



TECNICATURA SUPERIOR EN Telecomunicaciones

PROYECTO INTEGRADOR I

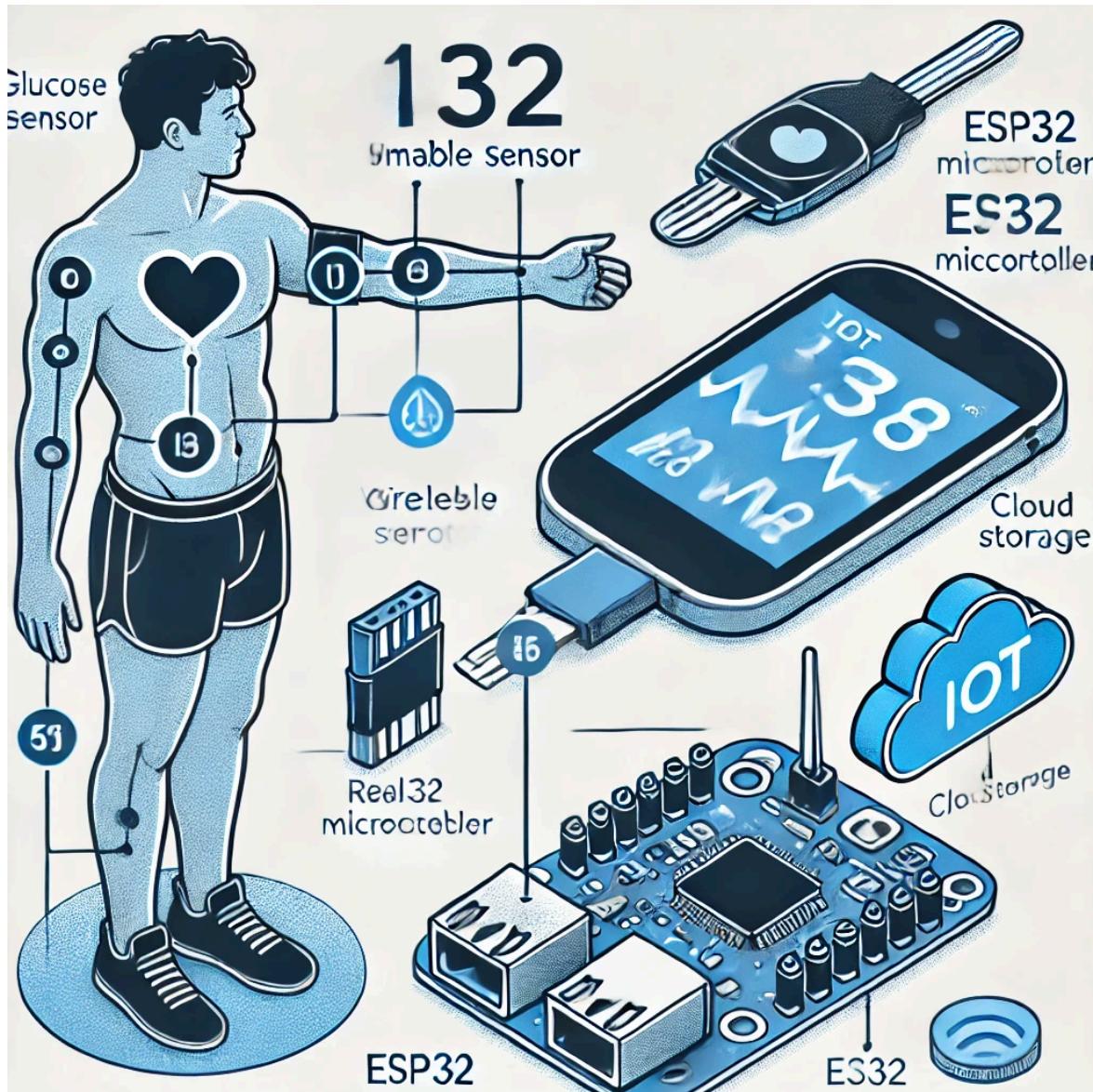
Módulo: Proyecto Final

Medidor de glucosa IOT

Dirección General de
**EDUCACIÓN TÉCNICA Y
FORMACIÓN PROFESIONAL**

Ministerio de
EDUCACIÓN

 **CÓRDOBA**
Seguimos haciendo



ÍNDICE

Prólogo	4
Introducción	6
Estructura del repositorio	7
Stack Tecnológico	9
Metodologías ágiles	11
Capa de percepción	16
Capa de preprocesamiento	12
Capa de conectividad	15
Capa de Almacenamiento	37
Capa de Análisis y aplicación	57
Capa de procesamiento	62
Capa de presentación	94
Conclusión	105

Información General del Curso

- **Institución:** Instituto Superior Politécnico Córdoba (ISPC)
- **Curso:** Proyecto Integrador I
- **Docente:** Cristian Gonzalo Vera

Equipo de Desarrollo Opalo

- **Luciano Lujan** | [GitHub](#)
- **Joaquin Garzón** | [GitHub](#)
- **Vittorio Durigutti** | [GitHub](#)
- **Joaquin Zalazar** | [GitHub](#)
- **Lisandro Juncos** | [GitHub](#)
- **Nahuel Velez** | [GitHub](#)
- **Jose Marquez** | [GitHub](#)
- **Tiziano Paez** | <https://github.com/tpaez>

Prólogo

En un mundo donde la tecnología evoluciona a un ritmo vertiginoso, el IoT se ha consolidado como una herramienta esencial en diversas

áreas, desde la vida cotidiana hasta la gestión de grandes infraestructuras. Dentro del ámbito educativo y práctico de la materia PI, exploramos el potencial de IoT para abordar problemas reales y desarrollar soluciones innovadoras que impacten de forma positiva en la vida de las personas.

Este proyecto nace de la necesidad de mejorar y simplificar ciertos procesos rutinarios a través de la automatización y el monitoreo remoto, con el objetivo de optimizar recursos, aumentar la precisión en la recolección de datos y facilitar la toma de decisiones. La propuesta se presenta de manera global, planteando una solución que no solo satisface una demanda específica, sino que además sienta las bases para futuras implementaciones y mejoras en sistemas de monitoreo inteligente. En este contexto, se entrelazan conceptos clave de IoT con los conocimientos técnicos adquiridos en la materia PI, para diseñar un sistema que responda a desafíos actuales mediante el uso de tecnología avanzada.

A lo largo de este informe, se desarrollan de forma armónica y estructurada los temas esenciales que fundamentan este proyecto, integrando conocimientos teóricos con aplicaciones prácticas. Se brinda

una visión general de los componentes tecnológicos involucrados, los objetivos propuestos y la metodología implementada, todo orientado a resaltar la importancia de IoT en la creación de soluciones eficientes y sostenibles.

Introducción

En el contexto de la salud, la tecnología IoT (Internet de las Cosas) ha facilitado la creación de sistemas avanzados de monitoreo en tiempo real, permitiendo un mejor control y seguimiento de condiciones críticas, como los niveles de glucosa en pacientes diabéticos. Nuestro proyecto, un dispositivo medidor de glucosa basado en IoT, está diseñado para capturar, procesar y transmitir lecturas de glucosa, ofreciendo una herramienta que combina accesibilidad y precisión.

El dispositivo, que emplea un microcontrolador ESP32, un sensor óptico CNY-70 y conectividad en la nube, permite a los usuarios y profesionales de la salud visualizar datos en tiempo real y recibir alertas en casos de niveles críticos. La arquitectura del sistema se basa en una estructura de capas que incluye la percepción, almacenamiento, procesamiento y análisis, cada una de ellas responsable de garantizar la captura y gestión segura y confiable de los datos.

Este documento detalla el desarrollo y la implementación del dispositivo, así como los retos técnicos abordados en cada etapa, como la normalización de datos, la integración del middleware y el uso de algoritmos para la detección temprana de anomalías. La creación de esta herramienta busca ser una base para futuros desarrollos y un

aporte al monitoreo de salud en un contexto de constante innovación tecnológica.

Estructura del Repositorio

- **A** requisitos
- **B** investigación
- **C** prototipo
- **capa_de_analisis:**
 - **app.py**: Archivo principal de la aplicación, donde se inicializa Flask y se registran las rutas API.
 - **db_config.py**: Contiene la configuración de la base de datos MySQL, incluyendo las conexiones y parámetros de acceso.
 - **models.py**: Define los modelos de datos (tablas) para el sistema usando SQLAlchemy.
 - **routes**: Carpeta que contiene todos los Blueprints que manejan las diferentes rutas de la API.

- `lectura.py`: Maneja las lecturas captadas por cada dispositivo.
 - `dispositivos.py`: Gestión de dispositivos registrados en el sistema.
 - `seguridad.py`: Implementa mecanismos de autenticación y autorización.
 - `cliente.py`: Gestiona la información de los usuarios registrados en el sistema.
 - `requirements.txt`: Lista de todas las dependencias del proyecto (incluyendo Flask y SQLAlchemy).
 - `Dockerfile`: Archivo que contiene las instrucciones para construir y desplegar la aplicación en un entorno Docker.
- **D** presentación
 - **E** recursos

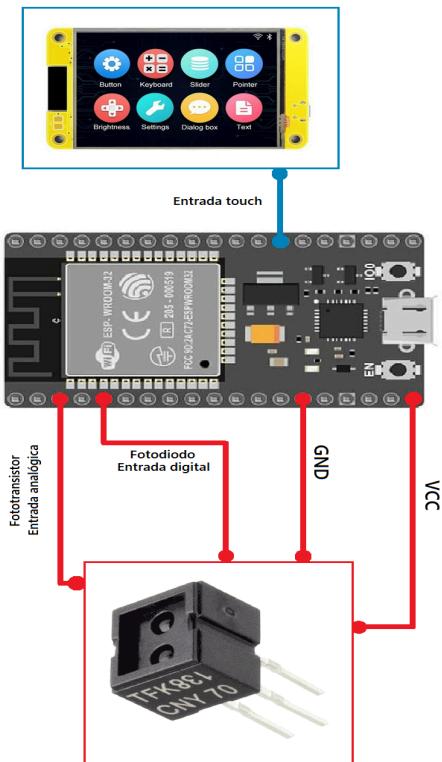
Stack Tecnológico

El stack tecnológico utilizado en este proyecto incluye:

- **Lenguaje de programación:** Python
- **Framework web:** Flask
- **Base de datos:** MySQL
- **ORM:** SQLAlchemy
- **Plataforma de despliegue:** Docker
- **Microcontrolador:** ESP32 (para la conexión IoT)
- **Sensor:** CNY 70 (para la captura de datos de glucosa)
- **Panel táctil:** Para la interacción del usuario con el dispositivo (opcional)
- **Plataforma en la nube:** Conexión a un servidor brindado por el profesor.
 - **Comando:** `ssh opalo@gonaiot.com`
 - **Password:** `opalo`

Diagrama de conexión

Opción para poder optimizar el proyecto, para futuras mejoras en el proyecto.



METODOLOGÍAS ÁGILES

Las metodologías ágiles son un conjunto de enfoques de gestión de proyectos que priorizan la flexibilidad, la colaboración continua y la adaptación rápida ante cambios. Surgieron como una respuesta a los

métodos tradicionales de desarrollo, como el modelo en cascada, que a menudo resultaban rígidos y lentos, especialmente en entornos de desarrollo de software donde la innovación y los requisitos pueden cambiar constantemente.

El enfoque ágil se basa en ciclos de trabajo cortos e iterativos, llamados sprints, en los cuales se planifican, desarrollan, prueban y entregan incrementos funcionales del producto. Al final de cada sprint, el equipo y los interesados revisan el avance y ajustan los planes según los resultados obtenidos, asegurando que el proyecto evolucione de acuerdo a las necesidades reales del cliente.

Entre las metodologías ágiles más populares se encuentran Scrum, Kanban y XP (Extreme Programming), cada una con características específicas para gestionar el trabajo y fomentar la colaboración entre los miembros del equipo. En conjunto, las metodologías ágiles fomentan una cultura de mejora continua y centrada en el cliente, proporcionando valor desde etapas tempranas del proyecto y permitiendo adaptarse rápidamente a los cambios.

Gracias a su flexibilidad y enfoque en el valor, las metodologías ágiles se han vuelto esenciales en sectores de alta innovación y tecnología, facilitando la creación de productos que realmente responden a las necesidades del usuario final.

SCRUM	KANBAN
Roles: <ul style="list-style-type: none">▪ Product Owner (Dueño del Producto)▪ Scrum Master (Facilitador)▪ Development Team (Equipo de Desarrollo)	Principles (Principios): <ul style="list-style-type: none">▪ Visualize Work (Visualizar el Trabajo)▪ Limit WIP (Limitar Trabajo en Proceso)▪ Manage Flow (Gestionar el Flujo)
Events (Eventos): <ul style="list-style-type: none">▪ Sprint (Iteración)▪ Daily Standup (Reunión Diaria)▪ Sprint Planning (Planificación)▪ Review & Retro (Revisión y Retrospectiva)	Board (Tablero): <ul style="list-style-type: none">▪ To Do (Pendiente)▪ In Progress (En Proceso)▪ Testing (Pruebas)▪ Done (Terminado)
Artifacts (Artefactos): <ul style="list-style-type: none">▪ Product Backlog (Lista de Producto)▪ Sprint Backlog (Lista de Sprint)▪ Increment (Incremento)	Metrics (Métricas): <ul style="list-style-type: none">▪ Lead Time (Tiempo de Entrega)▪ Cycle Time (Tiempo de Ciclo)▪ Throughput (Rendimiento)

Roles Principales

Product Owner:

- Representa al cliente y usuarios
- Define prioridades y requisitos
- Acepta o rechaza resultados

Scrum Master:

- Facilita el proceso Scrum
- Elimina obstáculos
- Protege al equipo y el proceso

Equipo Desarrollo:

- Construye el producto
- Auto-organizado
- Multifuncional

Componentes Clave

Sprint:

- Ciclo de trabajo definido
- Entrega valor al final

Product Backlog:

- Lista priorizada de requisitos
- Define el "qué" se va a hacer

Tablero Kanban:

- Visualiza el flujo de trabajo
- Muestra el progreso real

Daily Stand-up:

- Reunión diaria breve
- Sincroniza al equipo

WIP (Trabajo en Proceso):

- Límite de tareas simultáneas
- Evita la sobrecarga

Aplicación SCRUM

Equipo Actual:

- Product Owner: Luciano Lujan
- Scrum Master: Vittorio Durigutti
- Equipo: 6 desarrolladores

Product Backlog:

- Sistema de monitoreo de glucemia
- Sistema de alertas
- Interfaz médico-paciente
- Gestión de datos en tiempo real

Sprints Focalizados en:

- Desarrollo del dispositivo IoT
- Implementación de mediciones
- Sistema de comunicación remota
- Integración con apps móviles

Aplicación KANBAN

Tablero de Trabajo:

- Pendiente: Requerimientos técnicos
- En Proceso: Desarrollo IoT
- Testing: Mediciones de glucosa
- Completado: Diseño inicial

Límites WIP:

- Máx. 2 tareas por desarrollador
- Máx. 3 items en testing

Flujo de Trabajo:

- Prioridad: Funcionalidades críticas
- Desarrollo iterativo del dispositivo
- Testing continuo de precisión

Mejora Continua:

- Revisión semanal de métricas
- Optimización de procesos IoT

ARQUITECTURA

Para el proyecto, utilizaremos una arquitectura IoT de 7 capas, diseñada específicamente para optimizar el flujo de datos y mejorar la eficiencia de los procesos. Esta estructura permite organizar y gestionar las

diferentes etapas del sistema, desde la captura y procesamiento de datos en el dispositivo hasta su análisis y almacenamiento en la nube, asegurando que cada componente funcione de manera eficiente y en coordinación con el resto del sistema.

Capas:

1. Percepcion
2. Preprocesamiento
3. Conectividad
4. Almacenamiento
5. Analisis
6. Procesamiento
7. Presentacion



Capa de Percepción

Sobre el Proyecto

Este repositorio documenta el desarrollo y los componentes de la **Capa de Percepción** del sistema IoT de monitoreo de glucosa. La capa de percepción es la encargada de recolectar datos a partir de sensores en tiempo real, específicamente enfocándose en los niveles de glucosa en sangre. Estos datos son enviados a la capa de almacenamiento para su posterior análisis y monitoreo.

Objetivo Principal

El sistema busca capturar datos de glucosa en tiempo real y procesarlos en una arquitectura que permita el almacenamiento y análisis continuo. La capa de percepción incluye el diseño, configuración e implementación de sensores y controladores (ESP32) para la captura de datos.

Estructura del Repositorio

- 1. A_requisitos:** Documentación de requisitos específicos proporcionados por el docente.
- 2. B_investigacion:** Informes y estudios realizados para comprender y seleccionar los sensores y componentes adecuados.

- 3. C_prototipo:** Implementaciones técnicas de la capa de percepción, incluyendo configuraciones de hardware y software.
- 4. D_presentacion:** Bitácoras de reuniones, grabaciones y presentaciones de progreso del proyecto.
- 5. zassets:** Archivos de recursos gráficos y documentación de apoyo.

Objetivos Semanales

Semana 1 (08/04 - 14/04)

- **Avance:** Introducción al IoT y formación de equipos.
- **Sprint 1:** Como equipo, el objetivo es familiarizarnos con el entorno de desarrollo y las herramientas de colaboración para iniciar eficazmente el proyecto IoT.

Semana 2 (15/04 - 21/04)

- **Avance:** Fundamentos de programación para IoT y primeros pasos con controladores ESP32.
- **Sprint 2:** Comprender los fundamentos de programación aplicados a IoT y empezar a experimentar con los módulos de desarrollo para sentar las bases del prototipo de monitoreo.

Semana 3 (22/04 - 28/04)

- **Avance:** Estudio de sensores en IoT, enfocándose en selección y aplicación.

- **Sprint 3:** Diseñar y desarrollar un módulo inicial en el controlador ESP32 que permita la lectura de datos de sensores y su comunicación básica con los actuadores.

Semana 4 (29/04 - 05/05)

- **Avance:** Taller práctico sobre el uso de sensores y actuadores en el sistema.
- **Sprint 4:** Finalizar el desarrollo y configuración del controlador, implementando protocolos de comunicación y asegurando una integración efectiva con los sensores y actuadores.

Capa de preprocessamiento

Objetivo

Reducir la latencia de datos enviados por el medidor de glucosa, optimizar el uso de ancho de banda y mejorar la capacidad de respuesta.

¿Que es la capa de preprocessamiento?

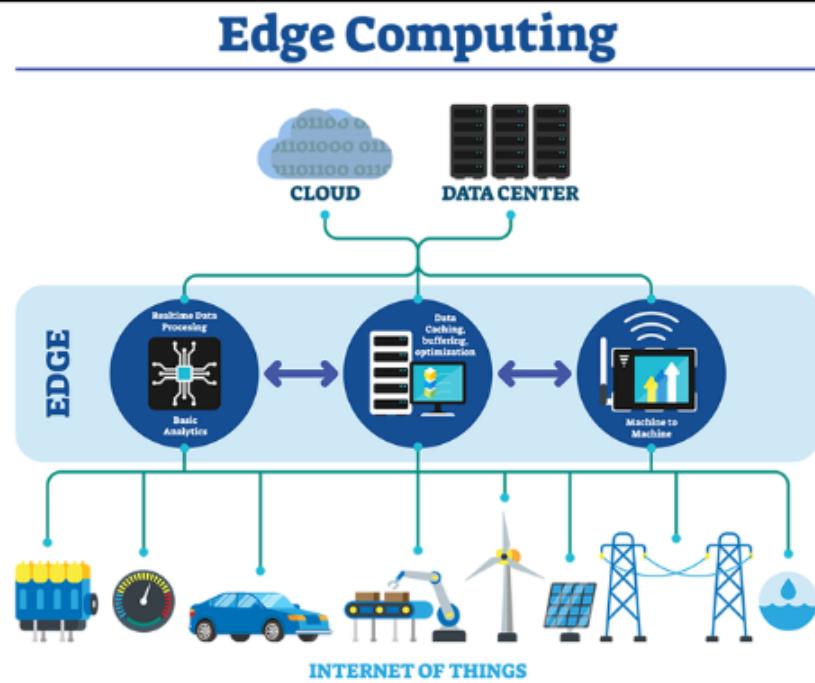
Esta capa se encarga de preparar, filtrar, y procesar los datos capturados por los sensores antes de enviarlos a la nube, a otro dispositivo, o a la capa de almacenamiento y análisis.

Funciones del dispositivo IoT en la capa de preprocessamiento

- Detección de Anomalías
- Filtrado de Señales
- Compresión y Reducción de Datos
- ALMACENAMIENTO TEMPORAL Y CONTROL DE FRECUENCIA
- ALMACENAMIENTO Y LECTURA DE GLUCOSA

Capa de preprocessamiento EDGE computing

El edge computing en un dispositivo IoT se refiere a realizar ciertos análisis y procesamiento de datos en el propio dispositivo, es decir, en el "borde" de la red, sin necesidad de enviar cada dato a un servidor o a la nube.



Capa de preprocessamiento FOG computing

El fog computing en un medidor de glucosa IoT se refiere a una capa de procesamiento de datos intermedia entre el dispositivo (el medidor de glucosa) y la nube. En lugar de procesar todo en el dispositivo (edge) o en la nube, el fog computing permite que los datos se procesen en nodos cercanos, como en gateways locales, routers, o servidores cercanos.

Diferencias Clave entre Fog y Edge Computing

Edge Computing:

- Procesamiento local en dispositivos.
- Baja latencia.
- Tareas específicas y limitadas.

Fog Computing:

- Procesamiento distribuido en nodos intermedios.
- Soporta aplicaciones complejas.
- Mayor capacidad de procesamiento y almacenamiento.

Capa de Conectividad

La capa de conectividad en un proyecto IoT (Internet de las Cosas) desempeña un papel crucial en la comunicación entre los dispositivos y el entorno en el que operan.

Las funciones principales de esta capa son:

1. Comunicación de Datos

Transmisión de Información: Facilita la transmisión de datos entre los dispositivos IoT (sensores, actuadores, etc.) y otros sistemas, como servidores o plataformas en la nube.

Protocolos de Comunicación: Utiliza diversos protocolos de comunicación (como MQTT, HTTP, CoAP, etc.) para garantizar que los datos se envíen y reciban de manera efectiva.

2. Interoperabilidad

Compatibilidad entre Dispositivos: Permite que diferentes dispositivos y tecnologías se comuniquen entre sí, independientemente de sus fabricantes o arquitecturas.

Adaptación a Diferentes Redes: La capa de conectividad puede adaptarse a diversas redes (Wi-Fi, LoRa, Zigbee, etc.), facilitando la integración de dispositivos en diferentes entornos.

3. Gestión de Conexiones

Establecimiento y Mantenimiento de Conexiones: Se encarga de establecer y mantener conexiones entre dispositivos y redes, gestionando aspectos como la autenticación y la seguridad.

Manejo de Desconexiones: Implementa mecanismos para manejar desconexiones y reconexiones, asegurando que los dispositivos puedan recuperar la comunicación después de perderla.

4. Seguridad

Cifrado de Datos: Proporciona mecanismos de cifrado para proteger la información transmitida entre dispositivos y servidores, garantizando la confidencialidad e integridad de los datos.

Autenticación y Autorización: Implementa procesos de autenticación para asegurar que solo dispositivos autorizados puedan comunicarse dentro de la red.

5. Optimización del Rendimiento

Reducción de Latencia: Optimiza el tiempo de respuesta en la comunicación, lo que es esencial para aplicaciones en tiempo real.

Gestión del Ancho de Banda: Controla el uso del ancho de banda, asegurando que la comunicación sea eficiente y no consuma recursos innecesarios.

6. Escalabilidad

Soporte para Múltiples Dispositivos: Permite la conexión y gestión de un gran número de dispositivos, lo que es fundamental en implementaciones de IoT a gran escala.

Adaptación a Cambios: Facilita la adaptación a nuevas tecnologías y dispositivos a medida que el sistema evoluciona.

La capa de conectividad es esencial para el funcionamiento de un sistema IoT, ya que asegura que los dispositivos puedan comunicarse de manera efectiva y segura. Sin esta capa, los dispositivos IoT no podrían intercambiar información, lo que limitaría su funcionalidad y utilidad en aplicaciones del mundo real.

¿Cómo se establece la comunicación?

La comunicación en un proyecto IoT se establece a través de varios métodos y protocolos, dependiendo de los requisitos específicos del sistema, la topología de la red y las capacidades de los dispositivos involucrados.

A continuación, se explican las formas más comunes en que se establece la comunicación en IoT:

1. Protocolos de Comunicación

Los protocolos son fundamentales para la comunicación en IoT. Algunos de los más utilizados incluyen:

- MQTT (Message Queuing Telemetry Transport)

- Ø **Función:** MQTT es un protocolo ligero basado en un modelo de publicación-suscripción, lo que lo hace ideal para dispositivos IoT que requieren una transmisión eficiente y rápida de datos.
- Ø **Aplicación:** En el sistema de monitoreo de glucosa, el ESP32 puede enviar los datos del sensor CNY70 y de las lecturas de glucosa a un servidor MQTT, que distribuye la información a los dispositivos autorizados, como el smartphone del usuario o la plataforma en la nube.
- Ø **Ventajas:** La naturaleza ligera de MQTT minimiza el uso de ancho de banda y el consumo de energía, lo cual es beneficioso en dispositivos portátiles de salud.

- HTTPS (Hypertext Transfer Protocol Secure)

- Ø **Función:** HTTPS garantiza que la transmisión de datos entre el dispositivo y el servidor está cifrada y segura.

- Ø **Aplicación**: Cuando el ESP32 se conecta a la nube a través de Wi-Fi, HTTPS se utiliza para proteger los datos de salud del usuario, cumpliendo con los estándares de privacidad necesarios para aplicaciones médicas.
 - Ø **Ventajas**: Proporciona una capa de seguridad adicional que protege los datos en tránsito y previene el acceso no autorizado.
- **CoAP (Constrained Application Protocol) (alternativa a MQTT):**
- Ø **Uso**: Protocolo de comunicación eficiente basado en UDP, ideal para dispositivos con restricciones de recursos.
 - Ø **Ventaja**: Consume menos ancho de banda que MQTT y es adecuado para redes de baja velocidad como LPWAN.
 - Ø **Aplicaciones**: Comunicación en áreas con conectividad limitada, donde el ancho de banda y el consumo de energía deben ser mínimos.

2. Topologías de Red

La forma en que los dispositivos se conectan también afecta la comunicación:

- Ø **Redes en Estrella**: Todos los dispositivos se conectan a un nodo central (como un hub o un router). Es fácil de gestionar y escalar.
- Ø **Redes en Malla**: Los dispositivos se comunican entre sí directamente, lo que mejora la redundancia y la cobertura de la red.
- Ø **Redes Ad Hoc**: Dispositivos que se conectan temporalmente y se comunican entre sí sin una infraestructura fija.

3. Conexiones Físicas

La comunicación puede establecerse a través de diferentes tipos de conexiones físicas:

- Ø **Wi-Fi**: Conexión a través de redes inalámbricas, ideal para dispositivos que requieren un alto ancho de banda.

Características:

Protocolo estandar: Soporta 802.11 b/g/n.

Modos de Operación:

- Estación (STA)**: Permite conectar el ESP32 a redes Wi-Fi existentes.

- Punto de Acceso (AP):** Permite que el ESP32 actúe como un punto de acceso, permitiendo que otros dispositivos se conecten a él.
- AP+STA:** Permite que el ESP32 funcione simultáneamente como un punto de acceso y como cliente de red.

Rango de Frecuencia: Opera en la banda de 2.4 GHz.

Seguridad: Soporta protocolos de seguridad como WPA/WPA2 para proteger las conexiones.

Capacidad de Conexión: Puede manejar múltiples conexiones simultáneas (hasta 10 dispositivos en modo AP).

Ø Bluetooth/Bluetooth Low Energy (BLE): Utilizado para comunicaciones de corto alcance, ideal para dispositivos portátiles y de bajo consumo.

Versión: Soporta Bluetooth v4.2, incluyendo BLE.

Características

Bajo Consumo de Energía: BLE está diseñado para consumir poca energía, lo que lo hace adecuado para dispositivos portátiles y aplicaciones que requieren una larga duración de la batería.

Rango de Comunicación: Generalmente hasta 100 metros en la teoría, dependiendo del entorno. Siendo el alcance

efectivo de 10 a 30 metros, esto se debe al entorno del dispositivo.

Ø **LoRa (Long Range):** Protocolo diseñado para comunicaciones de largo alcance con bajo consumo de energía, ideal para aplicaciones en áreas rurales.

Integración

Módulos Externos: Aunque el ESP32 no tiene LoRa integrado de forma nativa, puede conectarse a módulos LoRa (como el SX1278 – 433 MHz) para habilitar la comunicación LoRaWAN.

Tabla de referencia módulo LoRa SX1278

Característica	Descripción
Frecuencia de Operación	433 MHz
Modulación	LoRa, FSK
Alcance	Hasta 15 km (en condiciones óptimas)
Potencia de Transmisión	Ajustable hasta +20 dBm
Sensibilidad	Hasta -148 dBm (con LoRa en baja tasa de datos)
Consumo de Energía	Modo de transmisión: 10-120 mA

	Modo de recepción: 10-15 mA
	Modo de sueño: < 1 µA
Interfaz	SPI (Serial Peripheral Interface)
Voltaje de Operación	1.8 V a 3.7 V
Dimensiones	Aproximadamente 20 mm x 20 mm (varía según fabricante)
Aplicaciones Comunes	Redes de sensores, sistemas de monitoreo remoto, IoT

4. Capas de Comunicación

La comunicación en IoT se puede dividir en varias capas:

- ❖ **Capa de Percepción:** Involucra sensores y actuadores que recogen datos del entorno y ejecutan acciones.
- ❖ **Capa de Red:** Se encarga de la transmisión de datos entre dispositivos y servidores.
- ❖ **Capa de Aplicación:** Procesa y utiliza los datos transmitidos, permitiendo la interacción del usuario y la toma de decisiones

5. Seguridad en la Comunicación

La seguridad es fundamental en la comunicación IoT. Esto incluye:

- ❖ **Cifrado:** Utilizar protocolos como TLS/SSL para cifrar la comunicación entre dispositivos y servidores.
- ❖ **Autenticación:** Asegurar que solo dispositivos autorizados puedan comunicarse, utilizando métodos como tokens o certificados digitales.

La forma en que se establece la comunicación en un proyecto IoT es multifacética y depende de varios factores, incluidos los protocolos elegidos, la topología de la red y las características de los dispositivos. Elegir la estrategia adecuada es crucial para garantizar la eficiencia, seguridad y escalabilidad del sistema.

Protocolos de Comunicación (Anexo)

Aunque en este proyecto no se utilicen, es necesario que se anexen en este informe los protocolos de comunicación más utilizados en dispositivos electrónicos y sistemas embebidos, como son SPI, I2C y UART.

1. SPI (Serial Peripheral Interface)

Descripción: SPI es un protocolo de comunicación serial sincrónica que permite la transferencia de datos entre un microcontrolador y uno o más dispositivos periféricos. Utiliza múltiples líneas para la transmisión de datos, lo que permite una comunicación rápida.

Características:

Líneas de Comunicación: Utiliza cuatro líneas principales:

MISO (Master In Slave Out): Línea para datos del esclavo al maestro.

MOSI (Master Out Slave In): Línea para datos del maestro al esclavo.

SCK (Serial Clock): Línea de reloj generada por el maestro.

SS (Slave Select): Línea que selecciona el dispositivo esclavo activo.

Velocidad: Puede alcanzar altas velocidades de transferencia de datos (varios Mbps).

Topología: Conexión maestro-esclavo, donde un maestro puede controlar múltiples esclavos.

Ventajas:

Alta velocidad de comunicación.

Simplicidad en la implementación.

Soporta múltiples dispositivos en un solo bus.

Desventajas:

Requiere más pines en comparación con otros protocolos.

No tiene un mecanismo de control de errores incorporado.

2. I2C (Inter-Integrated Circuit)

Descripción: I2C es un protocolo de comunicación serial síncrono que permite la conexión de múltiples dispositivos en un solo bus utilizando solo dos líneas. Es comúnmente utilizado para la comunicación entre microcontroladores y sensores o dispositivos de memoria.

Características:

Líneas de Comunicación: Utiliza dos líneas:

SDA (Serial Data Line): Línea para la transferencia de datos.

SCL (Serial Clock Line): Línea de reloj para sincronizar la transferencia de datos.

Direcciones: Cada dispositivo en el bus tiene una dirección única, lo que permite que el maestro seleccione el esclavo con el que desea comunicarse.

Velocidad: Soporta varias velocidades, incluyendo 100 kHz (modo estándar) y 400 kHz (modo rápido).

Ventajas:

Requiere solo dos pines, lo que ahorra espacio en el diseño de circuitos.

Permite la conexión de múltiples dispositivos en un solo bus.

Soporta comunicación bidireccional.

Desventajas:

Menor velocidad en comparación con SPI.

Puede ser más complejo de implementar debido a la gestión de direcciones.

3. UART (Universal Asynchronous Receiver-Transmitter)

Descripción: *UART es un protocolo de comunicación serie asíncrono que se utiliza para la comunicación entre dos dispositivos. A diferencia de SPI e I2C, UART no requiere una señal de reloj, lo que simplifica la conexión entre dispositivos.*

Características:

Líneas de Comunicación: *Utiliza al menos dos líneas:*

TX (Transmitter): *Línea para enviar datos.*

RX (Receiver): *Línea para recibir datos.*

Configuración: *Requiere configuración de parámetros como la velocidad de baudios, paridad y número de bits de datos.*

Comunicación: *Puede ser full-duplex, permitiendo la transmisión y recepción simultánea de datos.*

Ventajas

Simplicidad en la implementación y uso.

No requiere líneas de reloj, lo que reduce el número de pines necesarios.

Adecuado para largas distancias.

Desventajas:

No soporta múltiples dispositivos en un solo bus sin lógica adicional.

La sincronización de la velocidad de baudios es crucial para la comunicación exitosa.

Cuadro comparativo de los protocolos de comunicación antes mencionados:

Característica	SPI	I2C	UART
Tipo de Comunicación	Serial síncrono	Serial síncrono	Serial asíncrono
Líneas de Comunicación	4 (MISO, MOSI, SCK, SS)	2 (SDA, SCL)	2 (TX, RX)
Topología	Maestro-esclavo	Maestro-esclavo	Punto a punto
Velocidad	Hasta 10 Mbps o más	100 kHz (estándar), 400 kHz (rápido)	Hasta 1 Mbps (dependiendo de la configuración)
Número de Dispositivos	Múltiples (requiere SS)	Múltiples (cada uno con dirección única)	Limitado a 2 (TX/RX)
Control de Errores	No	No (pero puede implementar ACK)	No
Complejidad de Implementación	Moderada	Alta (gestión de direcciones)	Baja
Distancia de Comunicación	Corta (varía según el entorno)	Corta (varía según el entorno)	Larga (mejor para distancias)

PROTOCOLOS UTILIZADOS EN NUESTRO

PROYECTO:

¿Por qué utilizar HTTPS sobre MQTT?

Utilizar HTTPS sobre MQTT puede tener varias ventajas, dependiendo del contexto y los requisitos de la aplicación, presentamos algunas razones para considerar HTTPS en lugar de MQTT:

1. Seguridad y Cifrado

- Cifrado de Datos: HTTPS utiliza TLS (Transport Layer Security) para cifrar la comunicación entre el cliente y el servidor, lo que protege los datos en tránsito contra interceptaciones.
- Autenticación: HTTPS proporciona autenticación del servidor, lo que ayuda a prevenir ataques de tipo "man-in-the-middle".

2. Compatibilidad con Navegadores

Acceso Web: HTTPS es el protocolo estándar para la web. Si tu aplicación necesita ser accesible a través de navegadores, HTTPS es esencial.

Interoperabilidad: Muchas aplicaciones web y servicios en la nube están diseñados para trabajar con HTTPS, lo que facilita la integración.

3. Facilidad de Implementación

Configuración sencilla: En algunos casos, configurar un servidor HTTPS puede ser más sencillo que implementar un broker MQTT, especialmente si ya tienes un entorno web establecido.

Uso de Infraestructura existente: Si ya tienes servidores web configurados, puedes aprovechar esta infraestructura para manejar la comunicación sin necesidad de configurar un broker MQTT adicional.

4. Estándar de la Industria

Adopción generalizada: HTTPS es ampliamente adoptado y se considera una práctica estándar para la seguridad en la web. Esto puede facilitar la aceptación y confianza en tu aplicación.

5. Limitaciones de MQTT

Conexiones Persistentes: MQTT está diseñado para conexiones persistentes, lo que puede no ser necesario para todas las aplicaciones. Si no necesitas mantener una conexión abierta, HTTPS puede ser más eficiente.

Requisitos de Recursos: MQTT puede requerir un broker y una infraestructura adicional, lo que puede ser un factor a considerar en entornos con recursos limitados.

Adicionalmente se confecciona esta tabla con las diferencias entre ellos:

Aspecto	MQTT	HTTPS
1. Seguridad y Cifrado	- Cifrado opcional, depende de la configuración. - Protocolo ligero, pero puede ser vulnerable sin medidas adicionales.	- Utiliza TLS para cifrar la comunicación. - Proporciona autenticación del servidor, previniendo ataques "man-in-the-middle".
2. Compatibilidad con Navegadores	- No es compatible con navegadores directamente. - Requiere un cliente específico para acceder.	- Protocolo estándar para la web. - Accesible desde cualquier navegador, facilitando la interoperabilidad.
3. Facilidad de Implementación	- Requiere un broker MQTT y configuración adicional. - Puede ser más complejo si no se tiene experiencia.	- Configuración más sencilla si ya se dispone de un entorno web. - Se puede utilizar infraestructura existente.
4. Estándar de la Industria	- Menos adoptado en aplicaciones web convencionales. - Popular en IoT, pero no tan conocido en la web.	- Ampliamente adoptado y considerado un estándar de seguridad en la web. - Genera confianza en los usuarios.
5. Conexiones y Recursos	- Diseñado para conexiones persistentes. - Puede requerir más recursos y mantenimiento.	- Ideal para comunicaciones puntuales. - No requiere mantener una conexión abierta, siendo más eficiente en ciertos casos.

APLICACIÓN A NUESTRO PROYECTO:

Para nuestro proyecto de acuerdo a los componentes de hardware seleccionados, indicados en la capa de percepción, la comunicación se realizará de la siguiente manera:

- Debido a que el sensor CNY70 no utiliza protocolos de comunicación en el sentido convencional como I2C o SPI, ya que es un sensor analógico, la captura y lectura de datos de debe realizar de la siguiente manera:

1. Protocolos de Comunicación del CNY70

Salida Analógica: El CNY70 proporciona una señal analógica que varía según la cantidad de luz reflejada. No se comunica a través de protocolos digitales como UART, I2C o SPI. En su lugar, la salida del sensor es una señal continua que debe ser leída por un pin analógico.

2. Conexión del CNY70 al ESP32

Para conectar el CNY70 al ESP32, se puede utilizar uno de los pines ADC disponibles. El ESP32 tiene varios pines que pueden funcionar como entradas analógicas. Los pines ADC típicos que puedes usar son: GPIO 34 | GPIO 35 | GPIO 32 | GPIO 33 | GPIO 36(VP) | GPIO 39(VN)

3. Conexión de los Pines:

Conecta la salida del CNY70 a uno de los pines ADC del ESP32 (por ejemplo, GPIO 34).

Conecta el VCC del CNY70 a la fuente de alimentación adecuada (generalmente 5V o 3.3V, dependiendo de la configuración).

Conecta el GND del CNY70 a tierra.

4. Lectura de la Señal:

En el código de C++, se utiliza la función analogRead(pin) para leer el valor analógico del pin conectado al CNY70. Este valor se puede procesar para determinar el nivel de glucosa.

Conectividad y Envío de Datos

Elección del Protocolo de Comunicación: Decide cómo enviar los datos leídos a un servidor, base de datos o aplicación.

En nuestro proyecto se optó por utilizar HTTPS para realizar el envío de datos:

HTTP/HTTPS: Para enviar datos a una base de datos o puede ser también a un servidor web.

Implementación: Agrega las bibliotecas necesarias a tu código (por ejemplo, `HTTPClient` para solicitudes HTTP) y configura la conexión Wi-Fi en el ESP32.

Conexión del ESP32 a Wi-Fi:

Se establece una conexión Wi-Fi mediante la configuración de las credenciales de la red (SSID y contraseña). Esto permite que el ESP32 se conecte a Internet y envíe datos a un servidor remoto. La conexión Wi-Fi es esencial para la transmisión de datos en tiempo real y para la interoperabilidad con aplicaciones web y móviles.

Una vez conectado, el ESP32 utiliza el protocolo HTTP para enviar los datos de glucosa a un servidor a través de solicitudes POST. Esto facilita la integración con bases de datos y sistemas de gestión de información.

Configuración de BLE (Bluetooth Low Energy):

El ESP32 se configura como un dispositivo BLE, lo que permite la comunicación con dispositivos cercanos, como teléfonos móviles o tabletas. Esta funcionalidad es útil para aplicaciones donde se requiere una conexión directa y de corto alcance.

A través de BLE, el ESP32 puede enviar notificaciones con los niveles de glucosa a dispositivos conectados, permitiendo un monitoreo en tiempo real sin necesidad de una conexión a Internet.

Implementación de la lógica de envío de datos:

Se desarrolla un algoritmo que determina cuándo y cómo se envían los datos, ya sea a través de Wi-Fi o BLE. Esta lógica permite al sistema adaptarse a diferentes escenarios, como el envío de datos en tiempo real a un servidor o la notificación a un dispositivo móvil cuando se está cerca.

La implementación de esta lógica asegura que los datos se transmitan de manera eficiente y oportuna, optimizando el uso de recursos y mejorando la experiencia del usuario.

Configuración de un servidor para recibir y almacenar datos:

Se establece un servidor web que recibe las solicitudes de datos enviadas por el ESP32. Este servidor puede estar basado en tecnologías como PHP, Node.js o Python, y es responsable de procesar las solicitudes y almacenar los datos en una base de datos (por ejemplo, MySQL).

La base de datos se utiliza para almacenar de forma persistente los niveles de glucosa, permitiendo el análisis posterior y la generación de informes. Además, se pueden implementar medidas de seguridad para proteger la información sensible del usuario.

Interfaz de Usuario (capas superiores)

Visualización: a través de una interfaz web o una aplicación móvil se podrán mostrar los niveles de glucosa en tiempo real.

Alertas: Se considera implementar alertas si los niveles de glucosa están fuera de un rango normal.

Pruebas y Validación

Pruebas de Funcionamiento: Realiza pruebas exhaustivas para asegurar que el sistema funcione correctamente y que los datos sean precisos.

Ajustes: Realiza ajustes en el hardware o el software según sea necesario basado en los resultados de las pruebas.

Documentación

Registro de Proceso: Documenta cada paso del proceso, incluyendo esquemas de conexión, código utilizado y resultados de pruebas.

Capa de Almacenamiento

Sobre el Proyecto

Este repositorio documenta el desarrollo y los componentes de la Capa de Almacenamiento de un sistema IoT para el monitoreo de glucosa en sangre. El proyecto tiene como objetivo ofrecer un almacenamiento seguro y escalable de datos de glucosa capturados en tiempo real, facilitando el análisis y seguimiento médico a través de una base de datos centralizada.

Actualmente, el sistema opera localmente, donde los datos recolectados se almacenan en una base de datos estática. Sin embargo, el proyecto está diseñado para actualizar estos datos en tiempo real una vez que el dispositivo IoT esté completamente operativo, utilizando Flask como backend para gestionar la comunicación con la base de datos.

Objetivo Principal

Desarrollar una infraestructura de almacenamiento de datos robusta y escalable para:

1. **Almacenar lecturas de glucosa** en tiempo real, provenientes de dispositivos IoT.
2. **Gestionar la seguridad y acceso** a los datos, garantizando que solo los usuarios autorizados puedan interactuar con la base de datos.
3. **Facilitar el análisis de datos** a través de herramientas de visualización y análisis que soporten una mejor comprensión de los patrones de glucosa.

Componentes Principales

- **app.py**: Archivo central de la aplicación Flask, en el cual se definen las rutas de acceso y la lógica para manejar peticiones HTTP.
- **static/ y templates/**: Directorios utilizados por Flask para almacenar archivos estáticos (CSS, JS) y plantillas HTML.
- **Dockerfile**: Script para crear una imagen de Docker con la configuración completa de la aplicación, optimizando el despliegue y ejecución.

- **docker-compose.yml**: Archivo de configuración que facilita la ejecución de múltiples contenedores Docker, integrando Flask con la base de datos.
- **requirements.txt**: Archivo de dependencias de Python para la correcta ejecución de Flask y los módulos necesarios.
- **Proyecto PI.sql**: Script SQL para la configuración y estructura de la base de datos en MySQL.
- **D_presentacion**: Registro de reuniones de equipo, presentaciones de avance y bitácoras de Scrum.
- **zassets**: Contiene elementos visuales y archivos adicionales requeridos para la documentación.

Stack Tecnológico Común

- **Control de Versiones**: Git y GitHub
- **Metodologías Ágiles**: Scrum y Kanban, para una organización eficiente de los sprints y tareas.
- **Aprendizaje Basado en Proyectos (ABP)**, promoviendo el desarrollo colaborativo.
- **Soporte DevOps**: Incluye integración y despliegue continuo (CI/CD) con la guía del docente.

Selección de Base de Datos SQL para el Proyecto

IoT de Medidor de Glucosa

Las bases de datos son sistemas que permiten almacenar, organizar y recuperar información de manera estructurada. En la actualidad, las bases de datos desempeñan un papel fundamental en numerosas aplicaciones y sectores, desde el comercio electrónico hasta la gestión de la información en empresas y organizaciones.

Las bases de datos SQL y NoSQL son dos enfoques diferentes para almacenar y procesar datos:

Comparativa SQL (MySQL) vs. NoSQL

Característica	SQL (MySQL)	NoSQL
Estructura de Datos	Modelo tabular, con esquema fijo y bien organizado.	Modelo flexible: documentos, clave-valor.
Esquema de Datos	Requiere definir la estructura previamente.	No requiere un esquema fijo.
Transacciones ACID	Sí, garantiza atomicidad, consistencia, aislamiento y durabilidad.	Limitadas; no todos los tipos las soportan.
Consistencia de Datos	Alta, ideal para datos críticos de salud.	Alta disponibilidad, pero a veces a expensas de la consistencia.
Relaciones Complejas	Gestión eficaz de relaciones entre datos.	Menos eficiente para relaciones complejas.
Escalabilidad	Vertical (puede escalar horizontal en algunos casos)	Horizontal, ideal para grandes volúmenes.
Seguridad y Control de Acceso	Altamente seguro y controlado.	Puede tener menos opciones de control de acceso robustas.

Ventajas de SQL

- 1. Estructura y Organización de Datos:** MySQL permite una **organización tabular** de los datos de glucosa, facilitando la administración de información crítica de salud en un entorno controlado.
- 2. Transacciones Seguras (ACID):** Las transacciones en MySQL garantizan que cada operación de guardado o actualización sea segura y completa. Este control es crucial para el monitoreo de la salud, donde los datos deben ser precisos y confiables.
- 3. Consistencia y Relaciones entre Datos:** La base de datos SQL permite gestionar relaciones complejas entre conjuntos de datos (p. ej., niveles de glucosa, horario, actividad física), mientras que mantiene la **consistencia y fiabilidad** de la información.
- 4. Seguridad y Control de Acceso:** MySQL ofrece una seguridad sólida y un control de acceso detallado, adecuado para manejar información sensible de los pacientes de forma segura y conforme a estándares de privacidad.
- 5. Respaldo y Recuperación:** MySQL facilita la creación de **backups periódicos** y recuperación de datos en caso de errores, esencial para un sistema IoT que monitorea datos de salud.

Desventajas de NoSQL en el Contexto del Proyecto

Aunque NoSQL (por ejemplo, MongoDB, Redis) es muy útil en aplicaciones con datos no estructurados y en grandes volúmenes, algunas de sus características presentan desventajas para este sistema:

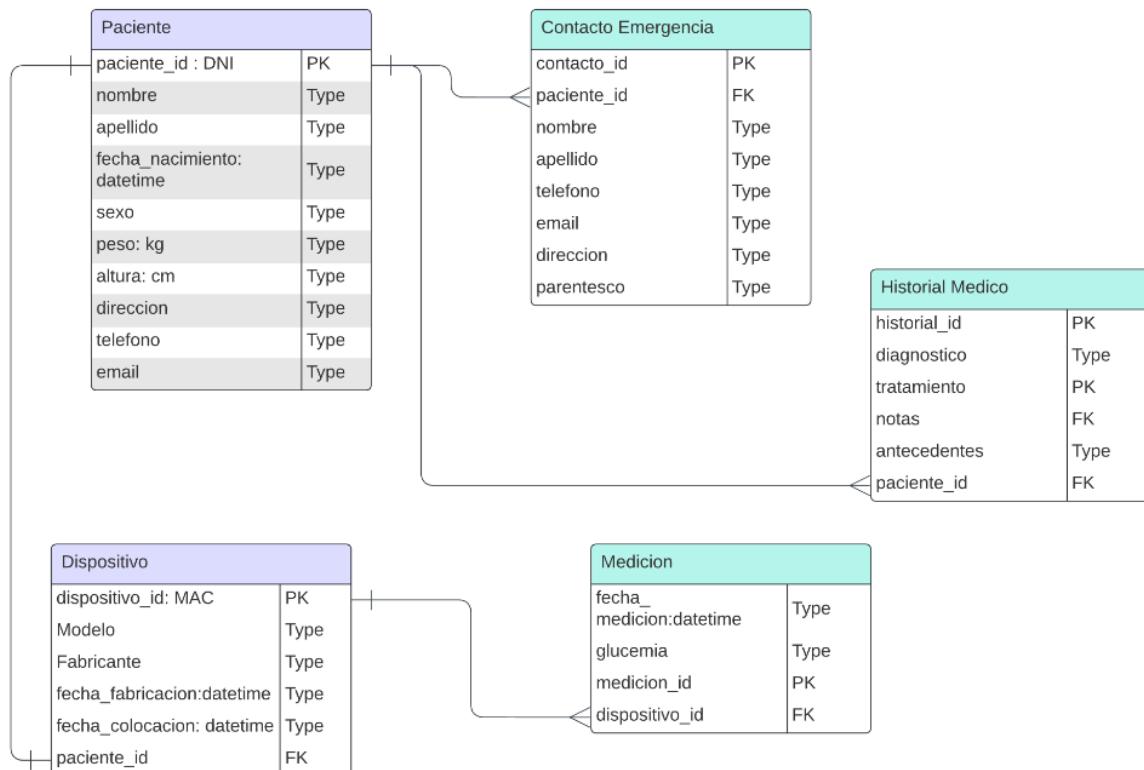
- **Falta de Consistencia Fuerte:** Las bases de datos NoSQL generalmente priorizan la disponibilidad sobre la consistencia, lo cual puede ser un inconveniente para datos críticos de salud.
- **Limitadas Transacciones Complejas:** La mayoría de las bases de datos NoSQL carecen de un modelo de transacción ACID completo, lo cual es esencial para evitar inconsistencias en los datos de glucosa.
- **Relaciones Limitadas entre Datos:** Los sistemas NoSQL son menos eficientes en el manejo de relaciones complejas, algo fundamental para correlacionar los niveles de glucosa con otros datos del paciente.

Justificación de la Elección: MySQL

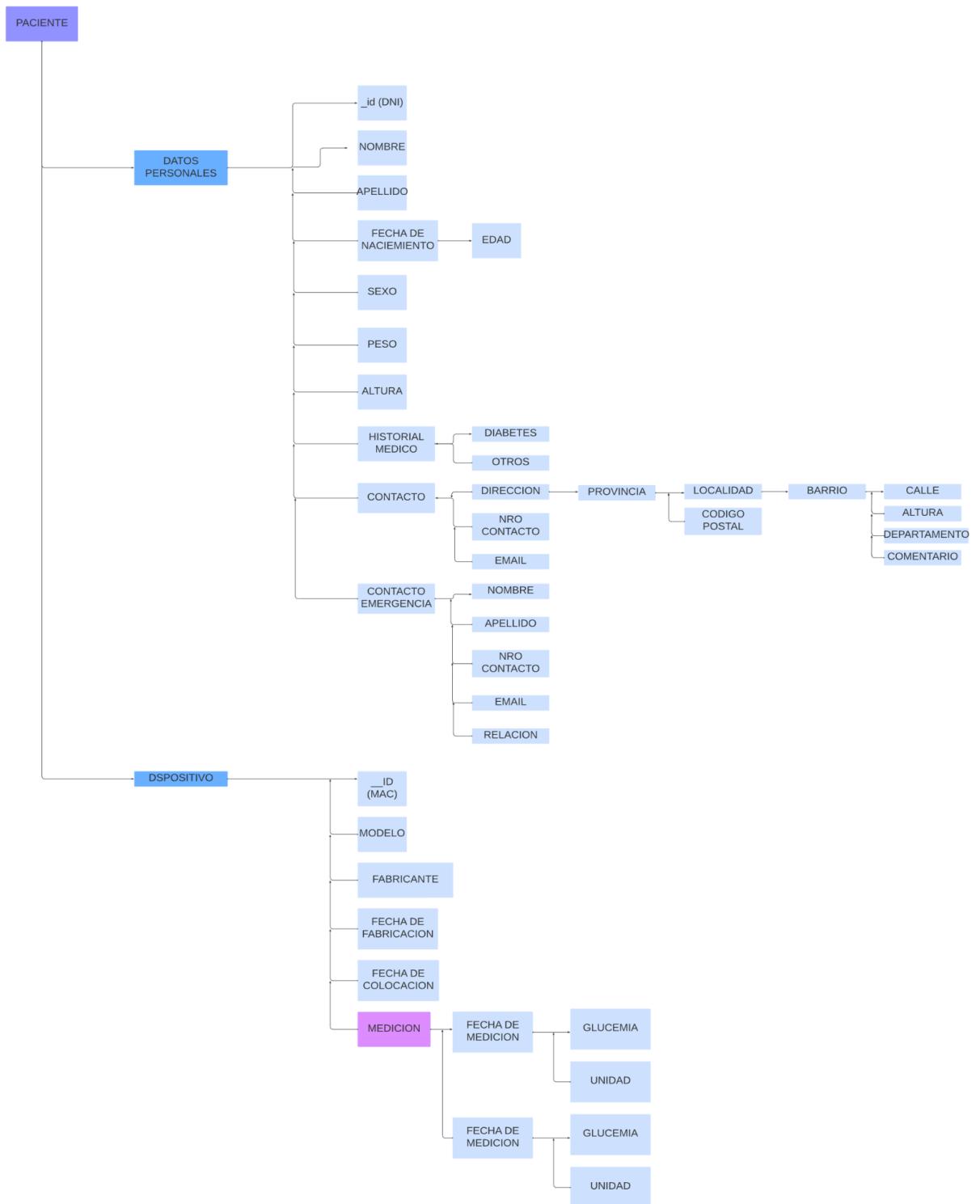
Para nuestro sistema de monitoreo de glucosa IoT, hemos decidido utilizar **MySQL**, una base de datos SQL de código abierto. La elección de MySQL se basa en su compatibilidad con múltiples entornos, su facilidad de uso y su capacidad para manejar una variedad de cargas de trabajo. MySQL es una opción sólida para garantizar la integridad y eficiencia en la gestión de datos en tiempo real, especialmente en aplicaciones de salud como este medidor de glucosa tipo pulsera.

Esquemas

Database ER



Esquema Base de datos NoSQL:



Middleware

Introducción

El desarrollo de dispositivos IoT está en auge, y una parte esencial en este ecosistema es el **middleware**, que actúa como un intermediario entre los dispositivos físicos y las aplicaciones de usuario o los sistemas backend. En este contexto, **Flask** se convirtió en una herramienta popular para implementar middleware debido a su simplicidad, flexibilidad y capacidad para manejar servicios web ligeros.

¿Cuál es su función?

En IoT, su función principal es facilitar la comunicación, gestión y control de los dispositivos conectados a la red. En lugar de que cada dispositivo tenga que comunicarse directamente con el servidor o la aplicación final, el middleware proporciona una capa que abstracta estas interacciones, manejando las complejidades de la comunicación, la seguridad, la integración de datos, y la escalabilidad.

¿Qué facilita el middleware?

- **Gestión de dispositivos:** Registrar, monitorear y actualizar dispositivos de manera remota.
- **Recopilación y almacenamiento de datos:** Capturar los datos enviados por sensores o actuadores, almacenarlos y prepararlos para su análisis.
- **Autenticación y seguridad:** Garantizar que solo dispositivos autorizados puedan enviar o recibir información
- **Interoperabilidad:** Traducir diferentes protocolos y estándares para que dispositivos heterogéneos puedan interactuar entre sí.

¿Cómo lo implementaremos?

Para la capa de procesamiento se debía integrar el middleware a nuestro proyecto de medidor de glucosa.

Los objetivos a lograr en tal capa eran los siguientes:

Unidad 6: Capa de Procesamiento en la Nube

- Fundamentos de procesamiento en la nube.
- Implementación de servicios en la nube para almacenamiento y procesamiento de datos.
- Middleware y adaptación de la capa de almacenamiento para procesamiento avanzado en la nube.
- Integración del procesamiento en la nube en el sistema IoT.

Funcionalidades clave que debe tener el middleware

- **Recepción de datos de los dispositivos Edge:** El middleware debe recibir las lecturas de glucemia y otros datos relevantes de los dispositivos en tiempo real.
- **Transformación y adaptación de datos:** Los datos que llegan desde los dispositivos necesitan ser adaptados antes de ser almacenados o enviados a la API.
 - Convertir los datos de formato hexadecimal o binario a un formato legible (como JSON).
 - Normalizar los valores de las mediciones de glucemia para asegurar que estén dentro de los rangos aceptados.
 - Transformar los datos para almacenarlos en la Base de Datos SQL del proyecto.
- **Validación y autenticación:** Verificar que los datos provengan de dispositivos autorizados, especialmente si se maneja información sensible como la glucemia de pacientes.
- **Gestión de eventos y alertas:** Si la glucemia está fuera de los niveles normales, el middleware puede generar una alerta que luego se envíe al Contacto de Emergencia o al Historial Médico.

- **Interacción con la API RESTful:** Configurar el middleware para que sirva de interfaz entre los datos que recibe y las rutas de la API RESTful. Así, el middleware toma los datos de los dispositivos, los adapta y los envía a la API.

¿Qué logramos implementando un middleware?

En nuestro proyecto de sistema IoT para **monitoreo de glucosa**, el **middleware** cumple funciones clave para asegurar que los datos críticos de los sensores de glucosa lleguen a la **nube** para su **procesamiento y análisis**. Además, este middleware se encarga de **adaptar y normalizar** los datos antes de almacenarlos, verificando que provengan de dispositivos **autenticados y autorizados**, lo cual es fundamental dada la **sensibilidad** de la información médica manejada.

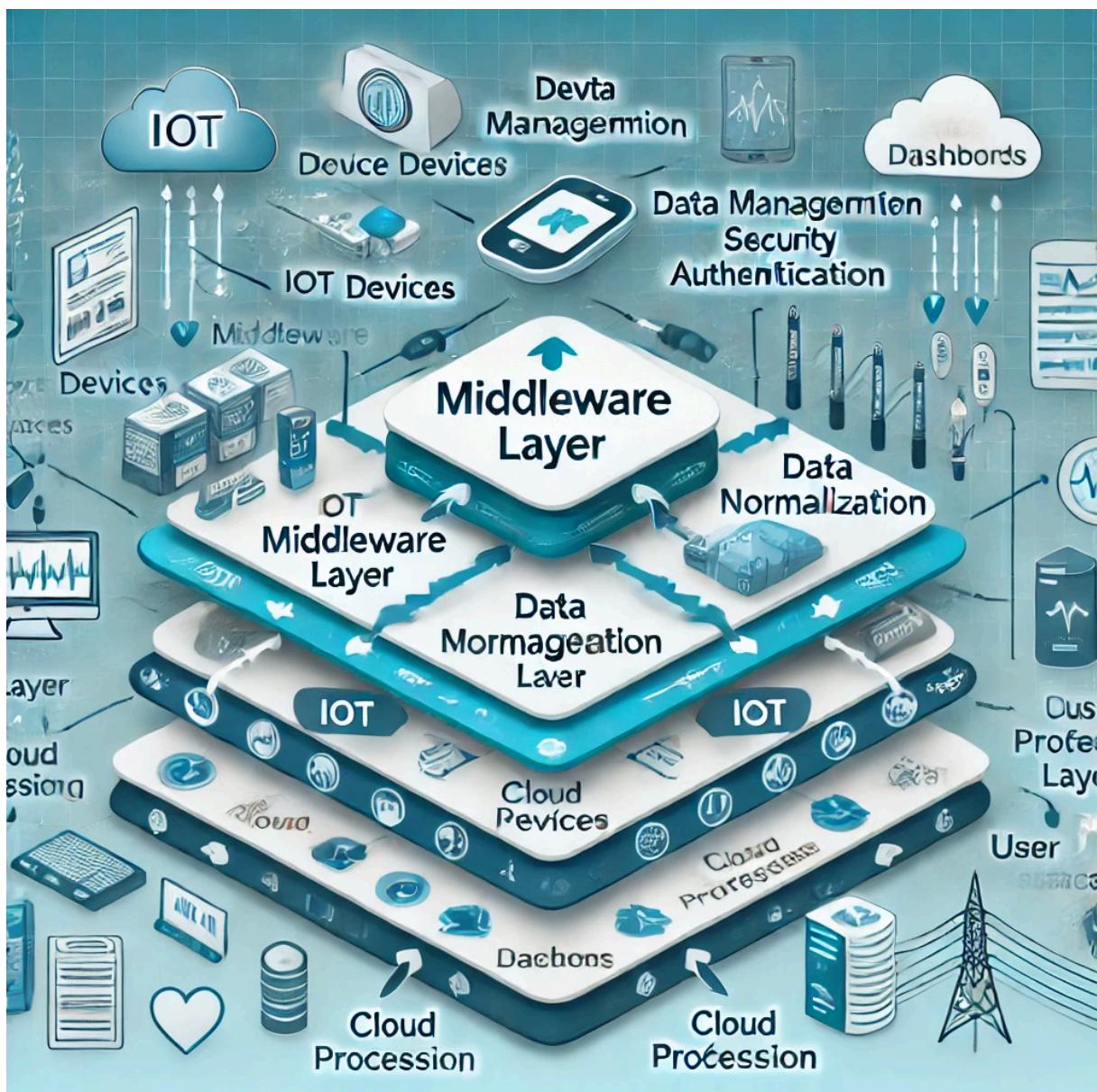
Al integrar middleware en la capa de procesamiento de nuestro sistema, logramos:

Gestión de Dispositivos y Seguridad: Facilitamos el registro y monitoreo de dispositivos en tiempo real.

Normalización y Almacenamiento de Datos: Adaptamos los datos en formatos como JSON para su almacenamiento adecuado y análisis posterior.

Interoperabilidad: Hacemos posible que distintos componentes del sistema, como sensores y la base de datos, se comuniquen fluidamente.

Con esta estructura, nuestro sistema de monitoreo de glucosa no solo captura datos precisos y en tiempo real, sino que también asegura una gestión eficaz de la información y su acceso controlado, fortaleciendo así la confiabilidad y escalabilidad del proyecto.



Funciones

- **Conexión al servidor SSH “gonaiot”:** Esta conexión sería iniciada al arranque del middleware y reutilizada en cada ruta que necesite enviar datos al servidor.
- **Función para registrar Cliente:** Procesa los datos del cliente, valida que el DNI no exista ya en la base de datos e inserta los datos en la tabla Cliente de la base de datos.
- **Función para registrar Dispositivo:** Comprueba si el cliente ya tiene un dispositivo activo, registra el nuevo dispositivo y actualiza el estado del anterior (si lo hubiera) como retirado, luego, enlaza el dispositivo con el cliente mediante su DNI.
- **Función para registrar Lecturas de Glucosa:** Valida que el dispositivo esté activo y que el valor de glucosa sea válido, luego inserta los datos en la tabla Lectura, asociando la lectura con el dispositivo mediante su MAC.

Explicación de la Implementación

1. Dispositivos IoT (Sensores de Glucosa): Los dispositivos de borde o edge (sensores de glucosa) recolectan datos en tiempo real sobre los niveles de glucosa del paciente. Estos datos son enviados hacia el middleware para su procesamiento inicial.

2. Middleware: Esta capa cumple múltiples funciones:

- **Gestión de Dispositivos:** Controla y monitorea los sensores conectados, verificando el estado de cada uno y gestionando sus registros en la red.
- **Normalización y Adaptación de Datos:** Convierte los datos recolectados en formatos estándar (como JSON) y los adapta para ser enviados al almacenamiento.
- **Autenticación y Seguridad:** Garantiza que sólo los dispositivos autorizados envíen datos, aplicando capas de seguridad para proteger la información sensible del paciente.

- **Procesamiento de Alertas y Eventos:** Genera alertas cuando los niveles de glucosa están fuera de los rangos normales, notificando a los usuarios y a los proveedores de salud a través de la interfaz de usuario.

3. Procesamiento en la Nube: Los datos normalizados y autenticados por el middleware son enviados a la nube para su almacenamiento y análisis avanzado. Esto incluye el uso de bases de datos en la nube y análisis de datos históricos para detectar patrones.

4. Interfaz de Usuario: Los datos procesados son accesibles para el usuario final a través de una aplicación que permite a los pacientes y proveedores de salud visualizar los niveles de glucosa, recibir alertas y revisar tendencias históricas.

Este flujo garantiza una arquitectura robusta, en la cual el middleware actúa como un intermediario seguro, eficiente y escalable entre los dispositivos IoT y el almacenamiento y procesamiento en la nube, facilitando así una interacción ágil y confiable para el monitoreo de la salud.

Licencia

MIT License

Copyright (c) [2024] [Sistema IoT para un dispositivo Medidor de Glucosa/Integrates

Equipo Opalo]

Permiso por la presente se concede, sin cargo, a cualquier persona que obtenga una copia de este software y archivos de documentación asociados (el "Software"), para tratar en el Software sin restricción, incluyendo sin limitación los derechos a usar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar y/o vender copias del Software, y a permitir a las personas a quienes se les proporciona el Software hacerlo, sujeto a las siguientes condiciones:

La anterior nota de copyright y esta nota de permiso deberán ser incluidas en todas las copias o porciones sustanciales del Software.

EL SOFTWARE SE PROVEE "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO PERO NO LIMITÁNDOSE A LAS GARANTÍAS DE COMERCIABILIDAD, IDONEIDAD PARA UN PROPÓSITO PARTICULAR Y NO INFRACCIÓN. EN NINGÚN CASO LOS AUTORES O

TITULARES DEL COPYRIGHT SERÁN RESPONSABLES DE NINGUNA RECLAMACIÓN, DAÑOS U OTRA RESPONSABILIDAD, YA SEA EN UNA ACCIÓN DE CONTRATO, AGRAVIO O DE OTRA MANERA, QUE SURJA DE O DE OTRA MANERA EN CONEXIÓN CON EL SOFTWARE O EL USO U OTRAS MANIPULACIONES EN EL SOFTWARE.

13

Referencias Bibliográficas

- Manuales y cuadernillos digitales de Ciencias de la Computación

<https://program.ar/material-didactico/>

- Unidad 1. Fundamentos de Informática SOPORTE LÓGICO EN UN ORDENADOR PERSONAL: EL SOFTWARE

<https://www3.gobiernodecanarias.org/medusa/ecoblog/mgoncal/>

Capa de análisis y aplicación

¿Qué son las capas de análisis y aplicación?

La función principal de la **capa de análisis** en proyectos IoT es **procesar, analizar e interpretar** los datos recolectados por dispositivos, transformándolos en información útil para la toma de decisiones.

La **capa de aplicación**, por otro lado, es la encargada de **interactuar** directamente con el **usuario final**. Esta capa define las **funcionalidades y servicios** que el usuario experimenta y puede incluir interfaces de usuario, funcionalidades específicas del negocio, y APIs que permiten acceder a la **información procesada** por la **capa de análisis**.

Relación entre ambas:

- La **capa de aplicación** consume la información generada por la capa de análisis para ofrecer servicios de valor a los usuarios.
- La **capa de análisis** proporciona el procesamiento necesario para que la capa de aplicación funcione correctamente, asegurando que las aplicaciones IoT ofrezcan información procesada y valiosa.

Implementación en nuestro proyecto

En nuestro proyecto tuvimos que diseñar y desarrollar una **API RESTful** para gestionar datos del proyecto IoT utilizando **Flask** y **MySQL** para que sirviera como puente entre la **Capa de Conectividad** y la **Capa de Almacenamiento**, asegurando la integración efectiva de datos. A partir de esta API gestionamos las operaciones **CRUD** (**Crear**, **Leer**, **Actualizar**, **Eliminar**) para varias entidades almacenadas como: **pacientes**, **dispositivos**, **mediciones**, **historial_medico** y **contacto_emergencia**. Con esto realizado, el programa comienza a recolectar **datos del sensor, fecha y hora** de la recolección y demás datos relevantes para posteriormente almacenarlos en la **base de datos MySQL** del proyecto.

Rutas del proyecto

Ruta 1: Mediciones

- GET /mediciones/: Obtiene todas las mediciones.
- GET /mediciones/int:id: Obtiene una medición específica por ID.
- POST /mediciones/: Agrega una nueva medición.

- PUT /mediciones/int:id: Actualiza una medición existente por ID.
- DELETE /mediciones/int:id: Elimina una medición por ID.

Ruta 2: Pacientes

- GET /pacientes/: Obtiene todos los pacientes.
- GET /pacientes/int:id: Obtiene un paciente específico por ID.
- POST /pacientes/: Crea un nuevo paciente.
- PUT /pacientes/int:id: Actualiza paciente existente por ID.
- DELETE /pacientes/int:id: Elimina un paciente por ID.

Ruta 3: Historial_medico

- GET /historial_medico/: Obtiene todos los historiales.
- GET /historial_medico/int:id: Obtiene un historial específico por ID.
- POST /historial_medico/: Crea un nuevo historial.
- PUT /historial_medico/int:id: Actualiza un historial existente por ID.
- DELETE /historial_medico/int:id: Elimina un historial por ID.

Ruta 4: Dispositivos

- GET /dispositivos/: Obtiene todos los dispositivos.
- GET /dispositivos/int:id: Obtiene un dispositivo específico por ID.
- POST /dispositivos/: Crea un nuevo dispositivo.

- PUT /dispositivos/int:id: Actualiza un dispositivo existente por ID.
- DELETE /dispositivos/int:id: Elimina un dispositivo por ID.

Ruta 5: Contacto_emergencia

- GET /contacto_emergencia/: Obtiene todos los contactos de emergencia.
- GET /contacto_emergencia/int:id: Obtiene un contacto específico por ID.
- POST /contacto_emergencia/: Crea un nuevo contacto de emergencia.
- PUT /contacto_emergencia/int:id: Actualiza un contacto existente por ID.
- DELETE /contacto_emergencia/int:id: Elimina un contacto de emergencia por ID.

¿Cómo se manipulan las rutas?

Se requiere una **clave API** para todas las solicitudes a modo de “Autenticación”. Nuestra clave es “**opalo**”.

Solicitud GET (Leer):

```
curl -H "X-API-KEY: opalo" https://api.gonaiot.com/opalo/usuarios/
```

Solicitud POST (Crear):

```
curl -X POST -H "X-API-KEY: opalo" -H "Content-Type: application/json"  
\ -d '{"nombre": "Juan Perez", "email": "juan.perez@example.com",  
"edad": 30}' \ https://api.gonaiot.com/opalo/usuarios/
```

Solicitud DELETE (Eliminar):

```
curl -X DELETE -H "X-API-KEY: opalo"  
https://api.gonaiot.com/opalo/usuarios/ID_USUARIO
```

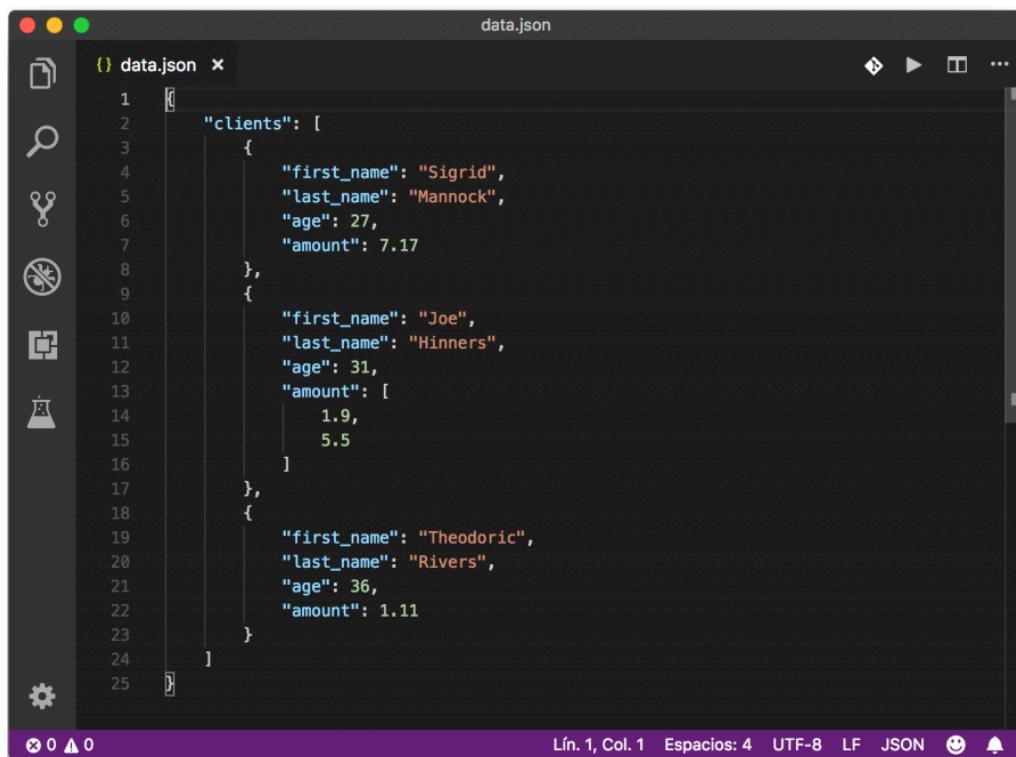
Solicitud PUT (Actualizar):

```
curl -X PUT -H "X-API-KEY: opalo" -H "Content-Type: application/json" \  
-d '{"nombre": "Juan P. Gomez", "edad": 31}' \  
https://api.gonaiot.com/opalo/usuarios/ID_USUARIO
```

Para especificar el parámetro (**POST**, **PUT** o **DELETE**) se debe usar “**-X [Parámetro]**”. Si se van a agregar o actualizar datos, debe especificarse el tipo (Ej: **JSON**) luego en el segundo “**-H**” El “**-d**” se utiliza para incluir los datos.

¿Qué son los archivos JSON?

JSON, que significa JavaScript Object Notation, es un formato de intercambio de datos originalmente derivado de JavaScript. Hoy en día se ha independizado y es ampliamente utilizado en muchos lenguajes de programación para la transmisión de datos. JSON es muy simple en términos de los tipos de datos que soporta: números, cadenas de texto, booleanos (verdadero o falso), arreglos “[]” (o listas), objetos (que son básicamente mapas o diccionarios), y el valor null. Un aspecto importante también es su sintaxis, que es bastante simple y fácil de entender ya que se representan en pares de clave-valor que se delimitan por llaves “{ }”. El uso de JSON en las APIs REST se ha convertido en el formato estándar para el intercambio de datos ya que poseen varias ventajas, entre ellas la facilidad de uso y la capacidad de representar estructuras de datos complejas.



The screenshot shows a code editor window with a dark theme. The title bar says "data.json". The code itself is a JSON object with a single key "clients" which contains an array of three objects. Each object has properties: first_name, last_name, age, and amount. The "amount" property for the second client is an array containing two values: 1.9 and 5.5.

```
data.json
{
  "clients": [
    {
      "first_name": "Sigrid",
      "last_name": "Mannock",
      "age": 27,
      "amount": 7.17
    },
    {
      "first_name": "Joe",
      "last_name": "Hinnens",
      "age": 31,
      "amount": [
        1.9,
        5.5
      ]
    },
    {
      "first_name": "Theodoric",
      "last_name": "Rivers",
      "age": 36,
      "amount": 1.11
    }
  ]
}
```

Comparación con otros formatos de intercambio de datos

- **XML vs JSON:** XML es un lenguaje de marcado que almacena información en una estructura anidada y etiquetada, lo que puede dar lugar a archivos voluminosos y más complejos de analizar. JSON es un formato de datos ligero que utiliza una sintaxis parecida a los objetos de JavaScript, siendo más sencillo de escribir y leer. No requiere el cierre de etiquetas, reduciendo así el tamaño del archivo.
- **CSV vs JSON:** CSV es un formato simple que almacena datos tabulares. Es útil para grandes conjuntos de datos, pero carece de la capacidad de representar estructuras complejas o anidadas. JSON puede representar datos complejos con varias capas de anidación, siendo más versátil, pero puede resultar en archivos más grandes para conjuntos de datos de tamaño considerable.

XML y CSV tienen sus usos según su contexto. XML es útil cuando se necesita un lenguaje de marcado fuerte y estructurado, mientras que CSV es una excelente opción para datos tabulares. La elección del formato de intercambio de datos depende en gran medida de las necesidades y restricciones específicas del proyecto.

```

<?xml version="1.0" encoding="UTF-8"?>
<listadeclientes>
    <cliente>
        <numerodecliente>12345</numerodecliente>
        <Nombre>Luis García</Nombre>
        <Direccion>
            <Calle>Calle de la Princesa</Calle>
            <Ciudad>Madrid</Ciudad>
            <Codigo Postal>28020</Codigo Postal>
            <Pais>España</Pais>
        </direccion>
    </cliente>
</listadeclientes>

```

Estructura de archivo XML

ID	Producto	Presentación	Precio	Cantidad
1	Maestro Limpio	Botella de 500ml	25.50	30
2	Pato Purific	Pastilla desodorante	27.00	20
3	Axión Lavatrastes	Bolsa de 250gr	12.00	25
4	Colgate Maxi White	Tubo de 230gr	24.50	40
5	Blanqueador Cloralex	Botella de 1lt	33.75	25

Estructura de archivo CSV

REST (Representational State Transfer)

REST es un estilo de arquitectura de software para sistemas hipermedia distribuidos, como la web. no es un estándar o protocolo, sino un conjunto de restricciones o principios que, cuando se siguen, permiten construir sistemas que escalan a las necesidades de la web. Uno de los principios clave de REST es la "interfaz uniforme", lo que significa que todas las interacciones con recursos en un sistema RESTful deben ser realizadas de una manera consistente.

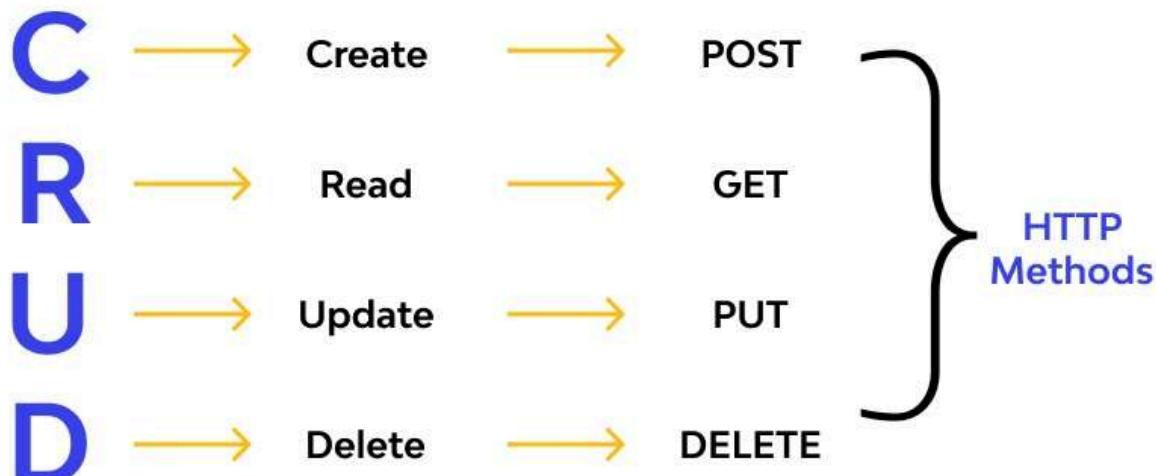
API RESTful

Es importante notar que aunque REST se basa en HTTP, no está limitado a él.

Puedes implementar REST sobre cualquier protocolo siempre y cuando puedas satisfacer sus restricciones. Esto ha llevado a la aparición de sistemas RESTful sobre protocolos como MQTT en el espacio de IoT. Sin embargo, la mayoría de las APIs RESTful en uso hoy en día se implementan sobre HTTP.

CRUD: ¿Qué es?

CRUD es un acrónimo que proviene de las palabras en inglés: Create (crear), Read (leer), Update (actualizar) y Delete (eliminar). Estas son las cuatro operaciones básicas que se pueden realizar en la mayoría de las aplicaciones que manejan algún tipo de persistencia de datos, y son fundamentales en el diseño de APIs, especialmente las que siguen la arquitectura REST.



Capa de Procesamiento

Descripción del Proyecto

El proyecto consiste en el desarrollo de un medidor de glucosa IoT que utiliza un microcontrolador ESP32, un sensor óptico CNY 70 y un panel táctil para el encendido y control del dispositivo. Este medidor será capaz de capturar y procesar los niveles de glucosa en sangre, enviando la información a una plataforma en la nube para su almacenamiento y análisis. El objetivo es proporcionar un dispositivo compacto, fácil de usar y conectado, que permita a los usuarios monitorear sus niveles de glucosa en tiempo real y acceder a sus datos desde cualquier lugar.

Resumen

En esta etapa del proyecto, nos centramos en el procesamiento de datos, implementando la capa de backend que gestionará la interacción entre el dispositivo IoT y la plataforma en la nube. Estamos desarrollando una API RESTful para la comunicación con la base de datos y la gestión de datos críticos del sistema IoT, como dispositivos, usuarios, proyectos y sus configuraciones.

Componentes del Proyecto:

Componentes Principales

1. Microcontrolador ESP32

- Descripción: El ESP32 es un microcontrolador de alto rendimiento con capacidades Wi-Fi y Bluetooth integradas. Es el cerebro del sistema, responsable de procesar los datos del sensor, controlar la interfaz de usuario, y manejar la comunicación con la nube.
- Función:
 - Recepción de datos del sensor CNY 70.
 - Procesamiento de los datos de glucosa.
 - Envío de los datos procesados a un servidor en la nube.
 - Control del encendido del dispositivo a través del panel táctil.

2. Sensor Óptico CNY 70

- Descripción: El CNY 70 es un sensor reflectivo infrarrojo que se utiliza para detectar la concentración de glucosa en una muestra de sangre. Funciona emitiendo luz infrarroja y midiendo la cantidad de luz reflejada, la cual varía según la concentración de glucosa.
- Función:
 - Detección de los niveles de glucosa en sangre.
 - Generación de una señal analógica que es enviada al ESP32 para su procesamiento.

3. Pantalla LCD (Opcional)

- Descripción: Una pequeña pantalla LCD podría ser utilizada para mostrar los resultados de la medición de glucosa y el estado del dispositivo.
- Función:
 - Visualización en tiempo real de los niveles de glucosa.
 - Visualización de mensajes de estado o alertas.

4. Conectividad Wi-Fi

- Descripción: La capacidad Wi-Fi del ESP32 permite que el dispositivo se conecte a internet y envíe los datos de glucosa a una plataforma en la nube.
- Función:
 - Sincronización automática de los datos con una base de datos en la nube.
 - Posibilidad de enviar alertas al teléfono móvil del usuario en caso de niveles de glucosa fuera de rango.

5. Fuente de Alimentación

- Descripción: El dispositivo será alimentado por una batería recargable, con posibilidad de recarga mediante un puerto USB.
- Función:
 - Proveer energía al ESP32, sensor CNY 70, panel táctil y otros componentes del dispositivo.

- Posibilidad de integrar un indicador de nivel de batería en la interfaz del dispositivo.

Resultados Esperados:

1. Mediciones Precisas y Confiables:

El prototipo proporcionará mediciones precisas de los niveles de glucosa en sangre, con un margen de error mínimo, gracias al preprocesamiento de datos que incluye filtrado, calibración, y detección de anomalías.

2. Monitoreo en Tiempo Real:

El dispositivo permitirá a los usuarios obtener lecturas en tiempo real de sus niveles de glucosa, con la posibilidad de visualizar los resultados instantáneamente en una pantalla o a través de una aplicación móvil conectada a la nube.

3. Conectividad y Acceso Remoto:

Los datos de glucosa procesados serán enviados a una plataforma en la nube, permitiendo a los usuarios y a sus médicos acceder a

un historial completo de mediciones desde cualquier lugar, facilitando un mejor control y seguimiento de la salud.

4. Interfaz de Usuario Intuitiva:

El panel táctil permitirá a los usuarios encender el dispositivo y posiblemente seleccionar diferentes modos de operación de manera sencilla, mejorando la experiencia de uso.

5. Alerta y Notificación:

En caso de detectar niveles de glucosa fuera de los rangos normales, el dispositivo enviará alertas instantáneas al usuario o a su médico, permitiendo una respuesta rápida ante situaciones potencialmente peligrosas.

6. Optimización del Uso de Energía:

El dispositivo estará diseñado para ser eficiente en términos de consumo de energía, con opciones de apagado automático y un sistema de encendido mediante el panel táctil, prolongando la vida útil de la batería.

7. Base para Desarrollo Futuro:

El prototipo servirá como una base sólida para futuros desarrollos, permitiendo la integración de funcionalidades adicionales, como análisis predictivo, conexión con otros dispositivos médicos, o personalización según las necesidades del usuario.

Tecnologías Utilizadas:

- Microcontroladores: ESP32-Wroom
- Sensores: CNY-70
- Frameworks: C++ para el desarrollo de hardware y para la gestión de tareas en tiempo real.
- Herramientas de Desarrollo: Visual Studio Code con PlatformIO
- Fog Computing: Microcomputadoras, microservidores, etc.

Evaluación de Uso del Dispositivo para Medir Diferentes Parámetros Biológicos

Propuesta

El objetivo del presente informe es evaluar la posibilidad de utilizar el sensor CNY70 no solo para la medición de glucosa en sangre, sino también para medir otros parámetros biológicos, como el ritmo cardíaco y otros relacionados con la salud. Se analizará la viabilidad técnica del CNY70 en estas aplicaciones y, en caso de limitaciones, se evaluarán sensores alternativos que permitan la medición precisa de diversos parámetros, extendiendo así la capacidad del proyecto.

Análisis de la Viabilidad del Sensor CNY70

1. Funcionamiento del CNY70

El CNY70 es un sensor infrarrojo sencillo, diseñado para detectar la reflexión de luz en superficies cercanas. Está compuesto por un emisor de luz infrarroja y un fototransistor que mide la cantidad de luz reflejada.

2. Limitaciones del CNY70 para Parámetros Biológicos

El sensor CNY70 no está diseñado para medir los parámetros biológicos comunes que se encuentran en dispositivos como relojes inteligentes o

muñequeras de salud. Entre los parámetros que no puede medir, encontramos:

- **Frecuencia cardíaca (FC):** El CNY70 no tiene la precisión ni la capacidad de detectar cambios sutiles en el flujo sanguíneo como lo hacen los sensores de fotopletismografía (PPG).
- **Saturación de oxígeno (SpO₂):** Para medir SpO₂ se necesitan dos longitudes de onda (roja e infrarroja), algo que el CNY70 no puede ofrecer.
- **Ritmo de sueño, HRV, calorías quemadas, contador de pasos:** Estos parámetros requieren sensores de movimiento (acelerómetros) y sensores ópticos más avanzados que midan con precisión.
- **Glucosa en sangre:** Aunque el CNY70 opera en el infrarrojo, no está optimizado para medir la absorción de glucosa, ya que no emite ni detecta las longitudes de onda necesarias para la espectroscopía de glucosa.

3. Especificaciones de Sensores Requeridos para Medición de Glucosa

Para medir la glucosa en sangre de forma no invasiva utilizando infrarrojos, se necesitan las siguientes características:

- **Longitudes de onda:** El rango de 900 nm a 2500 nm es esencial para detectar los picos de absorción de glucosa.
- **Detectores sensibles:** Los fotodetectores InGaAs son ideales, ya que tienen alta sensibilidad en el rango NIR.
- **Emisores multiespectrales:** Para medir la absorción de glucosa de manera precisa, es necesario utilizar emisores que cubran múltiples longitudes de onda.

Sensores Alternativos Recomendados

Para cumplir con las demandas de medir diversos parámetros biológicos de manera precisa, se recomiendan los siguientes tipos de sensores:

1. Emisores Infrarrojos (LEDs y Láseres)

- **Osram SFH 4715A (940 nm):** Adecuado para aplicaciones básicas en infrarrojo cercano.
- **Thorlabs LED1300E (1300 nm):** Para mediciones más especializadas, en el rango de absorción de glucosa.

- **Roithner 1650D-01 (1650 nm):** Específico para uno de los picos de absorción de glucosa.

2. Fotodetectores InGaAs

- **Thorlabs FGA21:** Detecta luz en el rango de 800 nm a 1700 nm.
- **Hamamatsu G8370-03:** Fotodiodo con alta sensibilidad en el rango NIR (900 nm - 1700 nm).

3. Espectrómetros NIR Portátiles

- **Ocean Insight NIRQuest 512:** Cubre longitudes de onda desde 900 nm hasta 2500 nm.
- **Si-Ware NeoSpectra:** Espectrómetro miniaturizado para análisis de espectros en tiempo real.

4. Filtros Ópticos

- **Thorlabs FESH1000:** Filtro de paso de banda para aislar las longitudes de onda de interés y minimizar interferencias.

Mejor Opción para Medir el Ritmo Cardíaco

Para sumar la capacidad de medir el ritmo cardíaco de manera económica y con fácil acceso, la mejor opción es utilizar un sensor de fotopletismografía (PPG) basado en luz infrarroja o verde. Estos sensores, como el MAX30102 o el MAX30100, son módulos compactos y asequibles que combinan un LED emisor y un fotodetector para detectar cambios en el volumen sanguíneo en los vasos capilares del dedo o la muñeca, lo que permite determinar la frecuencia cardíaca. Estos módulos están ampliamente disponibles en el mercado, son fáciles de integrar con microcontroladores como el ESP32, y ofrecen una buena precisión para aplicaciones personales de monitoreo de salud.

Conclusión

El sensor CNY70, aunque útil para detección de proximidad o reflejo en superficies, no es adecuado para medir parámetros biológicos como la glucosa en sangre, la frecuencia cardíaca o la saturación de oxígeno.

Para realizar estas mediciones, es necesario recurrir a sensores más avanzados como emisores multiespectrales, fotodetectores InGaAs y espectrómetros NIR.

A fines prácticos, y educativos, continuaremos el prototipado utilizando sensor CNY70 y evaluaremos la posibilidad de sumar al prototipo el sensor MAX30102/0 a fin de extender las capacidades del mismo.

La principal limitación para disponer de los sensores mencionados en puntos anteriores radica en la disponibilidad y/o precio. A futuro la incorporación de estos sensores alternativos para extender la capacidad del proyecto, permitiendo la medición precisa de parámetros biológicos relevantes, especialmente la glucosa en sangre mediante espectroscopía NIR, es el destino lógico.

El proyecto consiste en el desarrollo de un medidor de glucosa IoT que utiliza un microcontrolador ESP32, un sensor óptico CNY 70 y un panel táctil para el encendido y control del dispositivo. Este medidor será capaz de capturar y procesar los niveles de glucosa en sangre, y enviar la información a una plataforma en la nube para su almacenamiento y análisis. El objetivo es proporcionar un dispositivo compacto, fácil de usar y conectado, que permita a los usuarios monitorear sus niveles de glucosa en tiempo real y acceder a sus datos desde cualquier lugar.

Estructura del Proyecto

1. Capa EDGE

Descripción:

La capa EDGE es la primera línea de procesamiento en el dispositivo IoT. Esta capa es responsable de la adquisición de datos y el preprocesamiento inicial antes de que la información sea transmitida a las capas superiores.

Especificaciones:

- Captura de Datos: Los sensores de glucosa integrados en el dispositivo capturan datos en tiempo real. Estos datos pueden incluir niveles de glucosa, temperatura del sensor, y otros parámetros fisiológicos relevantes.
- Filtrado de Señal: Se aplican técnicas de filtrado para eliminar ruido y artefactos en los datos crudos. Normalización de Datos: Los datos de glucosa se normalizan para asegurar consistencia en las unidades de medida.
- Compresión de Datos: Implementación de algoritmos de compresión ligera para reducir la cantidad de datos a transmitir sin

pérdida significativa de información. Seguridad: Encriptación básica de datos antes de la transmisión para garantizar la privacidad del paciente.

2. Capa FOG

Descripción:

La capa FOG actúa como una extensión de la capa EDGE y permite un procesamiento intermedio antes de que los datos sean enviados a la nube. Esta capa reduce la latencia y permite una respuesta más rápida en tiempo real.

Especificaciones:

- Análisis Local: Los datos preprocesados en la capa EDGE son sometidos a análisis más avanzados, como la detección de patrones anómalos que podrían indicar condiciones peligrosas para el paciente (e.g., hipoglucemia o hiperglucemia).
- Almacenamiento Temporal: Se utiliza almacenamiento local para guardar datos críticos que pueden ser retransmitidos en caso de pérdida de conexión.

- Toma de Decisiones: Implementación de algoritmos de toma de decisiones locales, permitiendo acciones inmediatas como la activación de alertas al paciente si se detectan niveles peligrosos de glucosa.
- Balanceo de Carga: Gestión de la carga de datos para optimizar el uso del ancho de banda disponible y mejorar la eficiencia del sistema.
- Seguridad Avanzada: Se aplica una segunda capa de encriptación y autenticación de datos para proteger la integridad y confidencialidad de la información durante la transmisión a la nube.

3. Capa CLOUD

Descripción:

La capa CLOUD es la última etapa en el procesamiento de datos, encargada del almacenamiento masivo, análisis a gran escala y acceso remoto a la información.

Especificaciones:

- Almacenamiento en la Nube: Los datos recolectados y procesados en las capas EDGE y FOG son almacenados en la nube para análisis posteriores y acceso por parte de profesionales de la salud.
- Análisis Predictivo: Utilización de algoritmos de aprendizaje automático para predecir tendencias en los niveles de glucosa y proporcionar recomendaciones personalizadas.
- Dashboard de Monitoreo: Desarrollo de interfaces web y aplicaciones móviles para que los pacientes y médicos puedan visualizar los datos en tiempo real, acceder a históricos y recibir alertas.
- Integración con Otros Sistemas: La capa CLOUD permite la integración con sistemas de gestión hospitalaria, registros médicos electrónicos (EMR), y otros dispositivos médicos IoT.
- Seguridad y Cumplimiento: Cumplimiento con normativas de privacidad de datos como HIPAA y GDPR, asegurando la protección de la información sensible del paciente.

Objetivos de la Investigación

- Optimización del Preprocesamiento:

Investigar y desarrollar técnicas eficientes de preprocesamiento en las capas EDGE y FOG para mejorar la calidad de los datos transmitidos.

- Reducción de Latencia:

Minimizar la latencia en la transmisión y procesamiento de datos a través de las diferentes capas.

- Seguridad de Datos:

Implementar y evaluar métodos avanzados de encriptación y autenticación a lo largo de las tres capas para garantizar la seguridad y privacidad de los datos.

- Escalabilidad:

Desarrollar una arquitectura escalable que permita la integración de múltiples dispositivos y el manejo de grandes volúmenes de datos.

Conclusión

La implementación de un sistema IoT para el monitoreo de glucosa basado en una arquitectura de procesamiento en capas (EDGE, FOG y CLOUD) es fundamental para asegurar un análisis eficiente y en tiempo real de los datos, garantizando al mismo tiempo la seguridad y privacidad de la información. Este enfoque no solo mejora la calidad del monitoreo de salud, sino que también permite una intervención médica más rápida y precisa.

Costo

Componentes	cantidad	Precio unidad	total
Microcontroladores ESP 32	1	\$ 10,93	\$ 10,93
Sensor Óptico CNY 70	1	\$ 1,55	\$ 1,55
Pantalla Oled 0.96p Ssd1306 I2c	1	\$ 7,88	\$ 7,88
Batería pequeña de polímero de litio 1000mAh 3,7V recargable	1	\$ 9,75	\$ 9,75
Rollo Cable Rojo 5 Metros 0.5mm Unipolar	1	\$ 4,22	\$ 4,22
Rollo Cable celeste 5 Metros 0.5mm Unipolar	1	\$ 2,67	\$ 2,67
Diodo LED, 5mm Rojo 625nm, 5.000mcd 2V 100mW 20mA	1	\$ 0,11	\$ 0,11
Diodo LED, 5mm Verde, 525nm, 20.000mcd 3V 100mW 20mA	1	\$ 0,08	\$ 0,08
		Precio Total	\$ 37,19

```
curl -H "X-API-KEY: jade" https://api.gonaiot.com/jade/usuarios/
```

Licencia

Este proyecto está licenciado bajo los términos de la Licencia MIT. Consulta el archivo LICENSE para más detalles.

Proyecto IoT: API RESTful

Este proyecto proporciona una API RESTful para gestionar datos de un proyecto IoT utilizando Flask y MySQL. La API soporta operaciones CRUD para varias entidades como usuarios, dispositivos, proyectos, configuraciones, datos de dispositivos y seguridad.

Estructura del Proyecto

```
c_capa_de_analisis/
├── app.py
├── db_config.py
└── models.py
└── routes/
    ├── configuraciones.py
    ├── datos_dispositivos.py
    ├── dispositivos.py
    ├── proyectos.py
    ├── seguridad.py
    └── usuarios.py
└── requirements.txt
└── Dockerfile
```

Instalación

1. Clonar el repositorio.
2. Navegar al directorio del proyecto.
3. Ejecutar `pip install -r requirements.txt` para instalar las dependencias.
4. Configurar la base de datos en `db_config.py`.
5. Ejecutar `python app.py` para iniciar el servidor.

Uso

Endpoints

La API proporciona los siguientes endpoints:

Usuarios

1. Obtener Usuarios

- Método: GET
- URL: /usuarios/
- Descripción: Devuelve una lista de todos los usuarios registrados en la base de datos.
- Respuesta:

```
[  
  {  
    "id": 1,  
    "nombre": "Juan Perez",  
    "email": "juan.perez@example.com"  
  },  
  ...  
]
```

2.Crear Usuario

- Método: POST
- URL: /usuarios/
- Descripción: Crea un nuevo usuario en la base de datos.
- Cuerpo de la Solicitud:

```
{  
  "nombre": "Juan Perez",  
  "email": "juan.perez@example.com",  
  "password": "password123"  
}
```

- Respuesta: Código de estado `201 Created` si se crea con éxito.

Dispositivos

1. Obtener dispositivos

- Obtener Dispositivos
- Método: GET
- URL: /dispositivos/
- Descripción: Devuelve una lista de todos los dispositivos registrados.
- Respuesta:

```
[
  {
    "id": 1,
    "nombre": "Sensor de temperatura",
    "tipo": "Sensor",
    "ubicacion": "Sala de servidores"
  },
  ...
]
```

2.Crear Dispositivo

- Método: POST
- URL: /dispositivos/
- Descripción: Registra un nuevo dispositivo en la base de datos.
- Cuerpo de la Solicitud:

```
{
  "nombre": "Sensor de temperatura",
  "tipo": "Sensor",
  "ubicacion": "Sala de servidores"
}
```

Respuesta: Código de estado `201 Created` si se crea con éxito.

Datos de Dispositivos

1. Obtener Datos de Dispositivos

- Método: `GET`
- URL: `/datos_dispositivos/`
- Descripción: Devuelve una lista de los datos registrados para los dispositivos.
- Respuesta:

```
[  
  {  
    "id": 1,  
    "dispositivo_id": 1,  
    "valor": 23.5,  
    "unidad": "C",  
    "timestamp": "2024-06-24T00:00:00Z"  
  },  
  ...  
]
```

2. Enviar Datos de Dispositivo

- Método: POST
- URL: /datos_dispositivos/
- Descripción: Envía nuevos datos para un dispositivo.
- Cuerpo de la Solicitud:

```
{  
    "dispositivo_id": 1,  
    "valor": 23.5,  
    "unidad": "C"  
}
```

- Respuesta: Código de estado 201 Created si se envía con éxito.

Configuraciones

1. Obtener Configuraciones

- Método: GET
- URL: /configuraciones/
- Descripción: Devuelve una lista de las configuraciones del sistema.
- Respuesta:

```
[  
    {  
        "id": 1,  
        "nombre": "intervalo_muestreo",  
        "valor": "10s"  
    },  
    ...  
]
```

2. Actualizar Configuración

- Método: **PUT**
- URL: **/configuraciones/{id}**
- Descripción: Actualiza una configuración específica.
- Cuerpo de la Solicitud:

```
{  
    "nombre": "intervalo_muestreo",  
    "valor": "10s"  
}
```

- Respuesta: Código de estado **200 OK** si se actualiza con éxito.

Proyectos

1. Obtener Proyectos

- Método: GET
- URL: /proyectos/
- Descripción: Devuelve una lista de todos los proyectos registrados.
- Respuesta:

```
[  
 {  
   "id": 1,  
   "nombre": "Proyecto IoT",  
   "descripcion": "Proyecto para monitoreo de temperatura"  
 },  
 ...  
 ]
```

2. Crear Proyecto

- Método: POST
- URL: /proyectos/
- Descripción: Crea un nuevo proyecto en la base de datos.
- Cuerpo de la Solicitud:

```
{  
   "nombre": "Proyecto IoT",  
   "descripcion": "Proyecto para monitoreo de temperatura"  
 }
```

Respuesta: Código de estado 201 Created si se crea con éxito.

Seguridad

1. Obtener Información de Seguridad

- Método: GET
- URL: /seguridad/
- Descripción: Devuelve la información de seguridad del sistema.
- Respuesta:

```
{  
  "seguridad_activa": true,  
  "ultimo_chequeo": "2024-06-24T00:00:00Z"  
}
```

Componentes Clave de la Capa de Percepción

- **Controlador ESP32:** Configurado para la captura y transmisión de datos de glucosa.
- **Sensores:** Selección y calibración de sensores específicos para medir glucosa en sangre.
- **Protocolo de Comunicación:** Implementación de comunicación eficiente entre los sensores y el sistema, garantizando precisión y consistencia en los datos enviados.

Licencia

Este proyecto está licenciado bajo los términos de la **Licencia MIT**. Consulta el archivo [LICENSE](#) para más detalles.

Capa de Presentación:

Refiere a las interfaces gráficas que permiten al usuario interactuar con el sistema de manera intuitiva. Esta capa facilita la visualización de datos y el control remoto de los dispositivos conectados al sistema IoT en un formato comprensible. Su propósito es servir como un puente entre los usuarios y los dispositivos, interpretando los datos recopilados por los sensores IoT y ofreciendo funcionalidades para la toma de decisiones y ajustes operativos según los datos mostrados.

Rol de la misma

La capa de presentación desempeña un papel esencial al ser el punto de interacción principal entre el usuario y el sistema IoT. Su enfoque se centra en proporcionar una interfaz de usuario sencilla pero eficaz que permita visualizar los datos en tiempo real y gestionar dispositivos de manera remota.

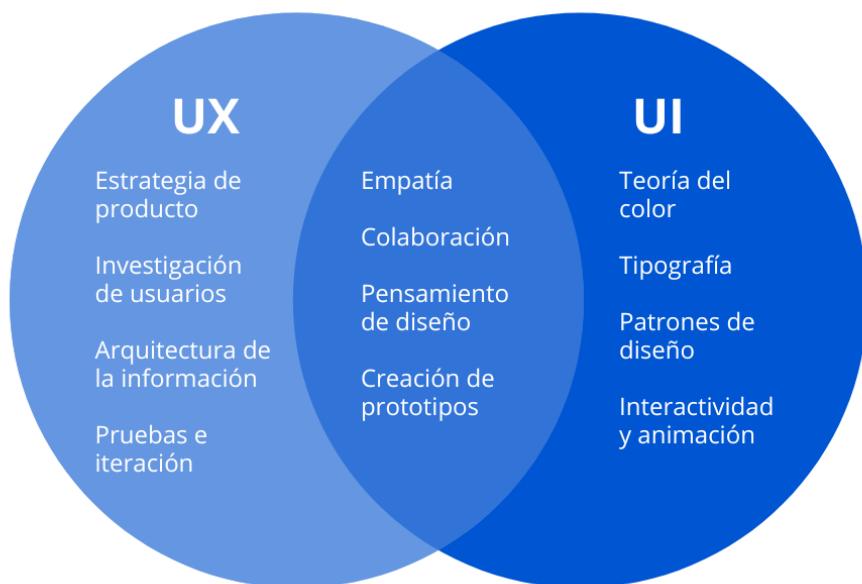
Rol de la capa en nuestro proyecto y su enfoque

En nuestro proyecto, el foco está del lado del profesional de la salud, que provea los dispositivos y pueda mediante esta llevar un seguimiento de los mismos. Haciendo uso de una presentación en modelo Landing Page, y accediendo a las diferentes opciones de interacción mediante un tab menu de fácil acceso en la parte superior de la página, debajo del header. La intención fundamental de nuestro modelo de interacción, es que el/los profesionales de la salud puedan hacer un uso de un

sistema simple y claro, que permita llevar bien la gestión de pacientes, sus contactos y datos personales, así como las lecturas individuales y compartidas de cada dispositivo en acción.

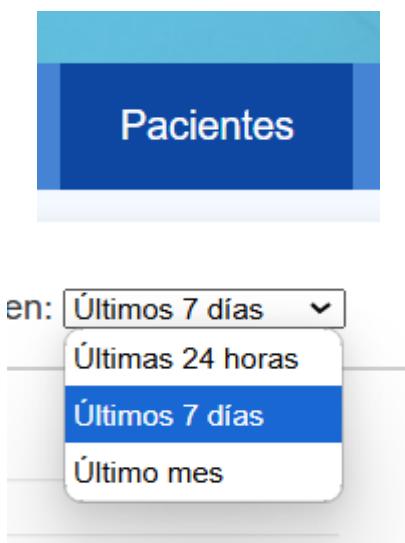
UX/UI

La UX (Experiencia de Usuario) se refiere a cómo se siente y percibe el usuario al interactuar con el sistema, centrándose en la facilidad de uso, accesibilidad y satisfacción. La UI (Interfaz de Usuario) es la parte visual e interactiva que el usuario ve y manipula, incluyendo botones, menús y gráficos, diseñada para que la interacción sea intuitiva y atractiva. En conjunto, UX/UI buscan crear una experiencia eficaz, agradable y sin fricciones para el usuario



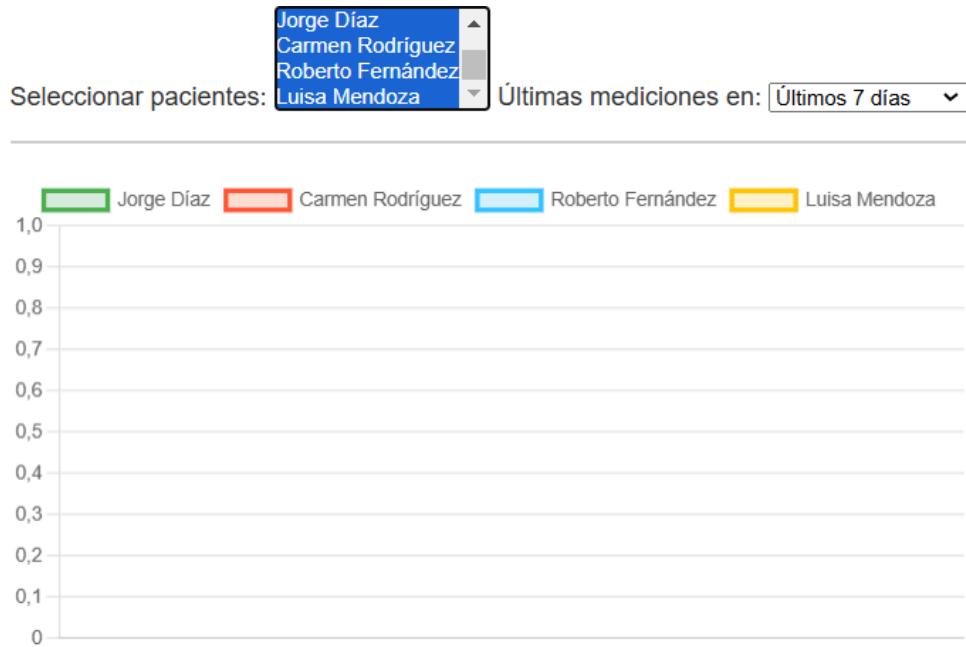
Def UX/UI - Foco de este en nuestro proyecto

La experiencia de usuario (UX) y la interfaz de usuario (UI) en nuestro proyecto están diseñadas para maximizar la usabilidad y la accesibilidad. La interfaz es intuitiva, de fácil navegación y responde a las necesidades específicas de los usuarios finales. La estructura del diseño UX/UI en nuestro proyecto se centra en hacer la interacción con el sistema IoT una tarea sin esfuerzo, donde los datos son claros, relevantes, y las opciones de control están rápidamente disponibles. Pero cabe destacar que estas disciplinas son amplias y requieren de conocimientos muy abarcativos, en los que lejos estamos de ser conocedores y expertos, por lo que el resultado es más una intención “de”, que un logrado.



Interfaces típicas

Las interfaces típicas en un sistema IoT incluyen paneles de control, gráficos de datos, controles de dispositivos y notificaciones. En nuestro proyecto, la interfaz principal es un panel de control web que permite al usuario monitorear todos los dispositivos conectados.



Esta interfaz principal se complementa con gráficos en tiempo real y opciones de ajuste rápido para ofrecer una experiencia de usuario integrada y conectada. Así no, es carente en su versión más reciente de notificaciones mediante el sistema PWA, del que hablamos más adelante, así como de interacción de control sobre los dispositivos. Pero no nos es ajeno la idea o posibilidad de implementarlo en versiones futuras

Web First

El enfoque Web First implica diseñar la capa de presentación priorizando la plataforma web. Esto significa que toda la funcionalidad está optimizada para ser accedida desde navegadores web antes de considerar adaptaciones móviles. Las ventajas incluyen mayor accesibilidad, facilidad de despliegue y menor complejidad en el mantenimiento. Sin embargo, entre las desventajas, se encuentran posibles limitaciones en la experiencia móvil y en la optimización offline.



Nuestro proyecto está construido bajo un enfoque Web First, donde todas las funcionalidades principales de la capa de presentación se desarrollaron inicialmente para la plataforma web. Esto permite a los usuarios acceder a todas las características sin necesidad de aplicaciones adicionales. La implementación web facilita la visualización de datos y el control de dispositivos en tiempo real desde cualquier navegador. Sumando la breves capacidades del modelo PWA, para poder brinda mayor facilidad, a muy bajo costo/esfuerzo, al momento de llevar la plataforma al mundo de móvil.



HTML

El HTML en nuestro proyecto organiza el contenido de la interfaz de usuario en una estructura clara y semántica. Utilizamos etiquetas HTML5 para dividir el contenido en secciones como encabezados, gráficos y controles de dispositivo, asegurando una fácil navegación y una estructura lógica que optimiza la accesibilidad

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plataforma de Monitoreo de Pacientes</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="{{ url_for('static', filename='js/scripts.js') }}" defer></script>
</head>
<body>
```

CSS

El CSS se utiliza para estilizar nuestra aplicación web, brindando coherencia visual y adaptabilidad en múltiples dispositivos. Nos permite definir la estética de cada componente de la interfaz, garantizando una experiencia de usuario agradable y funcional. Los estilos están organizados de manera modular para simplificar el mantenimiento y facilitar futuras modificaciones.

JS

JavaScript permite la funcionalidad dinámica en nuestro sistema IoT, como la actualización en tiempo real de los datos, el manejo de eventos y la comunicación con la API REST. Su implementación es crucial para la interacción instantánea entre el usuario y los dispositivos IoT, aportando una experiencia interactiva y reactiva.

- Frameworks (react/vue/angular), y por que no se utilizaron en el proyecto

Optamos por no utilizar frameworks como React, Vue o Angular debido a la naturaleza sencilla y específica de nuestra aplicación IoT. Dado el enfoque en una capa de presentación ligera, optamos por JavaScript básico, evitando la complejidad adicional que estos frameworks pueden aportar para una aplicación que no requiere de una estructura de componentes avanzada.

- Por qué no se utilizó Mobile First

El enfoque Mobile First no se aplicó ya que nuestro proyecto prioriza el acceso desde dispositivos de escritorio debido a la complejidad de los gráficos y controles

en la interfaz. El diseño responsive permite que la aplicación se adapte a diferentes tamaños de pantalla sin comprometer la funcionalidad.

- Qué es PWA (Progressive Web App) y por qué se utilizó

Una Progressive Web App (PWA) es una aplicación web que ofrece una experiencia similar a una app nativa, con capacidades como carga rápida, notificaciones push y acceso offline. Se utilizó PWA en nuestro proyecto para mejorar la accesibilidad y permitir el uso de la aplicación incluso sin conexión constante.

- Qué es ngrok y por qué lo utilizamos, relacionado a PWA

Ngrok es una herramienta que permite exponer el servidor local a una URL pública. Lo utilizamos en la fase de desarrollo de la PWA para probar la accesibilidad remota y simular el acceso desde ubicaciones fuera de la red local, asegurando la funcionalidad en entornos de producción.

- Dashboards. Grafana y su uso.

- Como implementamos estos en el proyecto

- Conexion de la base de datos con esta

- Instermamos informacion artificalmente a la base de datos, a fin de mostrar como interactua esta.

- Visualizacion en tiempo real y la aplicacion en nuestro proyecto

Investigacion

Bibliografia

Formato:

Link | Descripcion breve del contenido

<https://www.mdpi.com/1424-8220/17/1/182> | Descripcion de sensores de glucosa poco invasivo y sensores no invasivos

<https://www.mdpi.com/1424-8220/10/5/4558> | Principios de funcionamiento de los biosensores de glucosa

Referencias Bibliográficas

- Manuales y cuadernillos digitales de Ciencias de la Computación
<https://program.ar/material-didactico/>
- Unidad 1. Fundamentos de Informática SOPORTE LÓGICO EN UN ORDENADOR PERSONAL: EL SOFTWARE

<https://www3.gobiernodecanarias.org/medusa/ecoblog/mgoncal/>

Conclusión

El desarrollo del sistema IoT para monitoreo de glucosa ha demostrado ser una herramienta innovadora y esencial para el control de la salud en tiempo real, proporcionando un acceso constante a datos críticos para pacientes y profesionales médicos. Este sistema permite a los usuarios monitorear sus niveles de glucosa de manera confiable, con la opción de recibir alertas cuando se detectan anomalías, mejorando así la prevención de emergencias de salud.

La arquitectura basada en capas (EDGE, FOG y CLOUD) garantiza una alta eficiencia en el procesamiento y transmisión de datos, optimizando recursos y facilitando una respuesta ágil y precisa. Además, la implementación de medidas de seguridad refuerza la protección de los datos sensibles, brindando confianza a los usuarios y estableciendo un estándar para futuros proyectos de IoT en salud.

En perspectiva, este sistema sienta las bases para futuras mejoras, como la integración de algoritmos de análisis predictivo y la incorporación de otros parámetros de salud. Este proyecto no solo representa un avance tecnológico, sino también un paso hacia una atención médica más accesible, personalizada y proactiva.

Dirección General de
**EDUCACIÓN TÉCNICA Y
FORMACIÓN PROFESIONAL**

Ministerio de
EDUCACIÓN

