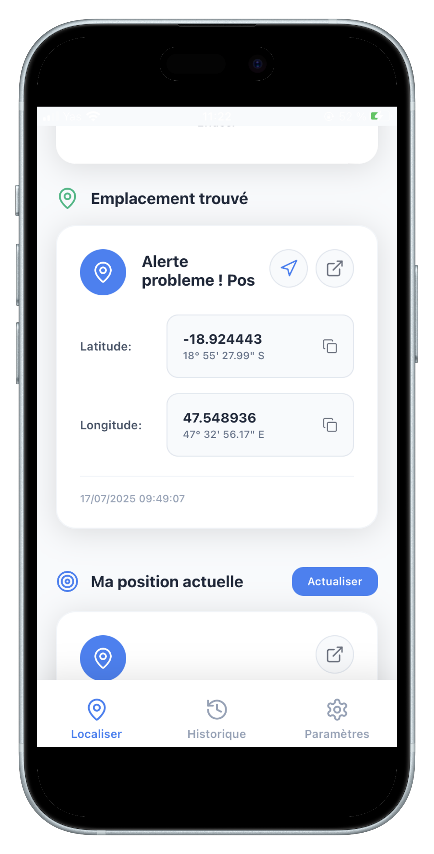
* Développement natif : Swift pour iOS, Kotlin/Java pour Android (deux codebases)
* Développement cross-platform : un seul code source compatible avec Android et iOS

Le développement cross-platform a été choisi pour sa simplicité de déploiement. Parmi les technologies disponibles, React Native a été retenue, en particulier avec Expo, qui facilite le développement et le test de petites applications.

* **Fonctionnalités de l’application**

L'application est composée de plusieurs pages :

* **Page de recherche**
* Champ pour coller directement le message reçu
* Extraction automatique des coordonnées (latitude, longitude)
* Bouton « Rechercher » qui affiche les résultats dans une **card** interactive
* Informations affichées :
  + Type d’alerte (CHUTE ou DANGER)
  + Coordonnées GPS
  + Date et heure
  + Bouton pour ouvrir la position dans une application de cartographie (Google Maps, Plan, etc.)
* **Page historique**
* Affichage de toutes les alertes enregistrées
* Les données sont récupérées depuis une base de données via l’API Spring Boot
* **Page réglages**
* Permet à l’utilisateur de définir ses préférences, comme l’application de cartographie par défaut
* **Interface de l’application**



* **Connexion avec l’API Spring Boot**

Le backend développé ne sert pas uniquement à déclencher l’alarme. Il enregistre également toutes les alertes reçues pour qu’elles soient consultables dans l’application mobile.

Deux endpoints sont exposés :

* **POST /api/dataplace**

Permet d’ajouter une nouvelle alerte.  
Le corps de la requête contient un objet JSON de la forme suivante :

* {
* "latitude": -18.924442,
* "longitude": 47.548936,
* "alertType": "DANGER",
* "time": "2025-07-16T12:19:25.488Z"
* },

En cas de succès, le serveur renvoie un code **201**.

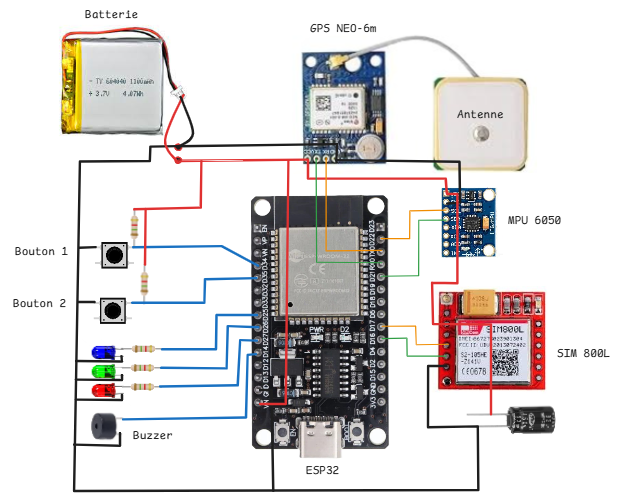
* **GET /api/allPlace**

Permet de récupérer toutes les alertes enregistrées.  
Réponse : liste JSON d’objets similaires.  
Code de succès : 200

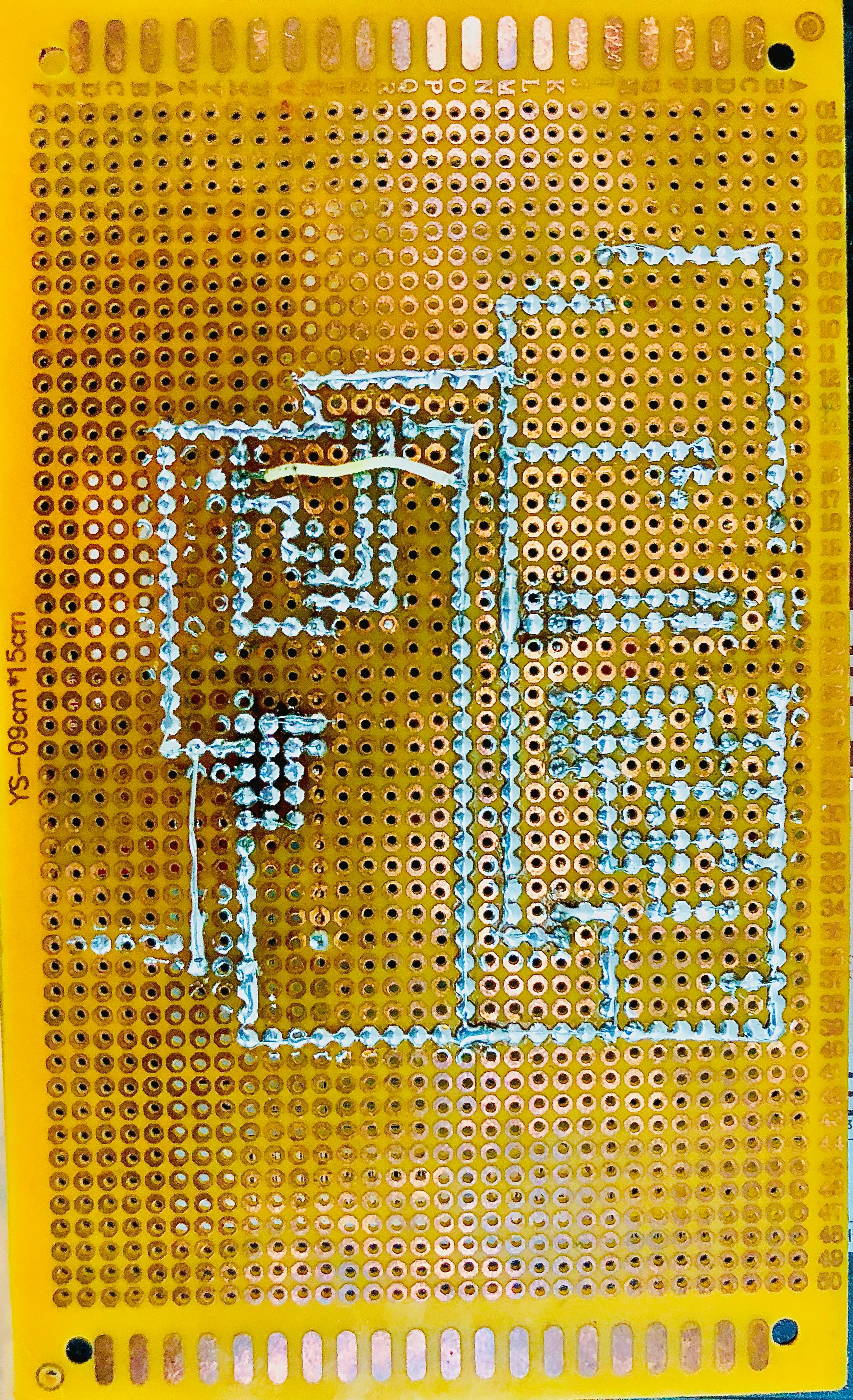
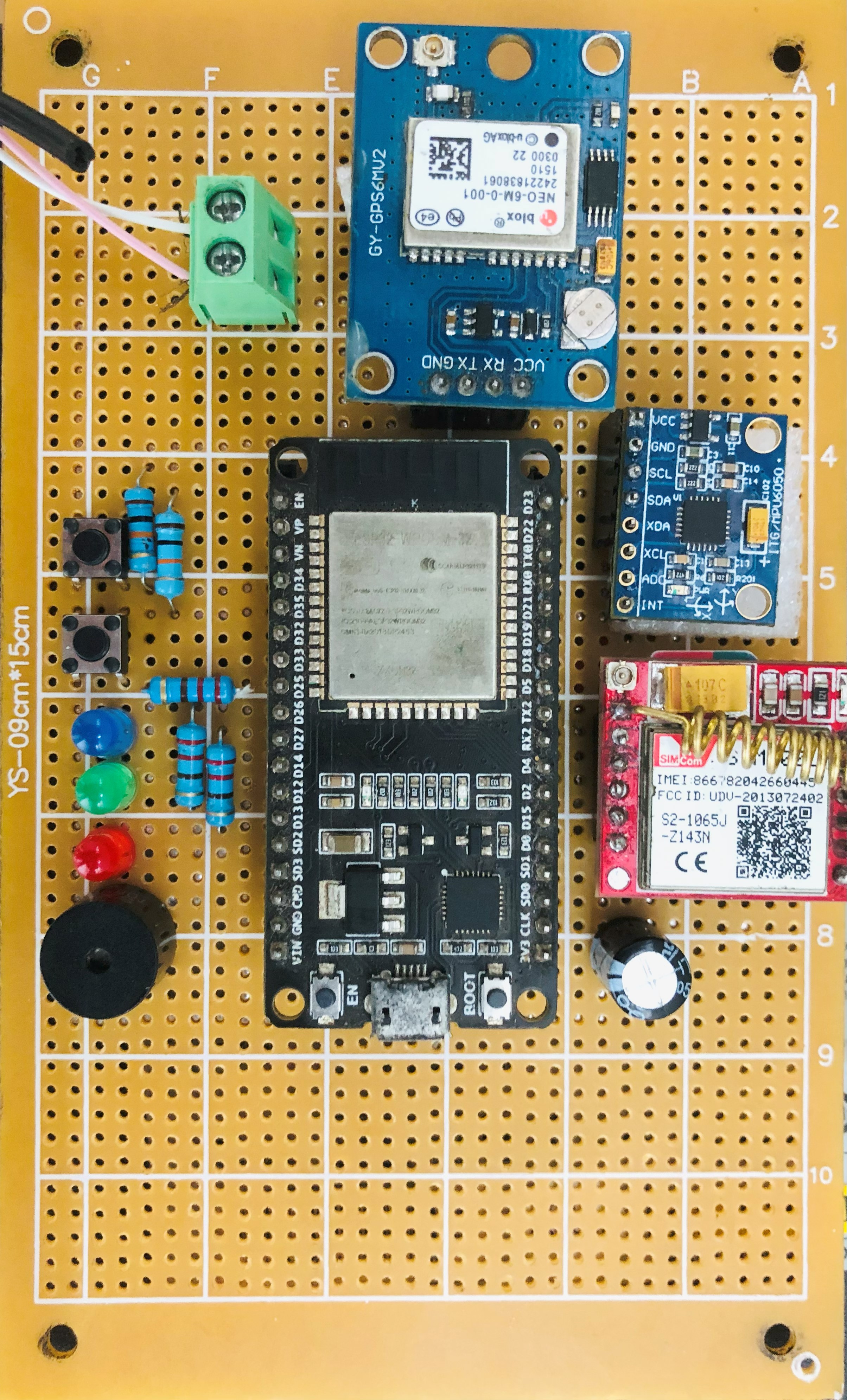
## Montage Final de tous les composant

Après avoir validé tous les tests de fonctionnement, l’assemblage définitif du dispositif peut être effectué.

* **Composants utilisés :**
* **2 cartes ESP32** :
  + L’une sert de **cerveau du dispositif porté par le patient** (capteurs, communication GSM/GPS).
  + L’autre est dédiée au **système d’alerte distant** (réception d’alerte et activation du buzzer).
* **Module SIM800L** : pour l’envoi des messages d’alerte à distance via le réseau GSM.
* **Capteur MPU6050** : pour détecter les chutes à l’aide de ses accéléromètre et gyroscope intégrés.
* **Module GPS NEO-6M** : pour déterminer les coordonnées de localisation.
* **3 LEDs** : pour les indications visuelles (état du système, transmission, alerte, etc.).
* **Buzzer** : pour générer une alarme sonore en cas d’alerte.
* **Plaquette de prototypage** : pour fixer et connecter les composants.
* **Batterie Li-Ion 3000 mAh** : pour alimenter le dispositif de manière autonome.
* **Connecteurs (dupont, alimentation, etc.)** : pour relier les composants de manière stable et sécurisée.
* Schema du montage



* Montage sur plaquette



# Chapitre 4 : Tests, résultats et validation

## Scénarios de test

Une fois le montage final réalisé, le programme installé et les applications fonctionnelles (mobile et backend), il est indispensable de passer à la phase de test. Cette étape permet de simuler des situations réelles afin d’évaluer le bon fonctionnement du dispositif, d’identifier ses éventuelles failles, et de mesurer ses limites.

L’objectif des scénarios de test est donc de reproduire des cas concrets d’utilisation, notamment une **chute réelle** d’un individu portant le dispositif.

**Scénarios testés :**

* **Efficacité de la détection de chute** :  
  Vérifier si le capteur MPU6050 détecte bien une chute lorsque le porteur tombe de manière soudaine. Le système doit déclencher une alerte automatiquement.
* **Communication à distance** :  
  Tester si l’envoi d’alerte via le module SIM800L fonctionne correctement (SMS ou requête HTTP). Cela permet de s’assurer que le proche du patient reçoit bien le message d’urgence.
* **Localisation avec l’application mobile et le backend** :  
  Vérifier si la position GPS envoyée est exacte, si elle est bien reçue par le serveur, et si elle s’affiche correctement dans l’application mobile. Ce test permet de valider la fiabilité du suivi de localisation en temps réel.

## Détection de chute : précision et efficacité

Le cœur du projet repose sur la **détection fiable des chutes**. Pour évaluer la précision du dispositif, des tests ont été réalisés avec un individu portant le système de manière réaliste.

* **Dispositif porté :**

Le système est porté comme une **ceinture**, positionné au niveau des **hanches**, une zone stratégique pour détecter les mouvements brusques du corps.

* **Tests effectués :**
* **Simulation de chute sur les quatre axes** (avant, arrière, côté gauche et droit) :  
  À chaque chute simulée, le dispositif déclenche une **alerte sonore immédiate** via le buzzer.
* **Cas d’immobilité prolongée** (> 5 secondes après la chute) :  
  Si l’individu ne revient pas en position verticale après 5 secondes, l’**alerte sonore continue** pour signaler un danger réel.
* **Cas de récupération rapide (< 5 secondes)** :  
  Si l’individu se relève avant 5 secondes, l’alerte se **désactive automatiquement**, évitant les fausses alertes.
* **Cas de fausse chute (saut, mouvement brusque)** :  
  L’alerte se déclenche brièvement. Cependant, en appuyant sur le **bouton d’annulation**, l’utilisateur peut **désactiver manuellement** l’alerte si aucune réelle chute n’a eu lieu.
* **Conclusion :**

La détection de chute est **efficace et précise** dans les différents scénarios testés. Le système est capable de :

* Différencier les vraies chutes des mouvements brusques,
* Gérer les fausses alertes grâce à un bouton d’annulation,
* Réagir rapidement aux situations critiques avec une alarme continue si nécessaire.

## Transmission des alertes et position GPS

Le système permet l’envoi d’alertes avec la **position GPS** du porteur via deux méthodes distinctes :

* **Méthodes de transmission :**

1. **Déclenchement automatique après une chute non relevée** :
   * Lorsqu’une **chute est détectée** et que l’utilisateur ne se relève pas dans un délai de 5 secondes, le dispositif envoie automatiquement un **SMS contenant les coordonnées GPS** à un proche prédéfini.
   * **Test validé** : Un message est bien reçu dans ce cas.
2. **Déclenchement manuel via un bouton d’alerte** :
   * L’utilisateur peut appuyer sur un **bouton d’urgence** pour envoyer une alerte manuelle, même sans chute.
   * Le SMS envoyé contient un **message d’alerte spécifique** ainsi que la **position actuelle**.
   * **Test validé** : Le message est bien reçu avec les données GPS.
3. **Annulation automatique si fausse alerte** :
   * Si l’utilisateur se relève avant les 5 secondes après une chute, **aucun SMS n’est envoyé**, évitant les notifications inutiles.
   * **Comportement conforme**.

* **Conclusion :**

La transmission des alertes fonctionne correctement dans tous les cas :

* Les **SMS sont envoyés au bon moment**,
* Les **coordonnées GPS sont précises**,
* Les **fausses alertes sont évitées** grâce à un délai d’attente.

## Tests de l'application mobile

Les tests ont été réalisés pour vérifier le bon fonctionnement des **fonctionnalités principales de l'application mobile** ainsi que la **connexion avec le service backend**.

* **Collage automatique du message d'alerte :**
* L’utilisateur copie le **message d’alerte reçu par SMS**.
* En ouvrant l’application, il clique sur le **champ de saisie**, et le message est **collé automatiquement**.
* L’application **reconnaît automatiquement** le message, en extrayant :
  + Le **type d’alerte** (chute ou bouton d’urgence),
  + Les **coordonnées GPS**.
* Une **notification de succès** est affichée confirmant la détection correcte du message.

**Résultat** : Le collage automatique fonctionne, et le message est correctement analysé.

* **Enregistrement et historique de la position :**
* En appuyant sur **"Rechercher"**, la position extraite est :
  + **Enregistrée dans la base de données**,
  + Affichée dans la section **"Emplacements recherchés"**,
  + Ajoutée à l’**historique des recherches**.

**Résultat** : La connexion entre l’application mobile et le backend fonctionne correctement.

* **Ouverture de l’emplacement dans Google Maps :**
* En cliquant sur l’**icône de localisation**, l’application ouvre **Google Maps** avec :
  + La **position du dispositif (patient)**,
  + La **position actuelle de l’utilisateur** (proche).

**Résultat** : L’ouverture dans Maps se fait sans problème avec les données GPS fournies.

* **Conclusion :**

Toutes les fonctionnalités essentielles de l'application mobile ont été **testées avec succès** :

* Extraction automatique des données à partir du message d’alerte,
* Enregistrement des positions via le service backend,
* Intégration fluide avec **Google Maps**.

## Limites et améliorations possibles

Bien que les tests du prototype final aient été concluants, certaines **limites** ont été identifiées, ainsi que des **pistes d'amélioration** pour optimiser le dispositif :

* **Connexion GPS lente :**

Le **module GPS** doit d'abord établir une connexion avec les satellites pour obtenir une position. Cette étape peut prendre du temps, surtout en environnement fermé ou urbain dense.  
➡️ **Amélioration proposée** : Utiliser une **antenne GPS plus performante** et plus réactive pour accélérer la synchronisation satellite.

* **Dépendance à la carte SIM :**

Le dispositif repose sur une **carte SIM** pour envoyer les alertes par **SMS** ou via la **connexion mobile (GPRS)**. Cela suppose que :

* La puce dispose en permanence de **crédit SMS**,
* Une **connexion de données mobiles** soit active.

➡️ **Amélioration proposée** : Intégrer un système de **vérification automatique du solde** ou un mode **hors-ligne alternatif** (ex. : alerte sonore si l’envoi échoue).

* **Taille et portabilité du dispositif :**

Le dispositif, dans sa version actuelle, peut encore être perçu comme **encombrant** pour un port continu par la personne.

➡️ **Amélioration proposée** : Optimiser le **design** et réduire la **taille du boîtier** à l’avenir, notamment en intégrant des composants plus compacts.

**Conclusion :**

Ces limites n’entravent pas le bon fonctionnement du dispositif, mais leur prise en compte dans les futures itérations permettrait d’en **améliorer l’efficacité, la fiabilité et le confort d’usage**.

**Conclusion Générale**

* Synthèse des travaux
* Apports du projet
* Perspectives d’évolution

**Annexes**

* Schémas électroniques
* Code source (extraits)
* Interface de l'application mobile

## Description des composants matériels

* 2.2.1 ESP32 : microcontrôleur principal
* 2.2.2 MPU6050 : capteur accéléromètre/gyroscope
* 2.2.3 NEO-6M : module GPS
* 2.2.4 SIM800L : module GSM pour envoi de SMS
* 2.2.5 Boutons : alerte manuelle et annulation
* 2.2.6 Buzzer et ESP32 secondaire côté surveillant  
  2.3 Conception des algorithmes de détection de chute  
  2.4 Gestion des cas d'urgence et fausses alertes

**Bibliographie**