IMPLEMENTACIÓN DE UNA SOLUCIÓN IoT

1st Fabián Andrés Díaz Martínez
Facultad de Ingeniería
Pontificia Universidad Javeriana
Bogotá D.C., Colombia
Fabian_diazm@javeriana.edu.co

2nd Nicolás Santiago Angulo Forero
Facultad de Ingeniería
Pontificia Universidad Javeriana
Bogotá D.C., Colombia
Angulof.@javeriana.edu.co

3rd Miguel Cera
Facultad de Ingeniería
Pontificia Universidad Javeriana
Bogotá D.C., Colombia
Cera-mangel@javeriana.edu.co

***Abstract*—Este proyecto presenta una solución IoT en forma de un sistema de detección de caídas en personas mayores a través de un sistema de identificación de movimientos bruscos. La solución emplea el sensor GY-91 para detectar caídas, activar una alerta y enviar notificaciones a las enfermeras del ancianato mediante el protocolo MQTT y la aplicación Telegram para aquellos encargados, facilitando una reacción rápida en situaciones de emergencia.**

I. Definición de la problemática

A. Introducción al problema

Una de las principales causas de lesiones en personas de edad avanzada son las caídas, sobre todo en aquellos que viven solos o que sufren algún tipo de discapacidad en su movilidad o en su vista. En Estados Unidos, 1 de cada 4 adultos mayores a 65 años tienen al menos una caída cada año, lo que significa aproximadamente 36 millones de caídas por año, de acuerdo con los Centros para el Control y la Prevención de Enfermedades [1]. Aunque no todas las caídas generan lesiones, más de un tercio de quienes caen requieren tratamiento médico o ven limitada su actividad, lo cual se estima en unos 8 millones de lesiones cada año. De hecho, alrededor del 20 % de las caídas ocasionan lesiones graves como fracturas óseas, especialmente de cadera, o traumatismos en la cabeza. Estos riesgos son aún más graves debido a la fragilidad de los huesos que se suele tener en estas edades. Además, el miedo a caerse puede generar problemas como la limitación de sus actividades diarias o la perdida de la confianza propia, afectando su independencia [1]. Por esto, el informar de las caídas al personal médico es crucial para prevenir futuros accidentes y reducir sus consecuencias.

B. Descripción de la necesidad

Por la problemática anterior, se tiene la necesidad de crear un sistema en el ancianato que pueda detectar las caídas en tiempo real y notifique a las enfermeras, para que así se tomen medidas rápidas. La rapidez en la respuesta es fundamental para reducir los riesgos en la salud que surgen por la falta de atención oportuna. Sin embargo, muchos adultos mayores no informan sobre sus caídas por miedo a perder su autonomía o por la idea errónea de que las caídas son una parte del envejecimiento. Por lo cual, un sistema automático puede superar esta dificultad, ya que permite alertar sobre una caída sin depender de que la persona informe el evento, ayudando así en la tranquilidad tanto de los adultos mayores como de sus familiares.

C. Propuesta de solución IoT

Para resolver la necesidad se propone una solución IoT que va a utilizar el sensor GY-91 (el cual tiene acelerómetro, giroscopio y magnometro (9 Ejes), los cuales ayudan a medir la aceleración relativa, velocidad de giro y posición correspondientemente. Además de la facilidad de poder incluir otros sensores útiles como temperatura para condiciones en personas mayores y enviar alertas automatizadas. El sistema estará diseñado para identificar una caída cuando detecte movimientos fuertes o bruscos; en ese momento, se enviará una notificación a los cuidadores mediante el protocolo MQTT o Telegram, permitiendo así una intervención rápida

D. Objetivo

Este proyecto tiene como objetivo realizar un sistema IoT que pueda detectar caídas en personas mayores y brindar un aviso de manera automática a sus familiares o cuidadores. Por otro lado, este sistema tiene como finalidad mejorar la calidad de vida de los adultos mayores al reducir el tiempo de respuesta a la alarma en caso de una emergencia y garantizar una atención oportuna y adecuada ante eventos peligrosos.

II. Diagrama esquemático de los dos dispositivos

A continuación se verán los esquemáticos correspondientes a los nodos IoT principales que utilizaremos en nuestro diseño de solución, estos esquemáticos corresponden al nivel de manejo a conexiones circuítales entre los embebidos

A. Nodo 1

Como podemos ver, este Nodo IoT corresponde a la simulación de uno de un Wearable que será otorgado a una persona mayor de edad, el cual podría vestir de manera como debido a su portabilidad y discreción debió al bajo tamaño de los componentes. Este nodo es bastante simple, ya que corresponde únicamente a una conexión entre el embebido ESP32 S3 Zero y el sensor que utilizaremos GY-91, la comunicación entre estos tiene una interfaz de comunicación I2C, en el que debemos primeramente conectar los puertos SCL y SDA (Clock y Data) y estos dirigirlos a unos pines del ESP los cuales permitan esta comunicación para finalmente ser configurados por medio de software y que pueden censar

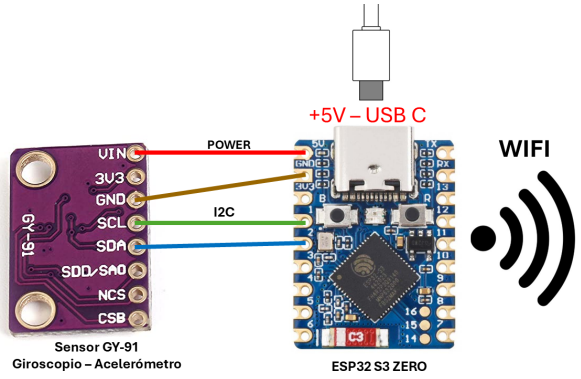

Fig. 1. nodo 1

las señales. La alimentación de ambos se puede realizar por medio del puerto USB-C que tiene el ESP y que alimentara ambos dispositivos, aunque se valida la posibilidad de usar una batería externa para simular aún más es Wearable. Y por último, la tecnología de comunicación será por medio de WI-FI de 2.4 Ghz el cual es lo bastante versátil para comunicarse de manera inalámbrica y hacer publicaciones MQTT.

B. Nodo 2

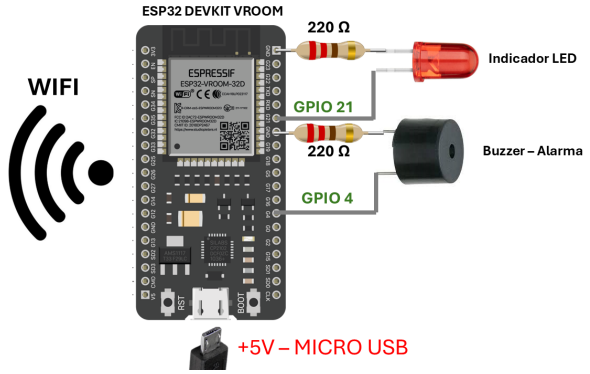

Fig. 2. Nodo 2

Este nodo ya es correspondientemente asignado a los puestos de las enfermeras, en donde encontraremos este sistema de alarmas simulado, la característica especial es que este sistema cuenta con 2 tipos de alarmas debió a la distracción o evasión que se puede realizar en una de estas, debido a esto se usará entonces una alarma visual (Lumínica) y otra auditiva (Sonido) para alertar a la enfermera, además de una intervención por medio de ID en la aplicación y mensaje de texto para identificar parámetros especiales como, qué persona fue específicamente el del accidente, número de habitación y demás metadatos que son útiles para la identificación del paciente. El nodo a ser únicamente receptor de alertas, no es esencial en el tamaño por lo que podemos usar un ESP32DEVKIT o incluso una tarjeta NodeMCU ESP8266, en donde la comunicación se hace por WIFI esperando las alertas externas, y se asigna por pin del embebido al circuito para que previamente pueda ser usado ambos actuadores. La alimentación si se puede realizar por medio micro USB debido a que se encuentra en un lugar estacionario y no debería ser movilizado en mayor cantidad.

III. Presupuesto

A continuación, se detalla el presupuesto de los componentes necesarios:

- Sensor GY-91: $37,000 COP
- NodeMCU ESP8266: $20,000 COP
- ESP32 S3 Zero: $20,000 COP
- Protoboard: $10,000 COP
- Actuadores: $5,000 COP
- Cables: $5,000 COP
- Total estimado: $97.000 COP

IV. Dispositivos integrados

Como se puede evidenciar, el número de dispositivos integrados en esta topología están definidos de acuerdo a los nodos IoT y los dispositivos externos como pueden ser los PC's, en este caso, tendremos 2 Nodos IoT importantes, el primero, aquel encargado de censar el estado del paciente y tener prioridad de enviar alertas en caso de detectar o suponer una caída, y aquel encargado de estar escuchando dichas alertas para realizar una serie de procesos en unos actuadores capaces de informar a las enfermeras de una posible hipótesis. Es por eso que podemos describir las siguientes funcionalidades de cada nodo de acuerdo a la siguiente lista:

Funciones Nodo 1 "Wearable"

- Censar variables de posición del paciente
- Hacer aproximaciones con la información para una posible localización de una caída.
- Definir información por paciente que porte el Wearable
- Enviar alertas por protocolo MQTT hacia un broker para informar de lo sucedido

Funciones Nodo 2

- Estar vigilante frente a cualquier alerta que se envíe desde los demás nodos.
- Dar alerta a los encargados del establecimiento como y a los familiares que se percató un incidente y ya se encargara de ello.
- Generar la lógica de aviso y señal para recurrir acerca de la situación y que las enfermeras se enteren por un medio visual y sonoro del incidente.

Aparte de ello y no menos importante, se integran una red computacional en la que las practicantes no solo se enteran por medio de actuadores, sino que, por el contrario, se cuenta con un programa computacional que tiene un registro y una serie de gráficas que muestra en estado de cuidado, así como información útil como nombre, número de habitación y demás datos del usuario que dan una mejor perspectiva de la situación. Es por eso que la integración de un computador también lo podemos incluir en los dispositivos útiles que nos brindan una red más comprometida. En este caso, la idea es contar que su utilidad radica en el programa computacional local que se conecta hacia los demás servicios brindados.

V. Topología de la red

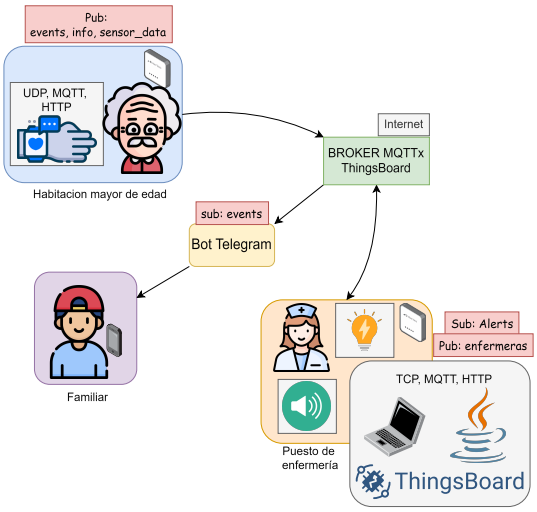

Fig. 3. Esta es una descripción de la imagen.

Para el diseño de esta topología de red se modifica un poco en comparación con la anterior, pues es necesario para un sistema de monitoreo remoto con mejor especificación, se implementa un esquema basado en MQTT para la comunicación entre dispositivos IoT, un bot en Telegram para notificaciones en tiempo real y una aplicación que alberga un dashboard de visualización, de métricas y demás.

La configuración está centralizada hacia un broker MQTT en el servicio MQTTX.app, que permite la transmisión de mensajes entre dispositivos dentro de una red interna. En la habitación de la persona mayor de edad, se añade el wearable que irá conectada a la persona mayor de edad y que envía datos mediante la publicación hacia el broker a través de un punto de acceso WiFi (Access Point) hacia el broker. Esta configuración permite monitorizar el estado de la persona de manera constante, Cabe aclarar, que en este caso este dispositivo solo se encargara de hacer publicaciones.

En la estación de enfermería, otro dispositivo MQTT suscriptor (MQTT-SUB) y publicador para los actuadores recibe los datos publicados desde la habitación. El dispositivo en la estación de enfermería está configurado con alertas visuales y sonoras para notificar al personal de cualquier situación relevante en la habitación de la persona mayor, además del acceso a la aplicación para obtener información más relevante (ID, Número de habitación, timestamp, etc.).

Además, se integrará un bot de Telegram que actúa como cliente suscriptor del broker. Este bot es capaz de enviar mensajes a los usuarios (Familiares - Enfermeras) o notificaciones directamente en Telegram en caso de alertas. La arquitectura incluye también una aplicación en un servidor central que recibe los datos de los dispositivos a través del broker, procesa la información y permite su visualización para el análisis de esta. Gráficas y posición permiten ser graficadas, además de información de la persona.

A partir de ello es importante definir los tópicos que vamos a tratar, por ende consideremos primero al usuario potencial, los mayores de edad deben ser constantemente monitorizados, es por ello que debemos tener en cuenta 3 pilares básicos respecto a la información.

- Primero, de quienes estamos hablando y a quién se está monitorizando, aparte de ello donde corresponde su lugar y demás, estos datos permiten localizar a la persona de manera eficaz.
- Segundo, dada la emergencia, se debe considerar el estado de la persona, es por ello que podemos dividir esto en un subtopico hijo de aquella persona monitoreada, por ende el estado corresponde de manera directa una métrica a considerar.
- Por último, pero no menos importante, la aplicación en IoT para obtener métricas que nos permitan hacer análisis, por ejemplo en este caso de sedentarismo, lo que puede ser útil, por ende podemos obtener los valores del sensor en el tiempo y así generar un análisis y enfoque más particular hacia los datos censados, dejando atrás la parte de la emergencia, este también se puede considerar un subtema para la persona monitorizada.

Es por ello que la definición de los tópicos finalmente quedaría de la siguiente manera, para ser más general se consideran tópicos más grandes, los cuales pueden dar un mejor contexto:

- hospital/patient/info: Para dar información con respecto a la persona que se están censado, tiene valores como nombre, habitación, género y un ID especifico.
- hospital/patient/info/events: Para el manejo de los estados de como se encuentra el paciente, en total se consideran 3 valores, el tipo de alerta (alertYype), un mensaje que de contexto (alertMessage), y un alerta del tiempo en que ocurre (alertTimestamp)
- hospital/patient/info/sensor$_data$: $Estetópicobuscabrindarunainformaciónentiemporealacercadelapos$

VI. Modificaciones adicionales

No se realizaron cambios radicales en cuanto al diseño de la solución, pues se considera una idea robusta, es por eso que el mayor cambio radica en la viabilidad del censado, ya que a pesar de que es posible la detección de caídas con esta seria de sensores, la lógica puede llegar sé compleja y tan diversa como se desee, en nuestro caso se opta por una solución sencilla y eficaz basado un la física y los valores obtenidos, la idea es simple y consiste en revisar un cambio brusco en cuanto a aceleración y ángulo para así comparar dichos valores obtenidos con unos umbrales los cuales podemos aproximar, es por eso que obteniendo la magnitud y la posición podemos establecer una relación.

Una característica importante es que para el diseño de nuestra aplicación, se decide usar la plataforma ThingsBoard la cual es una plataforma que puede usar la comunidad y nos permite al igual que MQTTx hacer publicaciones y suscripciones a un broker, con la característica de que podemos

diseñar interfaces de usuario (Dashboards) de manera más sencillas y profesionales en comparación a hacer toda la implementación por software, valor que puede ser útil tanto en la integración como en la visualización, a pesar de su uso la aplicación se realiza totalmente en Java con ayuda e JavaFX y SceneBuilder.

VII. Documentación componente

Para esta seccion se especifican los componentes y sus caracteristicas principales por lo que son destacados, desde los 3 principales, hasta aquellos mas comunes pero igual de relevantes. Primeramente se puede definir el ESP32 Devkit, una de las piezas clave de este sistema, es un microcontrolador basado en el chip ESP32, que destaca por su capacidad de conectividad inalámbrica integrada. El módulo cuenta con soporte para WiFi 802.11 b/g/n y Bluetooth 4.2 (tanto BLE como clásico), lo que permite una comunicación fluida en aplicaciones IoT. Su procesador dual-core Xtensa a 240 MHz ofrece un rendimiento robusto para manejar tareas concurrentes, mientras que su memoria RAM de 520 KB y soporte para almacenamiento externo, como microSD o Flash, amplían su versatilidad. Además, dispone de 30 pines GPIO configurables que soportan interfaces como UART, I2C, SPI y PWM, fundamentales para conectar sensores y actuadores en la habitación en caso de ser necesario.

El ESP32-S3 Zero, una variante avanzada del ESP32, incluye características orientadas a aplicaciones más complejas. Este microcontrolador incorpora un procesador dual-core Xtensa LX7 con soporte para operaciones vectoriales, ideal para inteligencia artificial y procesamiento de imágenes, ademas de una intervención propia del IoT en el que se hace una encriptación hardware. Cuenta con 512 KB de RAM interna y hasta 8 MB de PSRAM para manejar aplicaciones más pesadas, su capacidad gráfica destaca con su conectividad WiFi de doble banda y Bluetooth 5.0 mejora tanto la velocidad como la estabilidad de la comunicación. Esto lo convierte en una elección óptima para la estaciónes de monitoreo, donde se requieren respuestas rápidas y la capacidad de interactuar con interfaces avanzadas.

Por ultimo, el sensor GY-91 es un módulo compacto que integra dos dispositivos clave: el MPU-9250, que combina un acelerómetro y giroscopio de 9 ejes, y el BMP280, un sensor barométrico de alta precisión. El MPU-9250 permite medir aceleraciones y velocidades angulares en los tres ejes, proporcionando datos críticos para detectar caídas o cambios en la posición. Por su parte, el BMP280 es capaz de medir la presión atmosférica con una precisión de ±1 hPa, lo que resulta útil para calcular la altitud relativa en entornos más complejos. Este sensor se comunica a través de interfaces I2C y SPI, garantizando compatibilidad con los microcontroladores usados en el proyecto.

Para las notificaciones locales, se incluyen un LED y un buzzer. El LED, de tipo estándar, opera en un rango de voltaje de 1.8 a 3.3 V, proporcionando un indicador visual de estados del sistema o alertas activas. Su bajo consumo de corriente (10-20 mA) lo hace ideal para aplicaciones IoT con restricciones energéticas. El buzzer, por su parte, es un actuador sonoro piezoeléctrico con un rango de frecuencia ajustable entre 2 y 5 kHz, suficiente para generar sonidos claros y audibles en ambientes ruidosos. Ambos elementos son controlados mediante pines GPIO poniendo en alto el pin, asegurando una respuesta rápida y eficiente ante cualquier alerta detectada.

VIII. Creación De bot de Telegram:

Para realizar la creación del bot, primero iniciamos una conversación de Telegram con un bot llamado ”BotFather”. aquí creamos un nombre para el bot, el cual llamamos “Arduino”. Luego creamos un nombre de usuario al bot. En este caso le asignamos el nombre “Alerta-caida-bot”. Una vez hecho esto, el bot ”BotFather”, nos envió el token que consiste en una cadena de caracteres única, la cual permite autenticar las solicitudes que realizaremos a la API de Telegram usando el nombre de nuestro bot. El token de nuestro bot es: 7882291525:AAEfH-HGwX01omlYKM0E6C4WOln0Vh70jU9A.

Después de la creación del bot, se debe obtener el ID del chat. Este ID se usa para identificar a qué chat se debe enviar el mensaje. Para esta solución, se debe agregar el chat id en el código del Arduino para que el mensaje llegue al bot adecuado. Para obtener el ID, iniciamos una conversación con un bot llamado ”IDBot”, el cual con solo usar el comando /start, nos mostrará la información del ID. En este caso, nuestro ID es: 1154902509.

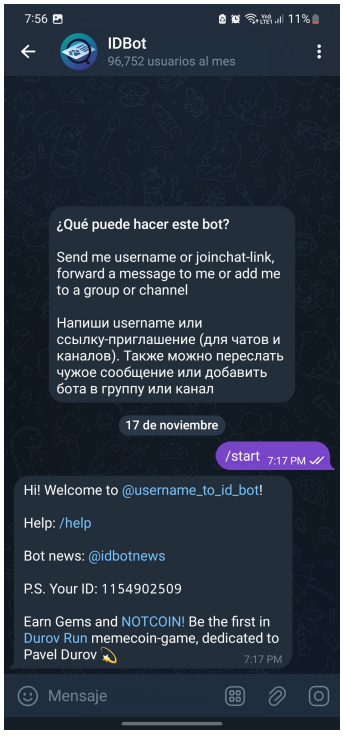

Fig. 4. Obtención ChatID

Fuente del codigo: https://github.com/savaged22/IOT$_{F}inals/blob/main/Proyecto_{r}edes_{f}inal/ProyectoRedes/src/n$

IX. Documentación de software:

A. Nodo 1

a aplicación consistente para este dispositivo, se realiza en IDE de Arduino, el cual brindaba librerías para la conexión y una integración con los embebidos que decidimos usar, es por eso que a continuación se da un desglose más especifico de como se desarrolló el código del Wearable.

1) Librerías implementadas: Las librerías utilizadas en el programa de Arduino, corresponde a:

```
#include <WiFi.h>
#include <WiFiUdp.h>
#include <PubSubClient.h>
#include <MPU9250_asukiaaa.h>
#include <Adafruit_BMP280.h>
#include <HTTPClient.h>
#include <NTPClient.h>
#include <ArduinoJson.h>
```

1) **WiiFi.h**: Permite la conexión a internet a través de WiFi. Esta permite el uso de funciones como WiFi.begin(ssid, password) la cual establece la conexión con el usuario y contraseña para un punto de acceso, proporcionando las credenciales de esta.
2) **WifiUdP.h**: Se usa para manejar conexiones UDP de manera más rápida, en nuestro caso se utilizó para sincronizar la hora para el manejo de los metadatos que se iban a usar.
3) **PubSubClient.h** : Permite la implementación de MQTT, permite el envío y recepción de mensajes entre el broker y el microcontrolador.
4) **MPU9250_asukiaaa.h** : Usado para manejar el sensor MPU9250, el cual cuenta con acelerómetro y giroscopio. Permite funciones útiles en el sensor como mySensor.accelUpdate(), mySensor.accelX() y mySensor.accelY(), mySensor.accelZ().
5) **HTTPClient.h** : Permite el manejo de solicitudes HTTP para enviar datos a la API de Telegram y ThingsBoard.
6) **NTPClient.h** : Permite sincronizar la hora del chip usando Network Time Protocol.
7) **ArduinoJson.h:** Permite el uso de objetos JSON para enviar datos al servicio MQTT. Usa funciones como serializeJson(doc, string) convierte JSON a cadena de texto para enviarla.

```
#define SDA_PIN PIN_SDA
#define SCL_PIN PIN_SCL
```

La definición del pin 21 como SDA(Serial Data Line) para transmitir datos, y el pin 22 como SCL para transmitir la señal de reloj para sincronizar comunicación por medio del protocolo de comunicación I2C.

2) Credenciales:

```
const char* ssid = "SSID_NETWORK";
const char* password = "PASSWORD_NETWORK";
const char* thingsboard_server =
    "THINGSBOARD_URL";
const char* mqtt_server =
    "MQTT_BROKER_URL";
const int mqtt_port = 1883;
const String bot_token =
    "TELEGRAM_BOT_TOKEN";
const String chat_id =
    "TELEGRAM_CHAT_ID";
```

En primero, lo que realizamos es definir las credenciales de la red que vamos a usar, como se puede ver, tenemos aquella del access point de la red, aquellas del broker ThingsBoard y también del server MQTTx, tanto valores estáticos como el puerto, e incluso valores para la conexión con el Bot de Telegram.

La configuración a la red wifi con el ssid y contraseña de la red, estos valores deben ser reemplazados por los de la red a la cual se va a conectar el chip. Por otro lado, se realiza la configuración del URL de ThingsBoard, una plataforma que nos permite almacenar y visualizar los datos enviados de forma más gráfica, con tablas gráficos y alertas. Por último, se realiza la conexión al broker MQTT, en este caso, se utilizó un broker público llamado MQTTX.

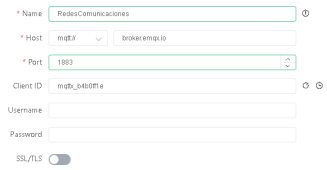

Fig. 5. Conexión MQTTX

```
WiFiClient wifiClient;
HTTPClient http;
WiFiUDP udp;
WiFiClient espClient;
PubSubClient mqttClient(espClient);
NTPClient timeClient(...);
Adafruit_BMP280 temperatureSensor;
MPU9250_asukiaaa motionSensor;
```

En el segmento de código se puede ver como se crean instancias de clases necesarias para diversas funcionalidades del ESP32. *WiFiClient wifiClient* maneja la comunicacipón con la red WiFi, se usa para conectarse a servidores HTTP y el broker MQTT. *HTTPClient http* esta intancia permite al chip realizar solicitudes HTTP como get o post en servidores. *WiFiClient espClient* es un cliente que permite las conexiones TCP/IP, mientras que *PubSubClient client(espClient);* es la instancia que maneja el protocolo MQTT (publish/subscribe). *NTPClient timeClient(udp, "pool.ntp.org", 0, 60000);* esta instancia obtiene la hora actual desde un servidor de tiempo, lo que permite sincronizar el timestamp con datos.

Por último, se realiza la instancia de los sensores: *MPU9250 asukiaaa mySensor* este sensor mide aceleración, cuenta con giroscopio y magnómetro y se usa para medir movimientos bruscos que puedan significar una caída o accidente.

```
// Información usuario
const int userId = 510;
const String userName = "USER_NAME";
const String location = "USER_LOCATION";
const String gender =
    "USER_GENDER";

// Variables sensor
float accelX, accelY, accelZ,
      gyroX, gyroY, gyroZ;
```

En la imagen anterior, se realiza la declaración de variables donde se almacenarán las medidas del sensor y los valores del usuario, valores que son importantes para identificar a cada usuario de acuerdo a su situación de movilidad, a partir de ellos existen 3 correspondientes variables, con respecto a la aceleración de los 3 ejes y el giroscopio que se realiza también.

```
// Función para enviar los datos al broker
void sendThings(
    String jsonData,
    String mqttTopic
) {}

// Función para conectar al broker MQTT
void connectMQTT() {}

// Función para detectar caídas
bool detectFall(
    float aX,
    float aY,
    float aZ
) {}

// Función para crear el JSON
    con datos de usuario
String createJson_user_data(
    int userId,
    String userName,
    String location,
    String gender
) {}

// Función para crear el JSON
    con datos del sensor
String createJson_sensor_data(
    float aX,
    float aY,
    float aZ,
    float gX,
    float gY,
    float gZ
) {}

// Función para crear el JSON
    con datos de alertas
String createJson_alerts_data(
    String alertType,
    String alertMessage,
    String alertTimestamp
) {}

// Función para enviar alerta a Telegram
void sendTelegramAlert(
    String message
) {}
```

A partir del código, podemos separar la lógica en varias funciones que se usan en la lógica completa, es por ello que haciendo una descripción de cada una de esta podemos definir así, que “sendThings” realiza el procedimiento para enviar datos al Broker. Como se puede evidenciar se reciben 2 parámetros, los datos que se desean enviar (JSON) y el topic a que se hace referencia. A partir de eso, la lógica consta de, primero, verificar la conexión WiFi, luego se establece la conexión con el Broker a través del *wifiClient* y las credenciales anteriores. Luego, se configura el tipo de contenido, en este caso se dice que los mensajes estarán en forma JSON. Después, se realizará envía el JSON, siendo jsondata los datos en JSON. Finalmente, se verifica el mensaje de respuesta del servidor y se libran recursos de la conexión.

connectMQTT, se encargará de enviar los datos a través de MQTT al broker MQTTX. Primero, se verifica conexión con el broker, este es únicamente para validar la conexión entre el nodo y el Broker, pues es una función persistente que verifica la conexión y se mantiene en esta hasta que se logre la conexión. es por esto que suele ser importante. Primero, mientras que no se haya establecido conexión, muestra por consola un mensaje ”Conectando al broker MQTT...” y así mismo se informa cuando se establezca conexión ”Conectado”.

La función *detectFall()*, es la logica fisica que se implementa en softwaren, encargada de validar por medio de booleanos el estado, esta asi recibe como parámetros las aceleraciones de los 3 ejex X,Y,Z provenientes del sensor MPU9250. Luego, se establecen los umbrales (como constantes) a partir de los cuales se consideraría una caída. Luego se realiza el cálculo de la magnitud escalar de la aceleración como la raíz de los cuadrados de las tres aceleraciones al igual que el cálculo del ángulo suponiendo una caída sobre el eje Z. Los valores encontrados se muestran en consola, y luego se comparan con los umbrales para evaluar una posible caída en caso de que alguno de los umbrales sea superado y retorna *true*. Esta hipótesis de caida esta basada en la lógica de [2] la cual muestra una seria de resultados próxima con una implementación relativamente fácil.

Anteriormente, se habían definido variables con el userId, userName, location y gender. Luego, se hace la conversión de JSON a string, y se retorna, por medio de las funciones createJson, la cuales toman los parámetros de las entidades para finalmente devolver un string en formato JSON, que próximamente será enviado hacia el broker. Así también se cuenta con los datos de una alerta como tipo de alerta, mensaje y el timestamp. Por último lo convierte en formato JSON para

ser retornado y enviado. De igual forma y no menos importante para las métricas, la función JSON para los datos del sensor.

3) setup():

*4) **inicialización del puerto serie**:* Esta función siempre se ejecuta al inicio, debidó a que corresponde a la configuración inicial del firmware. Con "Serial.begin(115200);" establecemos comunicación serial, para hacer nuestro propio Debug, se configura la velocidad de comunicación del puerto serial a 115200, para permitir imprimir datos en el monitor de depuración. Por otro lado, "while (!Serial);" nos da garantía que la comunicación esté lista antes de continuar, esta línea es importaré para que haya el monitoreo de mensajes de depuración durante el proceso de arranque. En el setup también se establece la primera conexión a wifi y además el cliente MQTT, y de igual forma la inicialización de los sensores por la función Wire.begin.

Para ser más precisos en la "Configuración de Wifi en el setup", primero se inicia una conexión usándose el SSID y la contraseña especificados con "WiFi.begin(ssid, password)", luego se inicializa el cliente NTP(timeClient), que lo usamos para obtener la hora actual desde un servidor NTP(usado más adelante en el bucle loop) y con el bucle "while (WiFi.status() != WL_CONNECTED)" hacemos que al microcontrolador espere hasta que se logre la conexión. Durante este proceso, se imprimen puntos (".") para indicar que la conexión está en progreso. Una vez se logre conectar se mostrara un mensaje en el monitor "wifi conectado", y la red le asignara una IP que es útil para la depuración y monitoreo de estado.

Con "client.setServer(mqtt_server, mqtt_port)" podemos establecer la conexión con MQTT, este establece la dirección del broker MQTT(mqtt_server) y el puerto (mqtt_port) la cual se conectara el cliente MQTT(client), en este caso se va a ser uso de un broker público. La secuencia "Wire.begin(SDA_PIN, SCL_PIN)" inicia el bus I2C en los pines 21(SDA) y 22(SCL) del ESP32, permitiendo la comunicación con los sensores. Luego, "mySensor.setWire(&Wire)" asigna el bus al sensor MPU9250 para que pueda comunicarse correctamente. Posteriormente, "mySensor.beginAccel()" inicia el módulo de acelerómetro del MPU9250, habilitando la lectura de datos en los ejes X, Y y Z. Por otro lado, "bme.begin()" inicia el sensor BMP280,el cual mide temperatura, presión y altitud. Finalmente, si la inicialización del BMP280 tiene éxito, se imprime "SI INICIO" en el monitor serie, para indicar al desarrollador que el sensor está funcionando correctamente.

*5) **loop()**:* Esta función se ejecuta continuamente, Comprueba la conexión con el broker MQTT y la restablece si se necesita, Luego de esto esta función toma la información de los sensores y genera un archivo JSON con estos datos.

*6) **Verificación de la conexión MQTT**:* Se verifica si el cliente MQTT (client) está conectado al broker. Si no está conectado, se llama a la función "connectMQTT()" para intentar restablecer la conexión. Esta reconexión asegura la capacidad continua de publicar datos. Luego, se llama a client.loop(), lo cual es necesario para mantener la conexión MQTT activa y gestionar la recepción de mensajes.

Se actualiza el cliente NTP (timeClient) para que de la hora actual desde un servidor NTP, proporcionando un timestamp que se utilizará para etiquetar las lecturas de los sensores y las alertas, esto permite que los datos tengan la hora actual.

```
if (mySensor.accelUpdate() == 0) {
    aX = mySensor.accelX();
    aY = mySensor.accelY();
    aZ = mySensor.accelZ();
}

if (mySensor.gyroUpdate() == 0) {
    gX = mySensor.gyroX();
    gY = mySensor.gyroY();
    gZ = mySensor.gyroZ();
}
```

A partir de esto, los datos del acelerómetro del sensor MPU9250 (mySensor) se actualizan. Si la actualización resulta exitosa (accelUpdate() == 0), las variables aX, aY y aZ reciben las lecturas de aceleración en los ejes X, Y y Z. La función detectFall(aX, aY, aZ) se usa para evaluar de si los valores de aceleración señalan una posible disminución. Si se identifica un descenso, las variables alertType y alertMessage se actualizan con datos que señalan un estado crítico. Además, el monitor serié, emite un mensaje para señalar la detección "CAÍDA DETECTADA", se envía una notificación al bot por medio de la función sendTelegramAlert.

```
if (detectFall(aX, aY, aZ)) {
    // Código dentro del if
    alertType = "caida_detectada";
    alertMessage = "Estado critico";
    sendTelegramAlert("Alerta,
    se ha generado una caida");
}

userJson =
createJson_user_data(userId,
                     userName,
                     location,
                     gender);
sensorJson
= createJson_sensor_data(aX, aY,
                         aZ, gX,
                         gY, gZ);
alertJson
= createJson_alerts_data(alertType,
                         alertMessage,
                         timestampStr);

sendThings(alertJson,
"hospital/patient/info/events");
alertJson = "";
sendThings(userJson,
"hospital/patient/info");
userJson = "";
sendThings(sensorJson,
```

```
"hospital/patient/info/sensor_data");
sensorJson = "";
```

Una vez detectada una caída podemos proceder con las alertas, por ende la idea es preparar la información ”Creación de datos JSON”, se generan cadenas JSON para los datos del usuario, sensores, y alertas con las funciones ”createJson_user_data()”, ”createJson_sensor_data()”, y ”createJson_alerts_data()”. Estas cadenas JSON se utilizan para empaquetar la información de forma estructurada antes de enviarla a los servidores.

Y así finalmente, la función ”sendThings()” transmite los datos JSON producidos mediante MQTT hacia los tópicos determinados. Primero se transmiten la información de las alertas, debido a que es la prioridad, después se realiza la información del usuario y, por último, las métricas de la movilidad. Los tópicos establecidos (userMqttTopic, sensorMqttTopic y alertMqttTopic) se emplean para estructurar y distinguir el tipo de información que se está transmitiendo.

B. Nodo 2

C. Aplicación local

D. Configuración Bot de Telegram:

Finalmente, se realiza la configuración del bot de Telegram, para esto, como se puede ver indagar en la funcion, se usan el token y el chat Id obtenido directamente desde Telegram, estas se establecen como constantes para establecer la conexión. Después de comprobar conexión a Wifi, se prepara el URL de solicitud a Telegram que básicamente concatena partes de un link con el token y chat Id creando una URL para interactuar con la API de Telegram. Luego, se realiza la conexión HTTP: *http.begin(url);* Después, se realiza la solicitud GET a la URL creada y se recibe el código de respuesta y se muestra en consola para comprobar conexión. Antes de liberar recursos, existe el manejo de errores como desconexión Wifi o el error al enviar el mensaje a Telegram.

E. Diagrama de clases de la aplicación

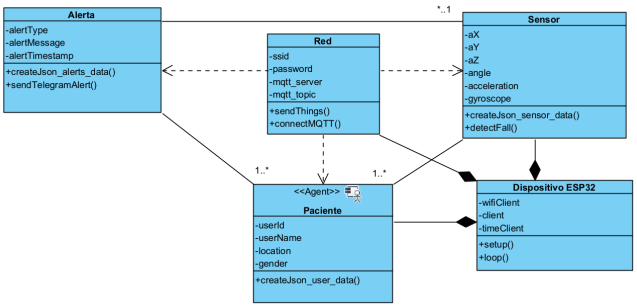

Fig. 6. Diagrama de Clases

X. Documentación Aplicación Java

Para el uso de la aplicación local de escritorio, se pensó en primera instancia en Java, debido a que es un lenguaje cómodo

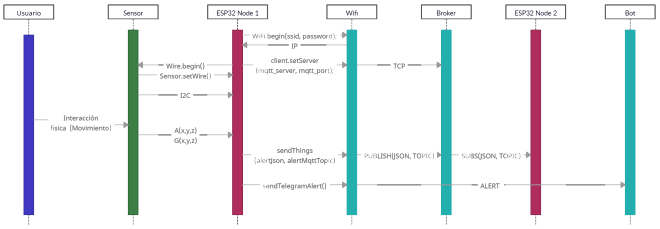

Fig. 7. Diagramas de entidades - Conectividad

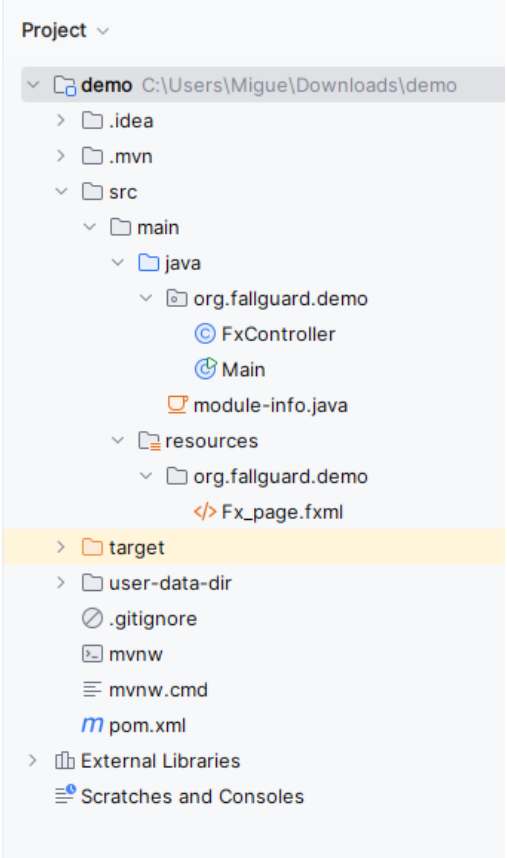

Fig. 8. Proyecto distribución de arvchivos

de usar, y útil en cuanto a su integración multiplataforma, y además que cuenta con gran documentación y es popular en el desarrollo de interfaces gráficas. En nuestro caso, optamos por revisar y profundizar en el desarrollo de una interfaz gráfica y debido a esto se usó las dependencias de JavaFX, las cuales son muy útiles y fáciles de usar si se integran con Scene Builder el cual es un programa que permite poner componentes gráficos sin necesidad de código, para así exportar el panel en una extensión .flxm y después cargarla en nuestra aplicación.

Por ende, lo primero que se realizó fue el crear un proyecto en el IDE IntelliJ, que nos permitiera usar ya JavaFx de manera más sencilla y además con el uso de Maven para importar dependencias por medio del archivo pom.xml, dándonos así la siguiente estructura del proyecto:

Como se puede ver en estas carpetas, se tienen 3 archivos

principales, los 2 primeros, la clase Main y la Clase FX-Controller dentro del paquete Java, y el segundo dentro del paquete resources, el cual corresponde a nuestra interfaz gráfica obtenida por Scene Builder. La clase main con el objetivo de cargar nuestro panel y finalmente lanzarlo, en donde nuestra clase FXController, se encarga de inicializar valores para nuestro frame y hacer el manejo de eventos que ocurran sobre este, y por último la escena Fx_page.fmlx la cual se compone principalmente de unos textos informativos y un Web view que nos permite interactuar con el dashboard.

Realmente la aplicación es bastante sencilla y útil, y su integración con MQTT viene dada por las instrucciones que pueda dar la enfermera por medio de publicaciones hacia el controlador actuador, pues tendrá un campo de texto para poder integrar así la comunicación entre dispositivos.

A partir de ello, podemos la interfaz gráfica se compone de:

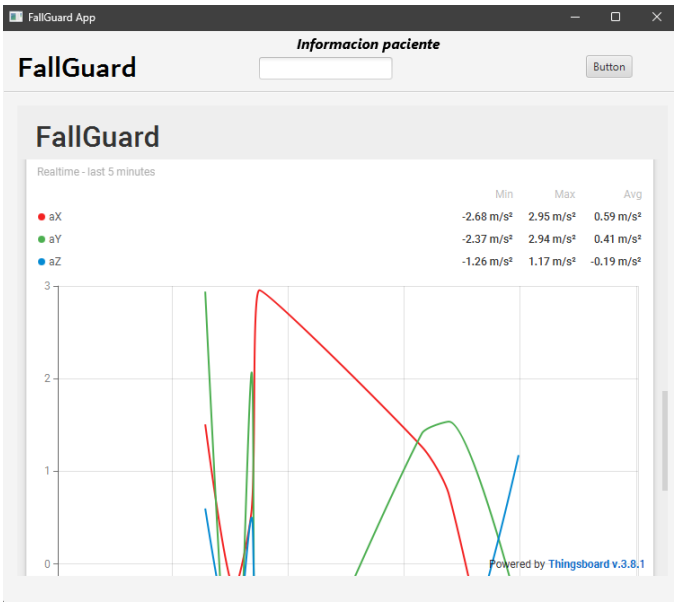

Fig. 9. Interfaz gráfica Java

XI. Pruebas documentadas

Para las pruebas documentadas, se tendrán en cuenta 3 pruebas principales, las cuales constan de la fiabilidad y robustos del proyecto, por ende se intentarán considerar la mayoría de casos posibles, de los casos que podemos encontrar, se definen así las siguientes pruebas, como sustento de estas el video adjuntado será clave para obtener certificado de las pruebas.

A. 1. Detectar una caída y que se genere así la correspondiente activación de las alarmas

De acuerdo a lo establecido, se procedió hacer una prueba en la cual se simulaba un golpe o caída en el que nos agachábamos con gran rapidez, este logro que el sensor percibir una gran aceleración en la caída, lo que implico que se enviaran los JSON hacia los topics definidos, esto hizo que se enviara la alarma y se activaran los actuadores. Para

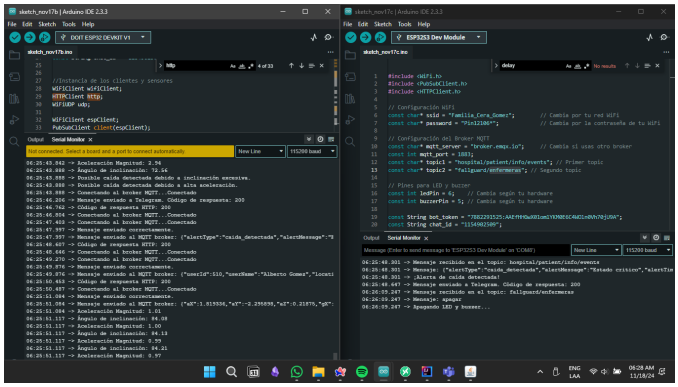

Fig. 10. Prueba 1

las evidencias de pruebas se hace adjunto de la salida de un serial: Además, la salida del broker a la misma hora:

Fig. 11. Salida MQTTX

B. 2. Evitar que se generen alarmas cuando se está únicamente caminado o no se generan movimientos bruscos.

Para esta prueba se simula una caminata, en donde no se presencia ningún cambio repentino de aceleración, lo que hace que la salida únicamente sea la evaluación del ángulo y de la magnitud de la aceleración, pero ninguna sin pasar el umbral.

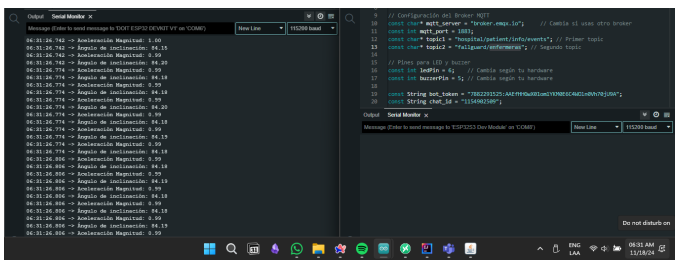

Fig. 12. Prueba 2

C. 3. Realizar la correspondiente desactivación de las alarmas por medio de la aplicación respecto al topic definido.

Para esta última prueba, se garantiza que la aplicación permite hacer uso del protocolo MQTT, debido a que se interviene la publicación por medio del broker, de acuerdo a los mensajes que se adjuntan en el campo de texto. En el caso de poner apagar, se visualiza un cambio en la salida del serial del microcontrolador, correspondiente al mensaje enviado.

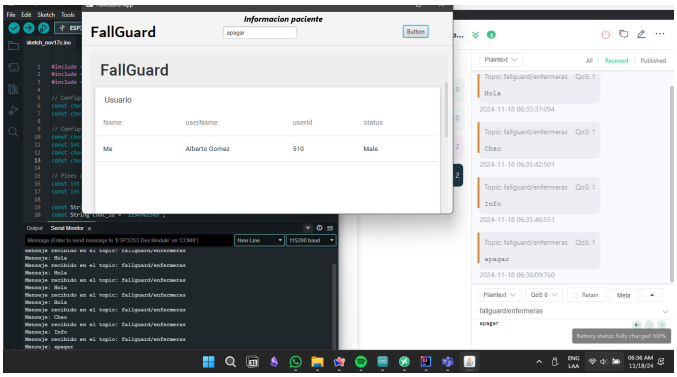

Fig. 13. prueba 3

REFERENCES

[1] M. Manuals, "Caídas en personas mayores," 2024, Último acceso: 3 de noviembre de 2024. [Online]. Available: https://www.msdmanuals.com/es/hogar/salud-de-las-personas-de-edad-avanzada/caídas-en-personas-mayores/caídas-en-personas-mayoresPrevención$_v836686_es$

[2] A. Rakhman, L. Nugroho, W. Widyawan, and K. Kurnianingsih, "Fall detection system using accelerometer and gyroscope based on smartphone," 11 2014.

Finalmente, se adjuntan los códigos y videos realizados, para una mejor intervención y revision del material.