



UNIVERSIDAD PERUANA
CAYETANO HEREDIA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

UNIVERSIDAD PERUANA CAYETANO HEREDIA
FACULTAD DE CIENCIAS E INGENIERÍA

***DESARROLLO DE UNA API PARA LA
INTEROPERABILIDAD DE PROTOTIPOS DE
DISPOSITIVOS MÉDICOS ESTUDIANTILES CON UNA
HISTORIA CLÍNICA ELECTRÓNICA LIBRE***

Tesis para obtener el título profesional de Ingeniero Biomédico

que presenta:

Willy Joel Valdivia Leon

Asesor(es):

Moises Stevend Meza Rodriguez

Lima, 2026

Jurado calificador

Presidente: Mag. Felix Alonso Agapito Cordova

Vocal: Dra. Mabel Karel Raza Garcia

Secretario: Mag. Miguel Rogger Hoyos Alvitez



UNIVERSIDAD PERUANA
CAYETANO HEREDIA

DECLARACIÓN DE ORIGINALIDAD

Los egresados:

N°	APELLIDOS Y NOMBRES
1.	VALDIVIA LEON WILLY JOEL

(Agregar filas adicionales si hay más autores)

Pertenecientes al programa de la **CARRERA PROFESIONAL DE INGENIERÍA BIOMÉDICA**, autores del trabajo titulado: **DESARROLLO DE UNA API PARA LA INTEROPERABILIDAD DE PROTOTIPOS DE DISPOSITIVOS MÉDICOS ESTUDIANTILES CON UNA HISTORIA CLÍNICA ELECTRÓNICA LIBRE**, el cual ha sido elaborado, sustentado y aprobado, según corresponda, para optar por el **TÍTULO PROFESIONAL DE INGENIERO BIOMÉDICO** bajo la modalidad de **TESIS**.

En calidad de docentes asesores de la Universidad Peruana Cayetano Heredia:

N°	APELLIDOS Y NOMBRES DEL DOCENTE	FACULTAD	NIVEL DE ASESORÍA
1.	MEZA RODRIGUEZ MOISES STEVEND	FACI	ASESOR

Declaramos que el contenido del presente documento es original y que las citas y referencias a otros autores cumplen con las normas académicas establecidas. En ese sentido, hacemos constar que:

- El documento presenta un porcentaje de similitud de **6%**, según el reporte emitido por el software Turnitin® (identificador de entrega: **3658G21043**; fecha de entrega: **24/06/2026**).
- Tras una revisión detallada del reporte y del contenido del trabajo en cuestión, no se han identificado indicios de plagio.
- Se certifica que el documento respeta los principios de integridad académica y cumple con los requisitos institucionales de originalidad.

Lugar y fecha: Lima, 24 de septiembre de 2026

Firma del asesor
N° DNI: 47470482
ORCID: 0000-0002-5806-9014

Asesor:

Mag. Moises Stevend Meza Rodriguez

Dedicatoria:

El presente trabajo de tesis se lo dedico a toda mi familia; en especial a mi madre, Yolanda León, y a mi hermano, Cristopher Casani, ya que su apoyo incondicional fue el principal motor que me impulsó durante mi carrera, sobre todo en los momentos más difíciles. También se lo dedico a mis amigos más cercanos del colegio y de la universidad, especialmente a Milagros Alvites, Frank Britto, Claudia Montufar, Erika Uchuya, Rodrigo Alarcón, José Cebrián, Ashley Delgado, Alexandra Calvo, Linda Portillo y Walter Castillo, quienes siempre me brindaron sus consejos y valiosa compañía a lo largo de este trayecto. Y a todas aquellas personas que me brindaron su apoyo y aliento en los momentos que más lo necesité.

Agradecimiento:

Expreso mi más profundo agradecimiento a mi asesor, Mag. Moises Meza, por su invaluable guía académica, su paciencia y por brindarme las herramientas y la visión necesarias para elevar este proyecto a un nivel de exigencia profesional. Mi gratitud también se dirige a mi casa de estudios, conformada por la Pontificia Universidad Católica del Perú y la Universidad Peruana Cayetano Heredia, y a todos los docentes de la carrera de Ingeniería Biomédica, por haberme formado con rigor científico y vocación humana durante estos años. Finalmente, gracias a todas las personas que, de una u otra forma, me brindaron su apoyo incondicional a lo largo de este trayecto universitario.

Fuentes de financiamiento:

El presente proyecto de investigación ha sido autofinanciado por el autor

Índice

Resumen.....	1
Abstract.....	2
I. Introducción.....	3
1.1 Problemática	3
1.2 Propósito del estudio.....	6
1.3 Estado del Arte.....	7
1.3.1 Categoría A: Sistemas basados en blockchain para intercambio de información clínica.....	8
1.3.2 Categoría B: Sistemas de integración y estandarización de datos clínicos....	10
1.3.3 Síntesis del estado del arte	13
II. Marco teórico	14
2.1 Interoperabilidad en Salud	14
2.2 Estándar HL7 FHIR	15
2.3 Protocolo MQTT.....	17
2.4 Ecosistema GNU Health.....	18
2.4.1 Arquitectura Técnica y Conceptos Fundamentales	20
2.4.2 Estrategia de Despliegue en Hardware Embebido.....	21
2.5 Frameworks de Desarrollo de la API.....	22
2.5.1 Framework del Backend: Flask	22
2.5.2 Framework del Frontend: Streamlit.....	22
2.6 Arquitectura de Servicios Web (API REST)	23
2.7 Microcontrolador ESP32	25
2.8 Metodología Input Space Partitioning (ISP).....	26
III. Pregunta de investigación y Objetivos.....	28
3.1 Pregunta de investigación.	28
3.2 Objetivo general.....	28

3.3 Objetivos específicos	28
IV. Materiales y métodos.....	29
4.1 Enfoque Metodológico	29
4.2 Materiales y Equipos	30
4.2.1 Recursos de hardware	30
4.2.2 Recursos de Software y Herramientas de Desarrollo	31
4.2.3 Frameworks y Librerías de Programación.....	31
4.2.3.1 Capa de Firmware (Arduino IDE / C++)	32
4.2.3.2 Capa de Backend y Servicios API (Python 3.11.2)	32
4.2.3.3 Capa de Visualización y Frontend (Streamlit).....	32
4.2.4 Configuración del Ecosistema GNU Health (HMIS v4.4)	32
4.3. Implementación del Firmware del Nodo Simulador (Etapa 1).....	33
4.3.1 Generación de datos fisiológicos sintéticos	33
4.3.2 Estandarización e Interoperabilidad Semántica (HL7 FHIR).....	34
4.3.3 Comunicación y Transmisión mediante Protocolo MQTT.....	35
4.4. Implementación del Middleware de Integración (Etapa 2)	35
4.4.1 Desarrollo del Módulo de Recepción y Almacenamiento de Datos	37
4.4.1.1 Suscripción Dinámica y Recepción de Mensajería.....	37
4.4.1.2 Procesamiento de Recursos FHIR y Almacenamiento	37
4.4.2 Instalación y puesta en marcha del HMIS de GNU Health	38
4.4.3 Identificación de estructura de la base de datos de GNU Health.....	40
4.4.4 Desarrollo de código para el almacenamiento de datos en GNU Health.....	43
4.4.4.1 Pruebas de Concepto: Registro Manual y Mapeo Relacional.....	43
4.4.4.1.1 Gestión de Restricciones y Estampado Temporal	43
4.4.4.1.2 Lógica de Mapeo de Identidades (DNI a ID Interno)	43
4.4.4.1.3 Ejecución del almacenamiento de datos (SQL)	44

4.4.4.2 Integración de la Recepción de Datos con el registro en GNU Health...	45
4.4.4.2.1 Configuración de Suscripción y Contexto	45
4.4.4.2.2 Lógica de Discriminación y Mapeo Dinámico	45
C. Almacenamiento puntual de datos fisiológicos en GNU Health	45
4.4.5 Consolidación e Integración Sistémica del Middleware.....	46
4.5. Implementación de la Interfaz Web de Visualización y Gestión (Etapa 3).....	47
4.5.1 Diseño de la Interfaz de Usuario (UI).....	50
4.5.2 Desarrollo de pantallas principales	54
4.5.2.1 Gestión de Estado y Enrutamiento Modular.....	54
4.5.2.2 Metodología de Acceso a Datos y Visualización	55
4.5.2.3 Diseño final de las pantallas principales.....	55
4.5.3 Desarrollo de pantallas de acceso	58
4.5.3.1 Criptografía y Protección de Credenciales	59
4.5.3.2 Gestión de Registro (Pantalla Register).....	59
4.5.3.3 Recuperación y Actualización (Pantalla Change-password)	60
4.5.4 Desarrollo de API RESTful para interfaz web	61
4.5.5 Integración de API RESTful con la interfaz de usuario	62
4.5.5.1 Protocolo de Comunicación y Flujo de Datos	63
4.5.5.2 Gestión de Estado mediante la variable st.session_state	63
4.5.5.3. Operaciones de Autenticación y Gestión de Usuarios.....	63
4.5.5.4 Visualización de Datos Clínicos y Navegación Dinámica	64
4.6 Implementación del Sistema Integrado (Etapa 4).....	65
4.6.1 Estrategia de separación de funciones en la arquitectura de la API final	68
4.6.1.1 Desarrollo del servicio Merry (API de Enrutamiento y Mensajería MQTT)	68
4.6.1.2 Desarrollo del servicio Sunny Health (API Principal de Integración y Lógica de Negocio).....	69

4.6.2 Integración del Nodo Simulador al ecosistema final	69
4.6.2.1 Desarrollo del módulo de configuración remota (Pagina Settings)	69
4.6.2.1.1 Interfaz de entrada de metadatos clínicos y técnicos	70
4.6.2.1.2 Interfaz de control para la generación de datos	70
4.6.2.1.3 Lógica de tópicos dinámicos y solución de escalabilidad	71
4.6.2.2 Actualización de funciones en el firmware del Nodo Simulador	71
4.6.2.2.1 Implementación de la suscripción a tópicos MQTT (configuración, comando y respuesta).....	71
4.6.2.2.2 Lógica de procesamiento y recepción de instrucciones	72
4.6.2.2.3 Configuración del disparador físico (botón) e indicadores de estado	72
4.6.3 Integración del Middleware y repositorios al sistema final	73
4.6.3.1 Lógica de comunicación inter-servicios para la configuración de parámetros.....	73
4.6.3.2 Lógica de procesamiento bidireccional para la inserción de datos.....	74
4.6.4 Pruebas de consistencia de datos de extremo a extremo	75
4.6.4.1 Escenario 1: Análisis de Trazabilidad Transaccional (Prueba Unitaria) 75	
4.6.4.2 Escenario 2: Evaluación de Consistencia y Segregación de Datos mediante Matriz Combinatoria (N=120)	76
V. Resultados	78
5.1 Trazabilidad de un ciclo de transmisión unitario (Prueba Unitaria)	78
5.1.1 Generación de parámetros y configuración remota	78
5.1.1.1 Descripción del flujo de configuración.....	80
5.1.1.2 Evidencia de ejecución de configuración de parámetros.....	80
5.1.2 Captura del comando desde el Nodo Simulador.....	84
5.1.2.1 Descripción del flujo de enrutamiento de comandos.....	86

5.1.2.2 Evidencia de ejecución de captura de comando desde el Nodo simulador	86
5.1.3 Generación de datos fisiológicos desde el Nodo Simulador.....	89
5.1.3.1 Descripción del flujo de generación de datos	89
5.1.3.2 Evidencia de ejecución de generación de datos desde el Nodo simulador	90
5.1.4 Enrutamiento y procesamiento en el Middleware (API Sunny y API Merry)	95
5.1.4.1 Descripción del flujo de enrutamiento y procesamiento en el Middleware	97
5.1.4.1.1 Almacenamiento en la base de datos espejo (MariaDB)	97
5.1.4.1.2 Almacenamiento en el sistema hospitalario (GNU Health / PostgreSQL).....	98
5.1.4.2 Evidencia de ejecución de enrutamiento y procesamiento de datos	98
5.1.4.3 Evidencia de ejecución almacenamiento en los repositorios de datos..	103
5.1.4.3.1 Evidencia de almacenamiento en el sistema hospitalario (GNU Health / PostgreSQL)	103
5.1.4.3.2 Evidencia de almacenamiento en base de datos MariaDB	106
5.2 Evaluación de consistencia y segregación de datos mediante Matriz Combinatoria (N=120).....	112
5.2.1 Renderizado y evolución clínica en la Interfaz Web	113
5.2.2 Trazabilidad en el sistema hospitalario (GNU Health).....	127
VI. Discusión	129
6.1 Análisis de la validación funcional de la arquitectura propuesta.....	129
6.2 Comparación con el Estado del Arte	131
6.2.1 Comparación con Sistemas Basados en Blockchain (Categoría A)	131
6.2.2 Comparación con Middlewares de Estandarización IoT (Categoría B)	132
6.3 Relación de la propuesta con el marco normativo peruano	133

6.4 Limitaciones del estudio	134
VII. Conclusiones	137
VIII. Referencias Bibliográficas	139
Anexos	145

Índice de Tablas y Figuras

Figura 1. Niveles de interoperabilidad en salud.	15
Figura 2. Representación de la interoperabilidad mejorada mediante HL7 FHIR.	16
Figura 3. Arquitectura del protocolo MQTT.	18
Figura 4. Ecosistema GNU Health	20
Figura 5. GNU HMIS a través de la interfaz de usuario GNU Health Client.	21
Figura 6. Arquitectura API RESTful	24
Figura 7. Diagrama de pines del módulo ESP32.	26
Tabla 1. Recursos hardware usados en el desarrollo de la presente tesis.	30
Figura 8. Diseño funcional del nodo simulador del prototipo biomédico.	33
Figura 9. Diseño funcional del middleware de integración.	36
Tabla 2. Estructura de la base de datos health_monitoring en MariaDB	37
Tabla 3. Usuarios configurados en el programa GNU Health Client	39
Figura 10. Captura de pantalla del listado de entidades en GNU Health Client.	40
Figura 11. Captura de pantalla de la estructura de la tabla gnuhealth_patient_evaluation en DBeaver.	41
Figura 12. Captura de pantalla de la estructura de gnuhealth_patient en DBeaver.	42
Figura 13. Captura de pantalla de la estructura de gnuhealth_healthprofessional en DBeaver.	42
Figura 14. Captura de código fuente de la consulta SQL para la inserción de datos.	44
Figura 15. Diagrama funcional de la interfaz web de visualización y gestión.	49
Figura 16. Diseño inicial de la interfaz de Login.	50
Figura 17. Diseño inicial de la interfaz Home.	51
Figura 18. Diseño inicial de la interfaz Details.	52
Figura 19. Diseño inicial de la interfaz Patients.	53
Figura 20. Diseño inicial de la interfaz Patients_details.	53
Figura 21. Diseño inicial de la interfaz Data_view.	54

Figura 22. Diseño final de la pantalla Home	55
Figura 23. Diseño final de la pantalla Details.....	56
Figura 24. Diseño final de la pantalla Patients	56
Figura 25. Diseño final de la pantalla Patients_details.....	57
Figura 26. Diseño final de la pantalla Data_view (Parte 1).....	57
Figura 27. Diseño final de la pantalla Data_view (Parte 2).....	58
Figura 28. Diseño final de la pantalla Login	59
Figura 29. Diseño final de la pantalla Register.....	60
Figura 30. Diseño final de la pantalla Change-password	61
Tabla 4. Módulos funcionales de la API.....	62
Figura 31. Diagrama funcional del sistema integrado final.....	67
Figura 32. Diagrama de flujo del proceso de configuración remota de parámetros	79
Figura 33. Captura de pantalla de la configuración de metadatos en la página Settings en Streamlit (Parte 1).	81
Figura 34. Captura de pantalla de la configuración de metadatos en la página Settings en Streamlit (Parte 2).	81
Figura 35. Registro de ejecución (logs) en la terminal de la API principal Sunny Health.	82
Figura 36. Registro de ejecución (logs) y suscripción dinámica en la terminal de la API Merry	83
Figura 37. Secuencia de inicialización y enrutamiento dinámico en el monitor serial del ESP32.....	84
Figura 38. Diagrama de flujo del proceso de envío y enrutamiento de comandos al Nodo Simulador.....	85
Figura 39. Captura de pantalla del envío de comando personalizado en la página Settings de Streamlit.	87
Figura 40. Registro de ejecución en la terminal de la API Sunny durante el enrutamiento del comando.....	88

Figura 41. Recepción del comando personalizado en el monitor serial del ESP32.....	88
Figura 42. Diagrama de flujo del proceso de generación y estructuración de datos fisiológicos en el Nodo Simulador.....	89
Figura 43. Procesamiento del comando, asignación en memoria y apertura de estructura JSON en el monitor serial (Parte 1).	91
Figura 44. Inserción de metadatos clínicos y marca de tiempo en el monitor serial del ESP32 (Parte 2).....	93
Figura 45. Ensamblaje de valores clínicos compuestos y cierre del mensaje HL7 FHIR (Parte 3).....	94
Figura 46. Diagrama de flujo del proceso de enrutamiento y procesamiento en el Middleware.	96
Figura 47. Registro de ejecución en la terminal de la API Merry tras la intercepción del mensaje en el tópico de entrada	99
Figura 48. Registro de ejecución en la terminal de la API Sunny confirmando el procesamiento de los repositorios de datos.....	100
Figura 49. Publicación del mensaje de estado transaccional (ACK) desde la API Merry hacia el tópico de salida.	101
Figura 50. Recepción del mensaje de confirmación (ACK) en el monitor serial del ESP32 (Parte 1).....	102
Figura 51. Recepción del mensaje de confirmación (ACK) en el monitor serial del ESP32 (Parte 2).....	102
Figura 52. Captura del registro en la tabla gnuhealth_patient_evaluation mediante DBBeaver (columnas diastolic y evaluation_start).	104
Figura 53. Captura del registro en la tabla gnuhealth_patient_evaluation mediante DBBeaver (columna systolic).....	104
Figura 54. Panel de evaluaciones clínicas del paciente Willy Valdivia en la interfaz de GNU Health Client.	105
Figura 55. Vista detallada del registro de presión arterial en GNU Health Client.	106

Figura 56. Captura del registro en la tabla observations mediante DBever (metadatos de usuario y paciente).....	107
Figura 57. Captura del registro en la tabla observations mediante DBever (metadatos de dispositivo, valores de presión arterial y marca de tiempo autogenerada)).....	107
Figura 58. Panel principal de monitoreo en Streamlit mostrando el historial de registros recientes.	108
Figura 59. Vista detallada de los metadatos del registro clínico en la interfaz web.	109
Figura 60. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 1).	110
Figura 61. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 2).	110
Figura 62. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 3).	111
Figura 63. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 4).	111
Figura 64. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 5).	112
Figura 65. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 1: Tabular).....	114
Figura 66. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 2: Gráfica).....	115
Figura 67. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 1: Tabular).....	115
Figura 68. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 2: Gráfica).	116
Figura 69. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 1: Tabular).....	116
Figura 70. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 2: Gráfica).	117

Figura 71. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria (Parte 1: Tabular).....	117
Figura 72. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria (Parte 2: Gráfica)	118
Figura 73. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 1: Tabular).	118
Figura 74. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 2: Gráfica).	119
Figura 75. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 1: Tabular)	119
Figura 76. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 2: Gráfica)	120
Figura 77. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 1: Tabular).	121
Figura 78. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 2: Gráfica)	121
Figura 79. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 1: Tabular).	122
Figura 80. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 2: Gráfica).	122
Figura 81. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 1: Tabular).	123
Figura 82. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 2: Gráfica).	123
Figura 83. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria (Parte 1: Tabular).....	124
Figura 84. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria (Parte 2: Gráfica)	124

Figura 85. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 1: Tabular).	125
Figura 86. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 2: Gráfica).	125
Figura 87. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 1: Tabular)	126
Figura 88. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 2: Gráfica)	126
Figura 89. Vista de la sección de evaluaciones de la paciente Milagros Alvites en GNU Health Client.	127
Figura 90. Vista de la sección de evaluaciones del paciente Jose Cebrian en GNU Health Client.....	128

Resumen

La integración de dispositivos de Internet de las Cosas Médicas (IoMT) con Historias Clínicas Electrónicas (HCE) presenta barreras de interoperabilidad, especialmente en el ámbito académico, donde los estudiantes de ingeniería biomédica carecen de herramientas accesibles para vincular sus prototipos con sistemas de información reales. La presente tesis tuvo como objetivo desarrollar una API y un sistema de middleware para permitir la interoperabilidad estandarizada, automatizada y de bajo costo entre prototipos biomédicos y la HCE de código abierto GNU Health.

La metodología abarcó el diseño de una arquitectura de microservicios dual basada en REST (APIs Merry y Sunny) que procesa tramas MQTT provenientes de un nodo simulador IoT. Dicho nodo (ESP32) fue programado para generar y estructurar datos fisiológicos en formato JSON empleando el recurso Observation del estándar internacional HL7 FHIR, favoreciendo interoperabilidad semántica. Simultáneamente, se implementó una interfaz web en Streamlit, soportada por una base de datos espejo en MariaDB, para el control remoto del nodo simulador y la visualización y trazabilidad de la información recibida.

El sistema fue validado mediante pruebas funcionales de extremo a extremo, incluyendo una evaluación de consistencia y segregación de datos con 120 transmisiones correspondientes a 24 escenarios experimentales definidos mediante la metodología Input Space Partitioning (ISP) y el criterio All Combinations Coverage (ACoC). Bajo las condiciones controladas del estudio, no se observaron pérdidas de información ni asignaciones cruzadas de datos clínicos, y se verificó el registro de la información fisiológica en GNU Health. Se concluye que la solución desarrollada constituye una herramienta orientada al ámbito académico para experimentar con mecanismos de interoperabilidad e integración entre un nodo simulador biomédico y una HCE de acceso libre.

Palabras clave: Interoperabilidad clínica, HL7 FHIR, Internet de las Cosas Médicas (IoMT), GNU Health, API REST.

Abstract

The integration of Internet of Medical Things (IoMT) devices with Electronic Health Records (EHRs) presents interoperability barriers, especially in the academic setting, where biomedical engineering students lack accessible tools to connect their prototypes to real-world information systems. The objective of this thesis was to develop an API and a middleware system to enable standardized, automated, and low-cost interoperability between biomedical prototypes and the open-source GNU Health EHR.

The methodology involved the design of a dual REST-based microservices architecture (Merry and Sunny APIs) that processes MQTT messages from an IoT simulator node. This node (ESP32) was programmed to generate and structure physiological data in JSON format using the Observation resource of the international HL7 FHIR standard, thereby promoting semantic interoperability. At the same time, a web interface was implemented in Streamlit, supported by a mirror database in MariaDB, for remote control of the simulator node and for the visualization and traceability of the received information.

The system was validated through end-to-end functional tests, including an assessment of data consistency and segregation using 120 transmissions corresponding to 24 experimental scenarios defined using the Input Space Partitioning (ISP) methodology and the All Combinations Coverage (ACoC) criterion. Under the controlled conditions of the study, no information loss or cross-assignment of clinical data was observed, and the recording of physiological data in GNU Health was verified. It is concluded that the developed solution constitutes an academic-oriented tool for experimenting with interoperability and integration mechanisms between a biomedical simulator node and an open-access EHR.

Keywords: Clinical interoperability, HL7 FHIR, Internet of Medical Things (IoMT), GNU Health, REST API.

I. Introducción

El ejercicio profesional del ingeniero biomédico se fundamenta en el diseño y prototipado de soluciones tecnológicas orientadas a resolver las problemáticas encontradas en los establecimientos de salud. Durante su formación académica, los estudiantes adquieren los conocimientos técnicos necesarios para desarrollar prototipos funcionales e innovadores, priorizando frecuentemente la accesibilidad y el bajo costo para responder a las necesidades reales de los usuarios. Sin embargo, la educación biomédica actual enfrenta un desafío crítico: existe una brecha entre el desarrollo aislado de estos prototipos y su viabilidad dentro del ecosistema digital sanitario moderno.

En la actualidad, la práctica clínica no solo requiere dispositivos capaces de medir o monitorear variables fisiológicas de forma local, sino que exige que estos datos se transmitan e integren fluidamente en las redes de información hospitalaria. Por lo tanto, proveer a los estudiantes de herramientas que les permitan incorporar capacidades de interoperabilidad en sus prototipos desde las aulas es una necesidad urgente para prepararlos ante los retos del entorno clínico real. En este ecosistema interconectado, el repositorio central que actúa como destino definitivo de toda esta información fisiológica es la historia clínica del paciente.

1.1 Problemática

Las historias clínicas son documentos médicos esenciales que registran de manera cronológica y detallada la información relacionada con la salud de un paciente. Este registro incluye datos relevantes como antecedentes familiares, exploraciones previas, diagnósticos anteriores y los tratamientos recibidos a lo largo de su vida, constituyéndose así en una herramienta clave para garantizar la calidad de la atención brindada (1). Además, las historias clínicas tienen el potencial de convertirse en una valiosa fuente de información para diversos procesos en el ámbito médico, como los estudios epidemiológicos y actividades de investigación (2).

En la actualidad, existen dos modalidades principales para la elaboración de historias clínicas: las historias clínicas manuales (HCM) y las historias clínicas electrónicas (HCE). En el Perú, la mayoría de estos registros continúan realizándose en formato manual por el personal asistencial (3). Sin embargo, el uso y la gestión de las HCM en entornos hospitalarios generan múltiples problemas que afectan tanto la calidad de la atención brindada a los pacientes como la satisfacción del personal de salud (4,5).

Por ejemplo, una tesis realizada en el Policlínico Parroquial Juan Pablo II, en Lima Norte, evidenció tiempos elevados de espera y atención asociados a la dificultad para localizar la HCM específica de cada paciente. Asimismo, se reportó una alta incidencia de pérdida y omisión de datos durante su elaboración, y una gran insatisfacción del personal debido al tiempo requerido para redactar y ubicar las historias clínicas dentro de la institución (4).

Estos problemas también se observan en otras instituciones, como el Instituto Médico de la Mujer, donde se ha reportado que el uso de formatos físicos conlleva riesgos de pérdida de información clínica. Asimismo, se identificó que la mayoría de HCM son almacenadas en ubicaciones externas al área de atención, lo cual dificulta el acceso oportuno del personal asistencial (5). La recurrencia de estos problemas, propios de la naturaleza de las HCM, resalta la necesidad de implementar soluciones digitales para la gestión de la información clínica. En este contexto, las HCE adquieren una relevancia crucial al ofrecer una solución capaz de superar estas limitaciones.

La HCE se define como un repositorio digital que almacena y organiza la información médica de un paciente a lo largo de todo su recorrido clínico, actualizándose en tiempo real para su consulta por el personal de salud autorizado. Entre sus principales funcionalidades se incluyen el acceso oportuno a la información clínica, el envío y gestión de solicitudes médicas, la incorporación de sistemas de apoyo a la decisión, la aplicación de protocolos de seguridad y la provisión de herramientas de comunicación entre distintos servicios y profesionales de salud (6).

A partir de estas características, se han desarrollado diversos estudios con el objetivo de determinar los beneficios de esta herramienta en un entorno clínico (7,8). En este sentido, una revisión encontró que la adopción de historias clínicas electrónicas en servicios como la Unidad de Cuidados Intensivos (UCI) puede reducir la mortalidad, la estancia hospitalaria y la incidencia de infecciones asociadas a catéter central (7). Asimismo, una revisión sistemática realizada en España identificó que las HCE facilitan el acceso del personal a la información, mejoran la trazabilidad de los datos, disminuyen el riesgo de pérdida de información y reducen errores de lectura y transcripción (8).

No obstante, a pesar de los múltiples beneficios asociados a su adopción, la realidad peruana presenta limitaciones en la implementación de las HCE. A pesar de que existen normativas como la Agenda Digital del Sector Salud hacia el 2030 y la Resolución Ministerial N.º 553-2025-MINSA, las cuales impulsan el despliegue obligatorio de HCE

en los establecimientos del primer nivel de atención (9,10), la implementación de estas herramientas aún no es masiva en el país (11).

Además, se ha observado que las HCE implementadas por el Ministerio de Salud del Perú (MINSA) no cumplen plenamente con los estándares de una historia clínica electrónica, sino que corresponden más bien a sistemas informatizados. Esta situación se genera debido a la falta de interoperabilidad entre establecimientos, la ausencia de módulos administrativos completos y el uso limitado de la firma digital para validar la información clínica (12). Aun así, la mejora de los sistemas actuales sigue siendo una prioridad a nivel nacional, con iniciativas orientadas a incorporar estándares internacionales en el desarrollo e implementación de HCE (13).

En paralelo a los impulsos nacionales por mejorar los sistemas, los dispositivos del Internet de las cosas médicas (IoMT) han adquirido una relevancia en el campo de la salud. Los dispositivos IoMT se definen como dispositivos capaces de capturar constantemente variables fisiológicas de un paciente y transmitirlos a una plataforma web para su continuo seguimiento (14,15). Entre las principales ventajas que ofrece este tipo de sistemas están el monitoreo constante de un paciente, lo cual deriva en la detección rápida de anomalías en el estado de salud, especialmente en personas con enfermedades crónicas como diabetes o con afecciones cardíacas (16,17), así como la mejora de la eficiencia en la atención médica mediante la automatización de procesos (16).

No obstante, el potencial de estos dispositivos puede aprovecharse de mejor manera cuando se vinculan con sistemas de información médica. En particular, la integración de estos dispositivos IoMT con las HCE puede potenciar significativamente la utilidad clínica de ambos sistemas, ya que no solo se permite un monitoreo continuo del paciente, sino que también posibilita la emisión de alertas inteligentes, la visualización de tendencias en el estado de salud y la priorización de la atención según la condición clínica (15). Además, esta integración garantiza un registro automático de los valores fisiológicos en la HCE, lo que reduce los errores asociados a la transcripción manual, un problema previamente reportado en este tipo de sistema de información (18).

Sin embargo, los dispositivos IoMT presentan limitaciones importantes, una de las principales es la falta de estándares de interoperabilidad que permitan transmitir información entre sistemas heterogéneos (14,19). A pesar de que en el Perú existen esfuerzos para adoptar estándares internacionales como HL7 FHIR, la fragmentación del sistema de salud, dividido en sectores público, privado y mixto, dificulta la

implementación homogénea de estas iniciativas (14). Por ello, y con el fin de aprovechar plenamente el potencial de las HCE junto con los beneficios documentados de la integración con tecnologías IoMT, resulta urgente desarrollar dispositivos capaces de transmitir sus datos de manera estandarizada hacia los sistemas de información en salud. En consecuencia, investigaciones futuras en el campo biomédico deberían explorar soluciones orientadas a mejorar esta integración. No obstante, en la práctica, la mayoría de los dispositivos biomédicos desarrollados por estudiantes de ingeniería biomédica no incorporan estas funcionalidades de interoperabilidad debido a factores externos.

Uno de los principales factores es la limitada accesibilidad a HCEs capaces de recibir y adherir información de dispositivos externos. Una alternativa es el uso de HCE de código abierto; sin embargo, muchos de estos sistemas aún se encuentran en fase de desarrollo y presentan capacidades restringidas para realizar pruebas con dispositivos externos (20).

En este contexto, se evidencia que los estudiantes de ingeniería biomédica no cuentan con los recursos necesarios para integrar capacidades de interoperabilidad a sus prototipos durante la fase de diseño y prototipado. Como consecuencia, los prototipos desarrollados suelen ser menos escalables tecnológicamente, ya que no logran integrarse a los sistemas de información utilizados en las instituciones de salud. Por todo ello, se identifica el siguiente problema central: La falta de integración entre prototipos biomédicos desarrollados por estudiantes de ingeniería biomédica y HCE de acceso libre.

1.2 Propósito del estudio

El propósito principal de este estudio es desarrollar una solución tecnológica que permita la interconexión de prototipos biomédicos estudiantiles con una HCE de acceso libre, utilizando estándares de interoperabilidad reconocidos (como HL7 FHIR) y herramientas de software disponibles en entornos de bajo costo.

Al lograr esta integración, la presente investigación busca responder a la brecha tecnológica existente en la formación académica. El valor de este estudio radica en desarrollar una herramienta que permita a los estudiantes de ingeniería biomédica experimentar con conceptos de interoperabilidad y sistemas de información en salud durante el desarrollo de prototipos biomédicos. De esta manera, se busca proporcionar una base tecnológica accesible que contribuya a incorporar estos conceptos durante su formación académica y favorezca el desarrollo futuro de prototipos con mayores capacidades de integración con sistemas de información en salud.

Para cumplir con este propósito, el presente documento se estructura de la siguiente manera: esta primera sección introductoria concluye con la revisión del estado del arte. El Capítulo II establece el marco teórico, abordando conceptos clave como la interoperabilidad en salud, el estándar HL7 FHIR, la metodología Input Space Partitioning (ISP), protocolos de comunicación y el ecosistema GNU Health. El Capítulo III define la pregunta de investigación y los objetivos trazados. El Capítulo IV detalla exhaustivamente los materiales y métodos empleados para la implementación técnica en las diferentes fases del desarrollo del sistema. El Capítulo V expone los resultados obtenidos durante la validación funcional del sistema, incluyendo las pruebas de trazabilidad, consistencia y segregación de la información biomédica. Finalmente, los Capítulos VI y VII presentan la discusión de los hallazgos, y las conclusiones de la investigación, respectivamente.

1.3 Estado del Arte

Con el objetivo de identificar las soluciones tecnológicas desarrolladas para la interoperabilidad estandarizada entre dispositivos IoMT y sistemas de información de salud, se decidió realizar una revisión bibliográfica centrada en estas tecnologías. En este sentido, se buscaba identificar principalmente cuatro aspectos clave de cada estudio los cuales son: metodología, resultados, ventajas y limitaciones. Al considerar estos criterios, se elaboró un cuadro comparativo de cada estudio que reúne los principales hallazgos de cada trabajo, el cual se incluye en el **Anexo A**.

A partir del análisis del cuadro comparativo, se identificaron dos categorías principales entre todos los estudios recopilados. La primera categoría corresponde a sistemas basados en blockchain para el intercambio de información clínica. En esta categoría, se busca aprovechar las características del blockchain y se utiliza como medio principal para transmitir la data necesaria para acceder a la información clínica. La segunda categoría recopila sistemas de integración y estandarización de datos clínicos. En esta categoría, se tienen estudios que buscan la forma de estandarizar los datos transmitidos para facilitar su uso e interoperabilidad. En las siguientes secciones se describen los estudios de cada categoría, destacando sus principales aportes, limitaciones y elementos comunes, con el fin de contextualizar y fundamentar el estado del arte de la presente tesis.

1.3.1 Categoría A: Sistemas basados en blockchain para intercambio de información clínica

Dentro de esta categoría, uno de los primeros estudios identificados que emplea este enfoque es el desarrollado por Sonkamble et al. (21). Los autores proponen una arquitectura basada en blockchain con un modelo de almacenamiento híbrido, donde la información clínica de mayor tamaño se conserva en una base de datos off-chain (MongoDB), mientras que los metadatos y los hashes se registran on-chain. Además, el sistema está diseñado bajo un enfoque centrado en el paciente, otorgándole control sobre su HCE y la capacidad de gestionar los permisos de acceso. Entre las ventajas del estudio destaca que el almacenamiento híbrido mejora los tiempos de respuesta y evita la sobrecarga de la cadena de bloques. No obstante, su rendimiento no fue evaluado en escenarios exigentes, ya que no se realizaron pruebas adicionales de escalabilidad ni de resiliencia frente a variaciones en la carga del sistema.

Por su parte, otro estudio, denominado *A Patient-Centric Health Information Exchange Framework Using Blockchain Technology* (22), empleó una blockchain privada para transmitir la información necesaria que permite descifrar los registros clínicos entre un conjunto de hospitales. El sistema incorpora dos módulos principales: uno encargado de conectar las bases de datos institucionales y otro orientado a gestionar los permisos de visualización otorgados por el paciente. La principal ventaja identificada es que, al estar centrado en el paciente, el intercambio de información se vuelve más flexible y no depende de un servidor central. No obstante, incluso en un entorno controlado, se observaron colas de transacción derivadas de las limitaciones de procesamiento inherentes a la blockchain utilizada.

En esta misma línea, Bae et al. (23) desarrollaron una plataforma denominada *HealthPocket*, la cual, basada en blockchain, utiliza un sistema de consentimiento dinámico. Cuando el paciente aprueba el acceso a su información clínica, el sistema obtiene los datos a través de las API de los hospitales y los transforma al estándar HL7 FHIR. Entre las principales ventajas del enfoque, destaca que el uso de blockchain permite disponer de trazabilidad completa sobre los accesos realizados. Asimismo, la adopción de HL7 FHIR facilita que la plataforma pueda interoperar con otros sistemas alineados con dicho estándar. No obstante, aunque la plataforma permite transmitir la información clínica, estos datos no se integran directamente en las historias clínicas electrónicas de los hospitales. Por otra parte, si bien las pruebas de usabilidad mostraron

altos niveles de satisfacción general, los indicadores de facilidad de uso alcanzaron únicamente valores aceptables.

Con un enfoque similar, Akkaoui et al. (24) desarrollaron una arquitectura híbrida que combina edge computing y blockchain para el intercambio de datos. En este enfoque, los nodos en el borde se encargan de procesar y validar la información antes de su transmisión, además de generar los metadatos necesarios para acceder a los datos completos, los cuales son posteriormente incorporados a la cadena de bloques. Los resultados mostraron que esta arquitectura ofrece un rendimiento superior en comparación con el uso exclusivo de una blockchain tradicional. Asimismo, el hecho de mantener únicamente los metadatos en la cadena facilita la atención de múltiples solicitudes en paralelo. Sin embargo, aún se requiere evaluar su desempeño en una red pública de blockchain y estimar el costo total de su implementación. Cabe señalar, además, que el sistema carece de un estándar internacional para el manejo de datos clínicos, lo cual podría dificultar su integración con futuras soluciones en el ámbito de la salud.

En comparación con los estudios previos, el trabajo titulado Electronic Health Records Sharing Model Based on Blockchain With Checkable State PBFT Consensus Algorithm propone un sistema en el que los datos clínicos permanecen descentralizados (25). Para posibilitar el intercambio de información, se emplea el mecanismo de consenso Checkable-State PBFT (CS-PBFT), el cual valida una transacción cuando se alcanza la mayoría entre los nodos activos, descartando aquellos que presenten un estado inválido. La transacción registrada en la blockchain incorpora metadatos que permiten a la entidad solicitante acceder al sistema donde se almacena el dato original. De esta manera, se logró reducir el tráfico entre nodos y mejorar la eficiencia del consenso. No obstante, el rendimiento real del sistema aún no ha sido evaluado en entornos con datos clínicos auténticos. Además, las pruebas se llevaron a cabo únicamente en una red privada, por lo que su comportamiento en redes públicas sigue siendo incierto. Finalmente, el sistema no emplea estándares internacionales para el manejo de datos clínicos, lo que limita su escalabilidad y dificulta su interoperabilidad con otros sistemas de salud.

Al analizar en forma conjunta todos los estudios de esta categoría, se evidencia que la tecnología blockchain ofrece múltiples beneficios, entre ellos la trazabilidad, seguridad, control de acceso y la resiliencia del sistema. Asimismo, se observa que su eficiencia puede incrementarse al combinarla con enfoques complementarios como el

almacenamiento híbrido, los mecanismos de consentimiento dinámico y la tecnología edge computing. La estrategia en común entre las propuestas es evitar almacenar la información clínica completa en la cadena de bloques, en su lugar, se registran únicamente metadatos que permiten acceder posteriormente a los datos completos. Esta estrategia busca reducir la sobrecarga en la blockchain y mejorar los tiempos de respuesta del sistema. Cabe resaltar un caso como *HealthPocket*, que incorpora el estándar internacional HL7 FHIR (23), lo que abre la posibilidad de una integración futura más amplia con otros sistemas de información en salud.

Sin embargo, también se identifican limitaciones comunes entre los estudios revisados. En primer lugar, la mayoría aún se encuentra en fases tempranas de desarrollo, con evaluaciones realizadas exclusivamente en entornos simulados o redes privadas. Por lo que, el rendimiento real y eficiencia de los sistemas propuestos siguen siendo inciertos. Por otro lado, si bien algunos sistemas permiten el intercambio de información entre instituciones, no integran plenamente los datos recibidos en las HCE de destino, lo que evidencia la ausencia de mecanismos completos de incorporación de la información transmitida. Finalmente, salvo un caso puntual, la mayoría de propuestas no utilizan estándares internacionales de interoperabilidad como HL7 FHIR, lo que restringe su escalabilidad, compatibilidad e integración futura con plataformas clínicas reales.

1.3.2 Categoría B: Sistemas de integración y estandarización de datos clínicos.

En esta categoría, el primer estudio identificado es el desarrollado por Mukhiya et al.(26), el cual propone una arquitectura que incorpora una capa intermedia basada en GraphQL. En este enfoque, los datos clínicos almacenados en OpenMRS son transformados al formato HL7 FHIR cada vez que un usuario los solicita. Una vez estandarizados, dichos datos se almacenan en un servidor FHIR y se visualizan en las aplicaciones cliente que realizaron la consulta. Para validar el sistema, se ejecutaron satisfactoriamente diez operaciones CRUD sobre los recursos HL7 FHIR generados, demostrando que el uso de GraphQL mejora la precisión y eficiencia en la construcción de consultas hacia servidores REST. Asimismo, la adopción de HL7 FHIR contribuye a lograr interoperabilidad semántica entre plataformas heterogéneas. Sin embargo, el estudio presenta algunas limitaciones. No aborda en profundidad los mecanismos de privacidad y seguridad de los datos; además, al encontrarse en una etapa temprana de desarrollo, la validación no

incluyó datos clínicos reales. Finalmente, no se realizaron pruebas de usabilidad, lo cual podría influir en su adopción práctica en entornos clínicos.

Con el mismo enfoque de estandarizar datos provenientes de sistemas heterogéneos, el estudio denominado “Towards Medical Data Interoperability Through Collaboration of Healthcare Devices” propone una arquitectura basada en contenedores de software que reciben la información proveniente de distintos dispositivos médicos y la trasladan a un middleware central (27). Este middleware se encarga de estandarizar según la semántica propuesta por IEEE 11073 y de gestionar simultáneamente la llegada de información desde múltiples contenedores. Una vez procesados, los datos estandarizados se consolidan en un servidor central para su posterior acceso. Entre las principales ventajas del enfoque, destaca que el uso de la semántica IEEE 11073 permite transmitir información con significado clínico consistente, y que el middleware demostró capacidad para manejar datos provenientes de varios dispositivos en paralelo. Sin embargo, la principal limitación recae en la capa de contenedores, ya que la diversidad de dispositivos implicaría desarrollar un contenedor específico para cada uno, lo cual complicaría la escalabilidad y despliegue de la arquitectura. Además, el estudio no aborda en profundidad aspectos de privacidad y seguridad, y las validaciones realizadas se limitaron a entornos controlados, por lo que se requieren estudios adicionales para evaluar su desempeño en escenarios reales.

En esta misma línea de investigación, Ray et al. (28) proponen una arquitectura de interoperabilidad conformada por un gateway y un middleware, que actúan como capas intermedias entre dispositivos IoMT y los sistemas de información. El gateway se encarga de normalizar los datos provenientes de múltiples dispositivos IoMT mientras que el middleware estandariza dichos datos en recursos HL7 FHIR. Al ser transformados a dicho estándar, estos son almacenados en un servidor FHIR que funciona como capa de acceso. Entre las principales ventajas reportadas destaca el uso de HL7 FHIR como modelo canónico, lo que otorga al sistema el potencial de comunicar dispositivos IoMT con plataformas clínicas actuales. Asimismo, al no depender del tipo de dispositivo, el middleware ofrece versatilidad para trabajar con múltiples equipos heterogéneos. Sin embargo, se requieren investigaciones adicionales, ya que las validaciones se realizaron únicamente en un entorno controlado. Además, los aspectos de seguridad y privacidad no fueron abordados en profundidad, a pesar de tratarse de datos clínicos.

A diferencia de los estudios anteriores, Malathi et al. (29) proponen una arquitectura que no emplea un estándar canónico común para todos los sistemas. En su propuesta, se utilizan capas de colaboración distribuidas en el borde (edge/fog), que actúan como nodos responsables de transformar los datos desde el estándar del sistema emisor hacia el estándar del sistema receptor. Este enfoque reduce la carga computacional destinada a la nube y mejora el tráfico de datos hacia ella. Entre sus principales ventajas, destaca la capacidad de interoperar información entre distintos estándares sin necesidad de adoptar uno único y universal. Además, se demostró que la arquitectura puede incorporar un mayor número de dispositivos manteniendo un rendimiento adecuado. No obstante, al no emplear un estándar internacional como HL7 FHIR, la interoperabilidad lograda es principalmente funcional y no plenamente semántica. Asimismo, para que el sistema opere correctamente, se requiere que los dispositivos emisores cuenten con capacidad de procesamiento cercana, lo cual puede resultar complejo en escenarios con un gran número de dispositivos. Finalmente, las pruebas fueron realizadas en un entorno simulado, por lo que se recomienda profundizar en evaluaciones adicionales para validar su desempeño en contextos más exigentes.

Al analizar los estudios incluidos en esta categoría, se identifican ventajas en términos de interoperabilidad, gracias al uso de mecanismos de normalización y estandarización que permiten integrar datos clínicos provenientes de sistemas heterogéneos. Tecnologías como GraphQL, middlewares de estandarización y gateways especializados resultan útiles, ya que permiten consolidar distintos tipos de datos sin comprometer el rendimiento del sistema. Asimismo, estrategias como el procesamiento en el borde (edge/fog) reducen el tráfico hacia la nube o servidores centrales, mejorando la eficiencia global. Por otro lado, la mayoría de estudios emplea estándares internacionales de interoperabilidad, como HL7 FHIR o IEEE 11073, los cuales otorgan una estructura semántica clara a los datos y añaden significado clínico al contenido transmitido. Esto no solo facilita su integración, sino que también proporciona escalabilidad al permitir que, en un futuro, estas soluciones puedan comunicarse con sistemas de información en salud más sofisticados que adopten los mismos estándares.

Sin embargo, también se identifican limitaciones comunes. En primer lugar, la mayoría de los estudios fueron validados únicamente en entornos controlados, por lo que se requieren evaluaciones en escenarios clínicos reales para determinar su rendimiento y robustez. Además, muchos enfoques dependen de una capa intermedia (como middleware

o múltiples contenedores), lo que, aunque facilita la transformación de datos, también introduce un punto crítico de falla que podría interrumpir todo el flujo de información. En términos de seguridad y privacidad, varios estudios solo mencionan mecanismos básicos de autenticación, sin abordar estrategias más completas para proteger datos sensibles. Finalmente, en los casos donde no se emplearon estándares internacionales, la interoperabilidad alcanzada fue principalmente funcional, lo que subraya la importancia de adoptar este tipo de estándares para garantizar la compatibilidad futura.

1.3.3 Síntesis del estado del arte

Al analizar de manera global todos los estudios revisados, se identifican fortalezas y debilidades relevantes en cada enfoque orientado a la interoperabilidad de datos clínicos. En la categoría A, los sistemas basados en blockchain destacan por garantizar trazabilidad, seguridad y control de acceso; sin embargo, su escalabilidad suele requerir mayores recursos computacionales, lo que incrementa el costo de implementación. Además, salvo un caso puntual, la mayoría de propuestas no incorpora estándares de interoperabilidad, lo que puede comprometer su compatibilidad con sistemas clínicos futuros. En la categoría B, los sistemas de normalización y estandarización demostraron ser eficaces para interoperar datos entre plataformas heterogéneas. El uso de estándares como HL7 FHIR o IEEE 11073 asegura interoperabilidad semántica y ofrece un mayor potencial de integración con sistemas de información en salud. No obstante, la dependencia de capas intermedias (middleware, gateways o contenedores) introduce puntos críticos de posible fallo, y persiste la necesidad de incorporar mecanismos más robustos de seguridad y privacidad. En conjunto, ambas estrategias evidencian avances importantes hacia la interoperabilidad, pero la mayoría de soluciones continúa en etapas tempranas de desarrollo y no se ha evaluado plenamente en entornos clínicos reales. Es importante resaltar que, si bien los estudios garantizan la transmisión de datos, entre los estudios revisados no se identificó ninguna propuesta que integre directamente información fisiológica proveniente de dispositivos médicos dentro de una HCE institucional. Con base en la literatura revisada, aún no se identificaron soluciones suficientemente maduras que permitan integrar datos de dispositivos médicos en HCE reales de manera automatizada, segura y estandarizada. Esta brecha justifica la necesidad de desarrollar propuestas que combinen seguridad, estandarización y aplicabilidad clínica real, lo cual respalda plenamente la pertinencia del presente trabajo de investigación.

II. Marco teórico

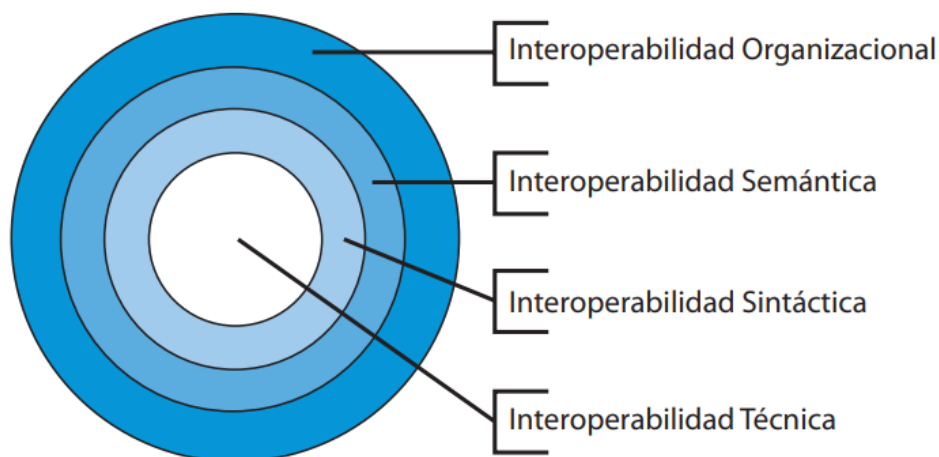
En el presente capítulo se exponen las bases teóricas y los conceptos fundamentales que sustentan el desarrollo del prototipo, permitiendo comprender la relevancia de las herramientas seleccionadas para la integración de prototipos biomédicos con sistemas de registro clínico.

2.1 Interoperabilidad en Salud

La interoperabilidad en el ámbito sanitario se define como la capacidad de diferentes sistemas de información, dispositivos y aplicaciones para acceder, intercambiar e integrar datos de salud de manera coordinada. Según la literatura especializada y organismos internacionales, se distinguen cuatro niveles de interoperabilidad, cada uno de los cuales aporta capacidades progresivas al sistema: (30,31)

- Interoperabilidad técnica (fundamental): Representa el nivel más básico y se refiere a la infraestructura física y lógica necesaria para que dos o más sistemas intercambien información. Se enfoca en el establecimiento de la conexión, los protocolos de comunicación y el hardware involucrado, limitándose esencialmente al intercambio de datos entre máquinas.
- Interoperabilidad sintáctica (estructurada): Este nivel garantiza que los mensajes transmitidos posean un formato y una estructura común que el receptor pueda procesar. El objetivo es proporcionar un orden lógico que facilite la interpretación técnica de los datos. Entre los formatos de intercambio de alto nivel más utilizados destacan XML, JSON y ASN.1.
- Interoperabilidad semántica: Es el nivel crítico para la práctica clínica. No solo asegura la recepción del dato, sino que garantiza que el significado de la información sea el mismo para el emisor y el receptor. Esto se logra mediante el uso de terminologías médicas estandarizadas y vocabularios comunes, como LOINC (para observaciones de laboratorio y clínica) y SNOMED-CT (para terminología médica integral) (32).
- Interoperabilidad organizacional: Constituye el nivel superior y se refiere a la capacidad de distintas organizaciones para colaborar de manera efectiva mediante procesos de gobernanza, marcos legales y políticas de confianza. Se permite la transferencia de datos con valor clínico a través de flujos de trabajo heterogéneos, dependiendo estrictamente del éxito de los tres niveles anteriores.

Figura 1. Niveles de interoperabilidad en salud.



Fuente: Tomado de la ref. (30).

Al analizar esta jerarquía, se concluye que los desarrollos tecnológicos orientados a la transmisión de datos clínicos deben aspirar, como mínimo, a alcanzar la interoperabilidad semántica. La ausencia de este nivel de integración puede comprometer la eficiencia de herramientas avanzadas como la HCE, limitando la utilidad de la información almacenada (31). Asimismo, cabe resaltar que la transición hacia la interoperabilidad organizacional suele estar supeditada a factores políticos y administrativos más que a capacidades técnicas adicionales. En virtud de lo anterior, la presente tesis se enfoca en alcanzar un nivel de interoperabilidad semántica en la API desarrollada. Para ello, se implementó el uso de terminologías estandarizadas como LOINC y se adoptaron estructuras de datos en formato JSON bajo el estándar HL7 FHIR.

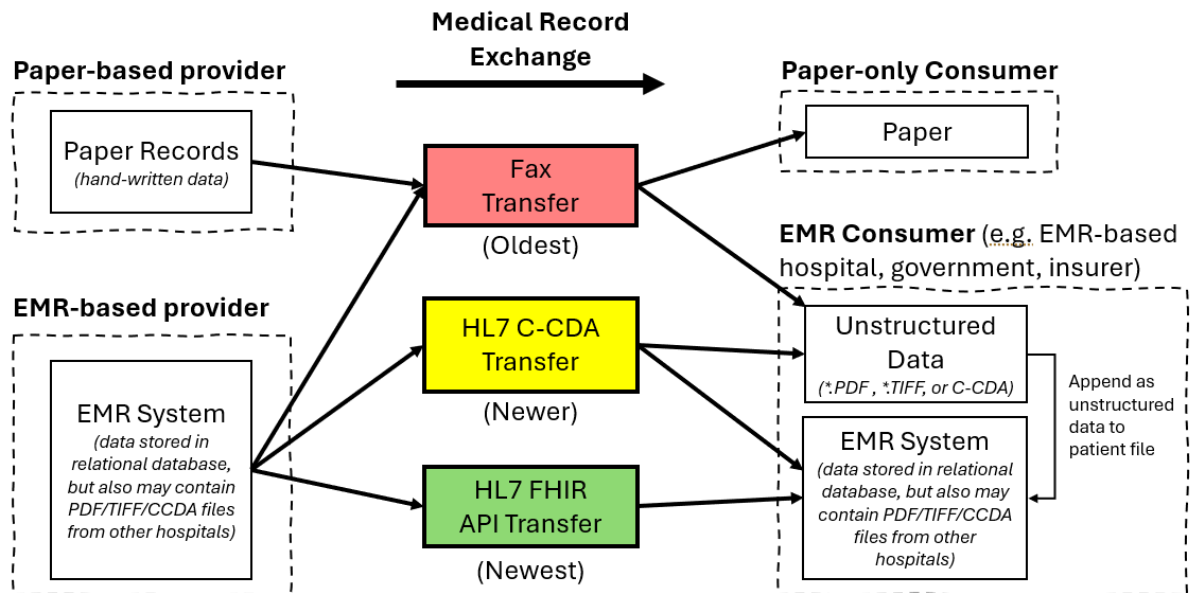
2.2 Estándar HL7 FHIR

El estándar *Fast Healthcare Interoperability Resources* (FHIR) se define como un marco de trabajo de código abierto para la interoperabilidad en salud, desarrollado por la organización HL7 (*Health Level Seven International*) (33). FHIR surge con el propósito de superar las limitaciones de estándares previos, facilitando un intercambio de datos ágil y eficiente entre sistemas de información de salud heterogéneos.

A diferencia de las versiones predecesoras de HL7, FHIR introduce el concepto de recursos. Estos representan las unidades de información que modelan conceptos clínicos (como pacientes, diagnósticos o mediciones), los cuales poseen la flexibilidad necesaria para adaptarse a diversos escenarios asistenciales (34,35). Otra diferencia relevante es que FHIR permite una transmisión de datos más directa entre sistemas de información en

salud, preservando la estructura interna de los datos (36), lo que facilita una interoperabilidad e integración más eficientes. A continuación, se muestra un diagrama que ejemplifica este proceso de intercambio de información

Figura 2. Representación de la interoperabilidad mejorada mediante HL7 FHIR.



Fuente: Tomado de la ref. (36).

Asimismo, este estándar destaca por su versatilidad para integrarse en arquitecturas centradas tanto en el proveedor de salud como en el paciente, apoyándose en tecnologías web modernas como las APIs RESTful y formatos de serialización de datos como JSON y XML (33,34). Además de la gestión clínica, el potencial de HL7 FHIR se extiende hacia la investigación y el análisis de datos masivos. Su estructura modular favorece la transición rápida de información, permitiendo el seguimiento de pacientes, especialmente en el contexto de enfermedades crónicas (34).

En el desarrollo de la presente tesis, se emplea específicamente el recurso Observation. Este recurso es fundamental para el monitoreo del progreso clínico, ya que está diseñado para capturar y transmitir mediciones puntuales de variables fisiológicas y otros diagnósticos (37). Debido a esta naturaleza, constituye el componente más adecuado para la transmisión de datos provenientes de prototipos biomédicos. Para la implementación técnica, se optó por el formato JSON (*JavaScript Object Notation*), dada su ligereza y

eficiencia en el procesamiento, características esenciales para la comunicación con microcontroladores y sistemas embebidos de recursos limitados.

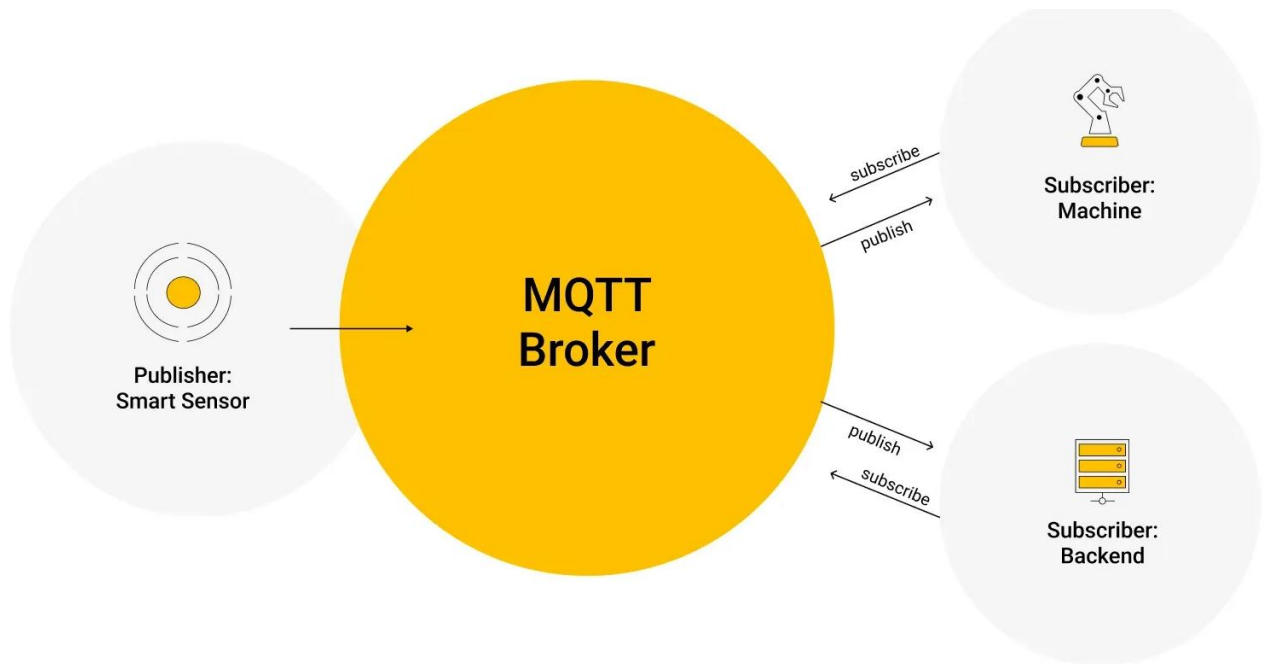
2.3 Protocolo MQTT

El protocolo Message Queuing Telemetry Transport (MQTT) es un estándar de mensajería ligero basado en el modelo de publicación/suscripción. Está diseñado específicamente para la comunicación entre dispositivos con recursos restringidos (baja capacidad de procesamiento y memoria) y redes con ancho de banda limitado o alta latencia, lo que lo convierte en la solución adecuada para el ecosistema del Internet de las Cosas (IoT) y el monitoreo remoto (38,39). La arquitectura de MQTT se fundamenta en el desacoplamiento entre el emisor y el receptor, permitiendo que la información fluya a través de un nodo central. Para ello, se definen los siguientes roles y conceptos fundamentales (38,40):

- Broker MQTT (Servidor): Actúa como el centro de gestión del protocolo. Su función principal es recibir los mensajes de los clientes publicadores, filtrarlos según su jerarquía de tópicos y distribuirlos de manera eficiente a todos los clientes suscritos a dichos temas.
- Publicador (Cliente): Es el dispositivo encargado de generar y enviar los datos hacia un tópico específico dentro del broker. En este flujo, el publicador no necesita conocer quiénes son los receptores finales de la información.
- Suscriptor (Cliente): Es el equipo que establece una conexión con el broker para "escuchar" tópicos de interés. Una vez suscrito, recibirá automáticamente cualquier mensaje que sea publicado bajo esa etiqueta.
- Tópico: Es una estructura de cadena de caracteres (generalmente jerárquica) que sirve como filtro para organizar los mensajes dentro del broker.

El funcionamiento de esta arquitectura se ilustra en la Figura 3. En ella se aprecia cómo un sensor inteligente, actuando como publicador, envía sus datos al broker, y estos son distribuidos a todos los clientes suscritos (una máquina y un backend).

Figura 3. Arquitectura del protocolo MQTT.



Fuente: Tomando de la ref. (41)

Debido a su eficiencia operativa, MQTT representa una alternativa técnica adecuada para el intercambio de datos en sistemas embebidos. En la presente tesis, se emplea este protocolo para establecer el canal de comunicación entre los prototipos biomédicos y la Raspberry Pi 4 donde reside la HCE. Además, se optó por la generación de recursos HL7 FHIR en formato JSON, garantizando que el mensaje enviado sea ligero y de fácil manejo.

2.4 Ecosistema GNU Health

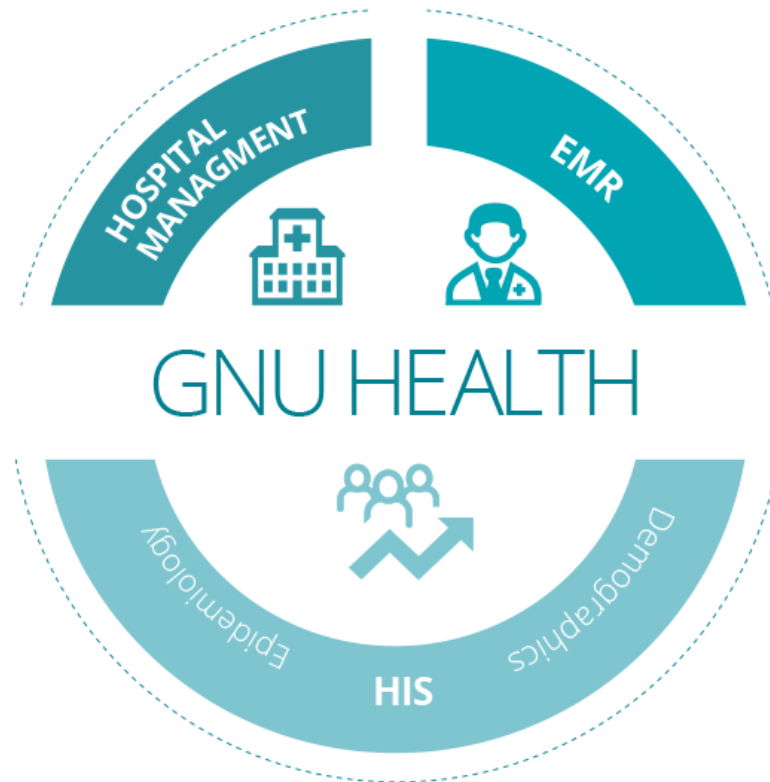
El ecosistema GNU Health comprende un conjunto de soluciones digitales diseñadas para el apoyo de instituciones sanitarias, profesionales de la salud y organismos gubernamentales (42). Este proyecto, liderado por la organización GNU Solidario, se fundamenta en el software libre, lo que garantiza el acceso abierto y la gratuidad de sus herramientas. El ecosistema está integrado principalmente por el Sistema de Gestión Hospitalaria (HMIS), el Sistema de Gestión de Laboratorios (LIMS) y el Historial Clínico Personal (MyGNUHealth) (42). Dada la naturaleza de la presente tesis, centrada en la interoperabilidad de prototipos con HCE de acceso libre, se seleccionó la herramienta GNU Health HMIS. Si bien existen otras HCEs de código abierto, como OpenEMR y OpenMRS, la presente investigación no tuvo como objetivo realizar un análisis comparativo entre dichas plataformas, sino validar una arquitectura de interoperabilidad

sobre una historia clínica funcional. En este contexto, GNU Health HMIS fue seleccionado por constituir una solución de software libre que satisface los requerimientos metodológicos del estudio. Adicionalmente, en el presente estudio, GNU Health HMIS pudo desplegarse sobre hardware de bajo costo, como la Raspberry Pi 4 empleada en la investigación, lo que favoreció la reproducibilidad del entorno experimental en un contexto académico.

Esta plataforma organiza sus funciones en diversas áreas críticas para la atención sanitaria, entre las que destacan (43)

- Demografía y Comunidad: Gestión de variables demográficas y determinantes socioeconómicos de los individuos y familias.
- Gestión del Paciente: Núcleo de la HCE donde se registran evaluaciones médicas, antecedentes, alergias y hospitalizaciones. En este estudio se hace énfasis en esta área, específicamente en el registro de evaluaciones para el monitoreo de variables fisiológicas.
- Centro de Salud: Administración de recursos institucionales, finanzas y gestión de proveedores.
- Laboratorio e Imágenes: Gestión de órdenes de diagnóstico y almacenamiento de resultados complementarios.
- Sistema de Información de Salud (Health Information System): Generación de análisis estadísticos y epidemiológicos sobre la población registrada.

Figura 4. Ecosistema GNU Health



Fuente: Tomado de la ref. (43)

2.4.1 Arquitectura Técnica y Conceptos Fundamentales

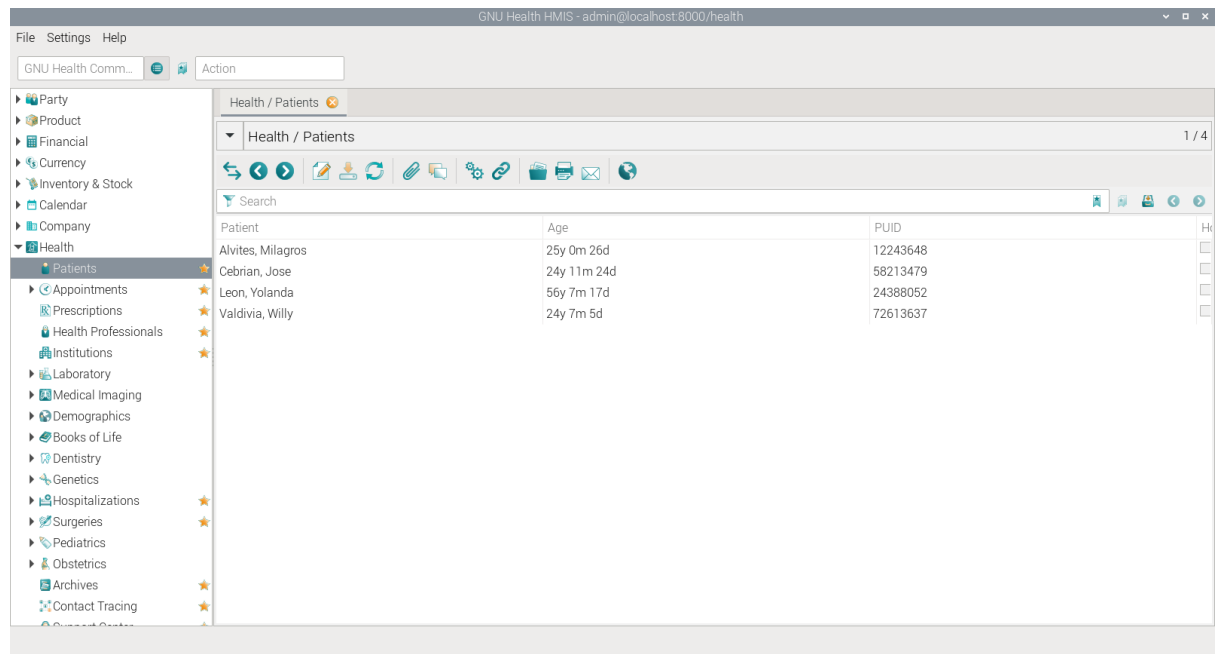
GNU Health HMIS destaca por su arquitectura modular, lo que permite un despliegue escalable según las necesidades del centro de salud (43). El sistema se basa en el framework de gestión empresarial Tryton, lo que introduce conceptos técnicos esenciales para el desarrollo de la API de integración (44):

- **Módulo (Module):** Unidad funcional que agrupa características específicas. En esta investigación se emplea el módulo health, el cual proporciona la lógica necesaria para representar datos fisiológicos y gestionar la relación entre pacientes y médicos.
- **Entidad (Party):** Representación de cualquier actor dentro del sistema (pacientes, profesionales, instituciones). Posee atributos específicos y niveles de acceso diferenciados según su rol.
- **Modelo (Model):** Clase de alto nivel en el framework Tryton que actúa como una capa de abstracción sobre la base de datos. Por ejemplo, el modelo `gnuhealth_patient` define la estructura y campos de la información de los pacientes.

- Tabla y Registro (Table & Record): La tabla constituye la representación física del modelo en la base de datos PostgreSQL, mientras que el registro corresponde a una entrada individual de datos (una fila).

A continuación, se presenta una vista del sistema GNU Health HMIS, específicamente la sección de pacientes, a través de la interfaz de usuario GNU Health Client:

Figura 5. GNU HMIS a través de la interfaz de usuario GNU Health Client.



Fuente: Elaboración propia.

2.4.2 Estrategia de Despliegue en Hardware Embebido

Para la implementación del prototipo, se seleccionó el sistema operativo Debian 12 sobre una Raspberry Pi 4. Si bien GNU Health ofrece imágenes oficiales para sistemas embebidos basadas en OpenSUSE (45), se optó por una instalación tipo "Vanilla" (instalación manual de componentes core) (46). Esta decisión técnica se justifica en la necesidad de contar con un entorno de desarrollo actualizado. Las versiones embebidas preconfiguradas suelen emplear distribuciones de Linux con librerías heredadas (legacy), lo cual genera conflictos de compatibilidad con frameworks modernos de desarrollo de APIs, como Flask. La instalación "Vanilla" sobre Debian permite el control total de las dependencias y la estabilidad necesaria para la ejecución simultánea de la HCE y los servicios de interoperabilidad (46).

2.5 Frameworks de Desarrollo de la API

Para la construcción de la lógica del servidor y la interfaz de usuario, se seleccionaron herramientas basadas en el lenguaje de programación Python, lo que permitió una integración fluida entre los diferentes componentes del sistema y el manejo eficiente de estructuras de datos complejas.

2.5.1 Framework del Backend: Flask

Para la arquitectura del servidor y el desarrollo de las API, se empleó Flask. Este se define como un micro-framework de desarrollo web cuya principal característica es su arquitectura minimalista y modular (47,48). A diferencia de otros frameworks más robustos, Flask no impone una estructura rígida de archivos o dependencias (47), lo que otorga al desarrollador la libertad de implementar únicamente las librerías necesarias para el proyecto.

Esta flexibilidad es fundamental para el despliegue en sistemas de recursos limitados, como la Raspberry Pi 4, ya que permite mantener un bajo consumo de memoria RAM dada su naturaleza ligera (47). Asimismo, facilita el enrutamiento de peticiones RESTful y la serialización de datos en formato JSON (47). Su amplia documentación y el soporte de su comunidad (48), facilitan la escalabilidad del prototipo, permitiendo transitar desde funciones básicas de recepción de datos hacia operaciones de integración clínica sin comprometer la estabilidad del sistema.

2.5.2 Framework del Frontend: Streamlit

La capa de presentación y visualización de datos fue desarrollada mediante Streamlit, un framework de código abierto diseñado específicamente para la creación de aplicaciones web interactivas orientadas a datos (49,50).

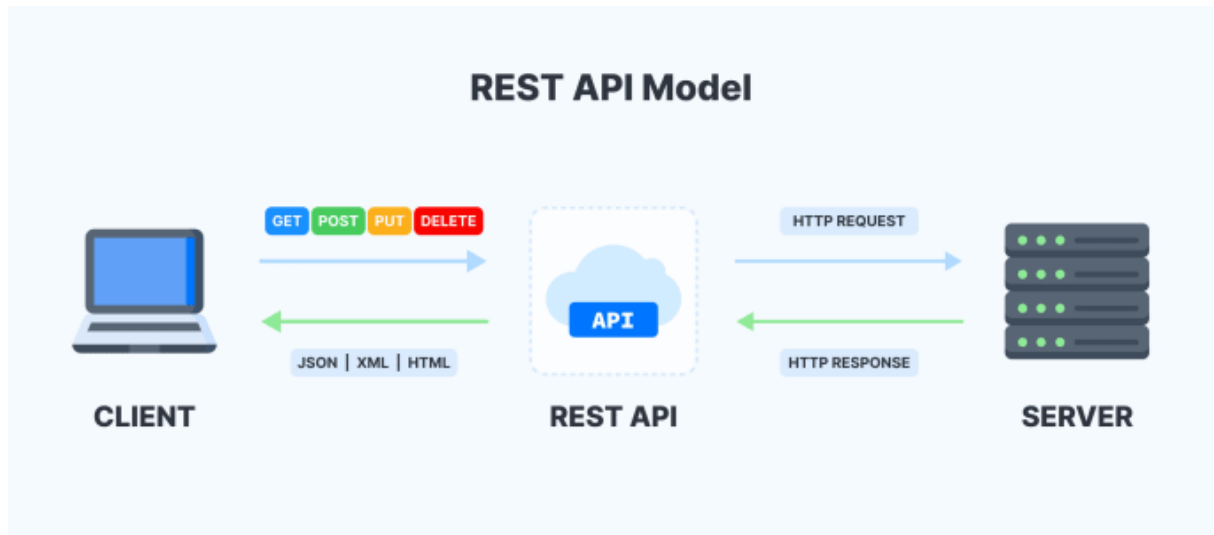
Streamlit permite transformar scripts de Python en interfaces gráficas dinámicas de manera eficiente, eliminando la necesidad de conocimientos profundos en tecnologías tradicionales de desarrollo web como HTML, CSS o JavaScript. Se seleccionó esta herramienta debido a su capacidad para gestionar flujos de datos en tiempo real y su compatibilidad con librerías de análisis de datos de Python (51). En el contexto de esta tesis, Streamlit facilita la supervisión de los datos transmitidos hacia la HCE y permite enviar parámetros de configuración hacia el prototipo biomédico mediante una interfaz intuitiva.

2.6 Arquitectura de Servicios Web (API REST)

Una API o interfaz de programación de aplicaciones es una herramienta que permite comunicar aplicaciones de software mediante protocolos, con el objetivo de compartir datos, funcionalidades o características (52). Cuando una API se diseña bajo las restricciones del estilo arquitectónico de Transferencia de Estado Representacional (REST), se denomina API RESTful (53). Dentro de esta arquitectura, destacan los siguientes elementos fundamentales: (54)

- ❖ **Recurso:** Es cualquier objeto o entidad que puede ser manipulada por la API. Representa la unidad principal de información que será consultada, modificada o eliminada.
- ❖ **URI:** Es una cadena de caracteres que identifica de manera única a un recurso dentro del sistema (por ejemplo: <http://apiRESTful.com/users/7258>). La URI direcciona directamente al recurso que se desea operar.
- ❖ **Métodos:** Todas las acciones que se ejecutan sobre los recursos deben estar asociadas a un método HTTP, entre ellos: GET, POST, PATCH y DELETE, cada uno con un propósito específico dentro del ciclo de interacción.
- ❖ **Endpoint:** Corresponde al punto de acceso mediante el cual los actores externos se comunican con la API. Un endpoint combina un método HTTP con una URI específica para ejecutar una operación determinada (ejemplo: GET <http://apiRESTful.com/users/7258>).
- ❖ **Headers:** Transportan información adicional necesaria para el procesamiento de las solicitudes, tales como formatos de contenido, credenciales de autenticación o parámetros de control.
- ❖ **Solicitud y Respuesta (Request and Response):** Representan la interacción entre los consumidores de la API y el servidor. La request es la petición que el cliente envía indicando el recurso y la operación deseada, mientras que la response es la respuesta generada por la API, que incluye el resultado de la operación y los datos correspondientes.

Figura 6. Arquitectura API RESTful



Fuente: Tomado de la ref. (54)

Además de los elementos que componen su arquitectura, las API RESTful presentan características que explican su amplia adopción en sistemas distribuidos y, particularmente, en entornos donde la interoperabilidad es un requisito esencial. Entre las principales características de una API RESTful es que todas siguen una interfaz uniforme; es decir, todas las solicitudes dirigidas a un endpoint deben seguir un formato estándar, independientemente del tipo de cliente (dispositivos IoT, aplicaciones web o móviles) que realice la petición mediante el protocolo HTTP (53). Por otro lado, se permite la independencia operativa; es decir, la evolución o escalabilidad del servidor no afecta al cliente y viceversa. La interacción sólo se produce mediante peticiones puntuales, lo que mejora la eficiencia del sistema (53). Por último, REST permite el uso de diversos formatos de serialización, principalmente JSON y XML (53), lo que resulta ideal para el intercambio de datos en sistemas heterogéneos como los entornos de salud.

Para la manipulación de los recursos dentro del servidor, la arquitectura REST emplea métodos estandarizados del protocolo HTTP, siendo los más relevantes para esta investigación: (55)

- GET: Método orientado exclusivamente a la obtención de información. El servidor devuelve la representación del recurso solicitado sin alterar su estado. El servidor devuelve un código de estado de confirmación en caso de éxito.

- POST: Utilizado para la creación de nuevos recursos en el servidor. El emisor envía datos y recibe una confirmación sobre el éxito de la operación. Además, este método puede utilizarse en otras operaciones.
- PATCH: Empleado para la actualización parcial de un recurso existente. A diferencia de otros métodos, solo modifica los campos específicos que se envían en la petición.
- DELETE: Destinado a la eliminación definitiva de un recurso del servidor.

En el contexto de la presente tesis, la arquitectura REST actúa como puente. Una vez que los datos biomédicos son estandarizados bajo el estándar HL7 FHIR y serializados en formato JSON, la API desarrollada permite su transferencia entre el intermediario de mensajería (Broker MQTT) y las bases de datos de la HCE. Esta estructura permite que el ecosistema sea interoperable y que sus componentes puedan escalar de forma independiente sin comprometer la integridad del flujo de información clínica.

2.7 Microcontrolador ESP32

El microcontrolador ESP32 es un sistema en chip (SoC) de alto rendimiento, ampliamente utilizado en el ecosistema del Internet de las Cosas (IoT) y en el desarrollo de prototipos rápidos. A nivel de conectividad, destaca por integrar de forma nativa soporte para redes Wi-Fi (802.11 b/g/n), Bluetooth tradicional y Bluetooth Low Energy (BLE) (56).

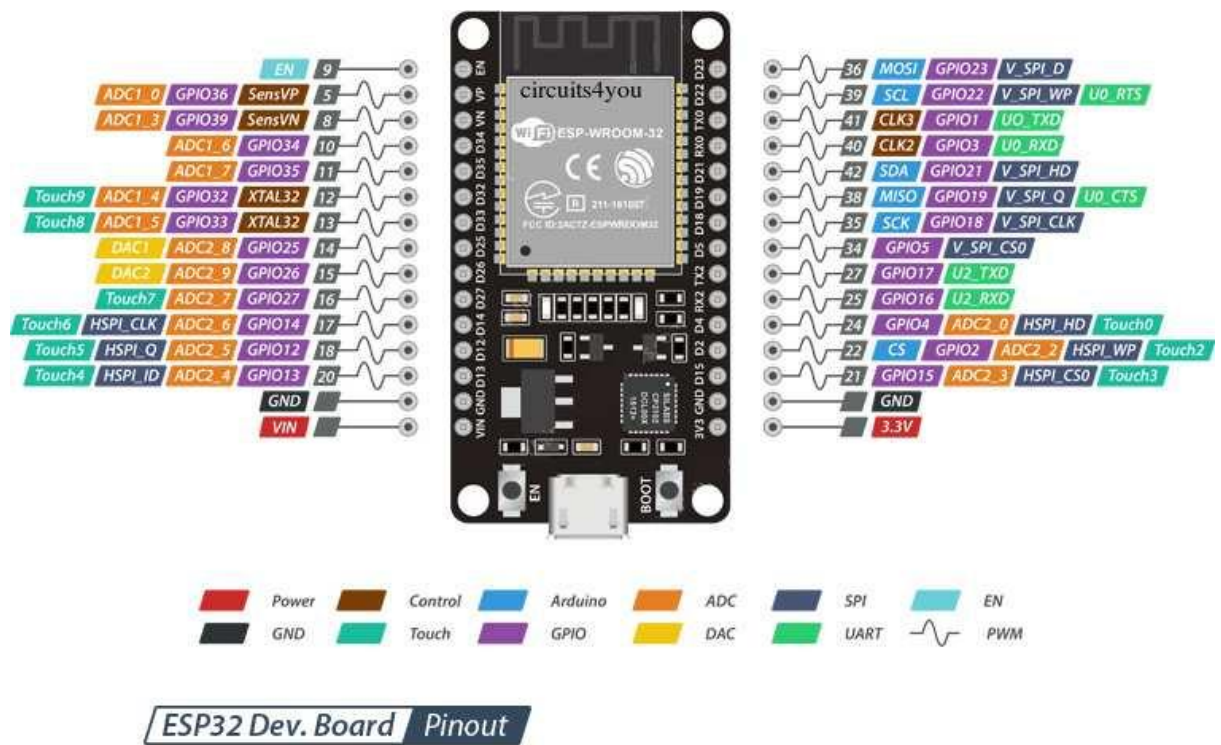
Esta versatilidad lo convierte en una opción idónea para arquitecturas que requieren comunicación inalámbrica continua, eliminando la necesidad de incorporar módulos externos. En cuanto a su capacidad computacional, posee un procesador Xtensa de doble núcleo de 32 bits que opera a una frecuencia de hasta 240 MHz, respaldado por una memoria SRAM de 520 KB, además de múltiples interfaces periféricas (I2C, SPI, UART, ADC) que facilitan la integración de sensores y módulos externos (56).

En el contexto de la presente investigación, el ESP32 se implementará como un nodo IoT simulador de dispositivos médicos. Aprovechando su capacidad de procesamiento, memoria y conectividad, el microcontrolador ejecutará tres tareas críticas: 1) la generación algorítmica de datos fisiológicos; 2) la estructuración nativa de estos datos en formato JSON bajo el estándar internacional HL7 FHIR; y 3) la transmisión de las cargas útiles (payloads) hacia el middleware mediante el protocolo MQTT.

Para validar la flexibilidad del ecosistema, el firmware del ESP32 ha sido diseñado para emular seis tipos de dispositivos clínicos y sus respectivas variables: termómetro

(temperatura), tensiómetro (presión arterial), monitor de constantes vitales (frecuencia cardíaca y frecuencia respiratoria), pulsioxímetro (saturación de oxígeno) y glucómetro (niveles de glucosa). Es importante precisar que, dado que el objetivo del estudio es evaluar la interoperabilidad y el flujo de datos del sistema, las mediciones transmitidas son valores sintéticos (artificiales) generados computacionalmente por el microcontrolador, prescindiendo de la toma de muestras reales en pacientes. A continuación, se presenta el diagrama de pines de la placa de desarrollo ESP32 DevKit V1 en su versión de 30 pines, la cual se usó en el desarrollo de la presente tesis.

Figura 7. Diagrama de pines del módulo ESP32.



Fuente: Tomado de la ref. (57)

2.8 Metodología Input Space Partitioning (ISP)

La metodología Input Space Partitioning (ISP) constituye una técnica de diseño de casos de prueba cuyo principio consiste en dividir el espacio de entrada del sistema en un conjunto de bloques o particiones (58). Dentro de cada bloque, los valores se consideran equivalentes desde la perspectiva de las pruebas, por lo que la selección de uno o más valores representativos permite evaluar el comportamiento funcional del sistema.

De acuerdo con Ammann y Offutt (58), una partición debe cumplir dos propiedades fundamentales: completitud y disyunción. La propiedad de completitud establece que la

unión de todos los bloques debe cubrir la totalidad del dominio de entrada, mientras que la propiedad de disyunción establece que los bloques no deben compartir valores entre sí, de modo que cada valor pertenezca únicamente a un bloque. Estas propiedades permiten construir un modelo del dominio de entrada que represente de manera adecuada las posibles condiciones de prueba del sistema.

Una vez definidas las particiones, la metodología contempla diferentes criterios para combinar los bloques y generar los casos de prueba. Entre ellos se encuentra el criterio All Combinations Coverage (ACoC) (58), el cual establece que deben evaluarse todas las combinaciones posibles entre los bloques de las características definidas. Si bien este criterio incrementa el número de casos de prueba conforme aumenta el número de particiones, proporciona una cobertura completa de las combinaciones consideradas durante el diseño experimental.

En la presente investigación, la metodología ISP fue empleada para estructurar los escenarios de validación funcional del sistema de interoperabilidad. Para ello, se definieron las características de entrada relevantes y sus respectivas particiones, aplicando posteriormente el criterio All Combinations Coverage (ACoC) para generar las combinaciones utilizadas durante las pruebas experimentales, cuyo desarrollo se describe en la Sección 4.6.4.2.

III. Pregunta de investigación y Objetivos

En este capítulo se presenta la pregunta de investigación que orienta el desarrollo del proyecto, así como el objetivo general y los objetivos específicos planteados para abordar la problemática identificada.

3.1 Pregunta de investigación.

¿De qué manera es posible la integración entre prototipos biomédicos estudiantiles e historias clínicas electrónicas de acceso libre como GNU Health usando estándares de interoperabilidad HL7 FHIR y una API REST?

3.2 Objetivo general

Desarrollar una API que permita la interoperabilidad de prototipos biomédicos desarrollados por estudiantes con HCE de acceso libre.

3.3 Objetivos específicos

- ❖ Desarrollar un nodo simulador de prototipos biomédicos capaz de generar, estructurar en formato HL7 FHIR y transmitir datos fisiológicos vía MQTT.
- ❖ Implementar una API REST que reciba los datos clínicos vía MQTT y automatice su registro en la historia clínica electrónica GNU Health.
- ❖ Diseñar una interfaz web para la visualización de la información recibida en tiempo real y el control remoto del nodo simulador.
- ❖ Validar la interoperabilidad del sistema integrado mediante pruebas funcionales que verifiquen la consistencia, trazabilidad y correcto almacenamiento de los datos, desde su generación hasta su almacenamiento final.

IV. Materiales y métodos

En este capítulo se definirán la metodología usada, los materiales y las etapas de implementación del prototipo desarrollado en la presente tesis.

4.1 Enfoque Metodológico

Para lograr la interoperabilidad entre los prototipos biomédicos y la HCE de acceso libre, la investigación se estructuró bajo un enfoque de desarrollo modular e integración incremental. Este proceso se llevó a cabo a través de cuatro etapas secuenciales, donde cada una valida un rol en el ciclo de vida del dato clínico:

- **Etapla 1: Desarrollo del nodo simulador del prototipo biomédico.** Esta etapa inicial consistió en la programación del firmware en C + + para el microcontrolador. Su función principal abarca la generación de datos biomédicos sintéticos (tanto aleatorios como personalizados), su estructuración bajo el estándar de interoperabilidad HL7 FHIR, y su posterior transmisión inalámbrica hacia el servidor central (middleware) utilizando el protocolo de mensajería MQTT.
- **Etapla 2: Implementación del middleware de integración (API REST).** En esta fase se desarrolló una Interfaz de Programación de Aplicaciones (API) en Python, desplegada sobre una Raspberry Pi 4. Este middleware actúa como el componente central de la arquitectura: recibe los recursos FHIR provenientes del microcontrolador, gestiona su almacenamiento en una base de datos operativa y ejecuta la inserción de los registros en la base de datos asociada al servidor HMIS de GNU Health. Adicionalmente, esta API procesa y muestra la información para que pueda ser consumida por la interfaz de usuario.
- **Etapla 3: Diseño y desarrollo de la interfaz web de monitoreo y gestión.** Esta etapa se enfocó en la creación de una aplicación web de cara al usuario final mediante el framework Streamlit. Esta herramienta cumple un doble propósito. En primer lugar, permite la visualización en tiempo real de la información clínica validada por la API. Y, en segundo lugar, opera como un centro de control remoto, enviando instrucciones de configuración y comandos de vuelta al microcontrolador (desarrollado en la Etapa 1) a través de la red MQTT.
- **Etapla 4: Integración sistémica y pruebas de consistencia.** La fase final consistió en la consolidación de los desarrollos previos en un sistema único e

integrado. Aquí se ejecutaron las pruebas de consistencia de datos de extremo a extremo (desde la generación en el nodo simulador hasta su almacenamiento efectivo en GNU Health), permitiendo la disponibilidad del sistema, la comunicación bidireccional y el correcto enrutamiento de la información en el ecosistema digital.

4.2 Materiales y Equipos

En esta sección se describen los recursos tecnológicos empleados para el diseño, desarrollo e implementación del sistema. La selección de estos componentes permite la viabilidad técnica del sistema de interoperabilidad propuesto.

4.2.1 Recursos de hardware

Los componentes físicos constituyen la infraestructura necesaria para la adquisición de señales biomédicas y el despliegue de los servicios web.

Tabla 1. Recursos hardware usados en el desarrollo de la presente tesis.

Componente	Especificación Técnica	Función
Microcontrolador	ESP32 (Dual Core 240MHz)	Encargado de la generación de datos, empaquetado en estándar HL7 FHIR y transmisión vía MQTT.
Computador Embebido	Raspberry Pi 4 Modelo B (4 GB RAM)	Alojamiento de la plataforma GNU Health, interfaz web en Streamlit, servicios API REST y motores de bases de datos.
Almacenamiento	Tarjeta Micro SD 32 GB (Clase 10)	Soporte de almacenamiento para el sistema operativo del computador embebido y almacenamiento de datos locales.

Periféricos	Teclado, Mouse y Monitor	Interfaz física para la configuración local de la Raspberry Pi 4.
Equipo de Soporte	Laptop con entorno de desarrollo	Estación de trabajo destinada a la programación del firmware y flasheo de imágenes.

Fuente: Elaboración propia.

4.2.2 Recursos de Software y Herramientas de Desarrollo

Se utilizaron diversas plataformas y herramientas para permitir que exista un flujo adecuado entre los componentes antes descritos. Para ello, con el objetivo de seguir estándares internacionales de interoperabilidad, se usaron las siguientes herramientas para validar que los datos enviados están en el formato HL7 FHIR:

- Simplifier.net: Repositorio para la búsqueda y referenciación de perfiles clínicos y recursos FHIR orientados a variables fisiológicas.
- Validator FHIR: Herramienta de validación para asegurar la integridad sintáctica y semántica de los objetos JSON generados por el microcontrolador.

Por otro lado, para la gestión de datos en la transmisión y el almacenamiento final de los mismos en el servidor receptor se usaron las siguientes herramientas:

- EMQX: Broker de mensajería MQTT de alto rendimiento para la comunicación entre el ESP32 y el servidor.
- DBeaver: Gestor universal de bases de datos para la administración de los motores de bases de datos PostgreSQL y MariaDB.
- MariaDB: Sistema de gestión de bases de datos relacionales para el almacenamiento de la interfaz web desarrollado en Streamlit.
- PostgreSQL (15.10): Motor de base de datos principal para el HMIS GNU Health.

4.2.3 Frameworks y Librerías de Programación

El desarrollo del software se dividió en tres capas: Firmware (C++), Backend (Python/Flask) y Frontend (Python/Streamlit).

4.2.3.1 Capa de Firmware (Arduino IDE / C++)

Se emplearon las siguientes librerías específicas para la gestión de red y la serialización de datos en el ESP32:

- WiFi.h y WiFiUdp.h: Gestión de conectividad inalámbrica.
- PubSubClient.h: Implementación del protocolo MQTT.
- ArduinoJson.h: Generación de recursos FHIR en formato JSON.
- NTPClient.h: Sincronización temporal para el estampado de tiempo (timestamp) de las mediciones.
- EEPROM.h: Almacenamiento no volátil de parámetros de configuración.

4.2.3.2 Capa de Backend y Servicios API (Python 3.11.2)

Para la lógica de las APIs, se utilizaron las siguientes librerías de integración:

- Flask: Micro-framework para la creación de servicios RESTful.
- Paho-MQTT: Cliente para la suscripción y publicación de mensajes en el Broker.
- Psycopg2: Adaptador para la interacción con la base de datos PostgreSQL de GNU Health.
- Mysql-connector-python: Driver para la comunicación con MariaDB.
- Werkzeug: Herramientas de seguridad para la gestión y cifrado de contraseñas (hashing).

4.2.3.3 Capa de Visualización y Frontend (Streamlit)

Para la programación de la interfaz virtual, que servirá de apoyo para la observación de los datos recibidos en el servidor, se utilizarán las siguientes librerías:

- Streamlit: Framework principal para el despliegue del dashboard interactivo.
- Pandas: Librería para el análisis y estructuración de series de datos biomédicos.
- Plotly Express: Motor de renderizado para la generación de gráficas dinámicas y visualización de tendencias en tiempo real.

4.2.4 Configuración del Ecosistema GNU Health (HMIS v4.4)

Se implementó la versión 4.4 de GNU Health como núcleo del sistema de información hospitalaria, seleccionada por su estabilidad y disponibilidad durante el desarrollo de la presente tesis. El despliegue en la Raspberry Pi 4 se realizó siguiendo una instalación tipo Vanilla, bajo los siguientes parámetros técnicos:

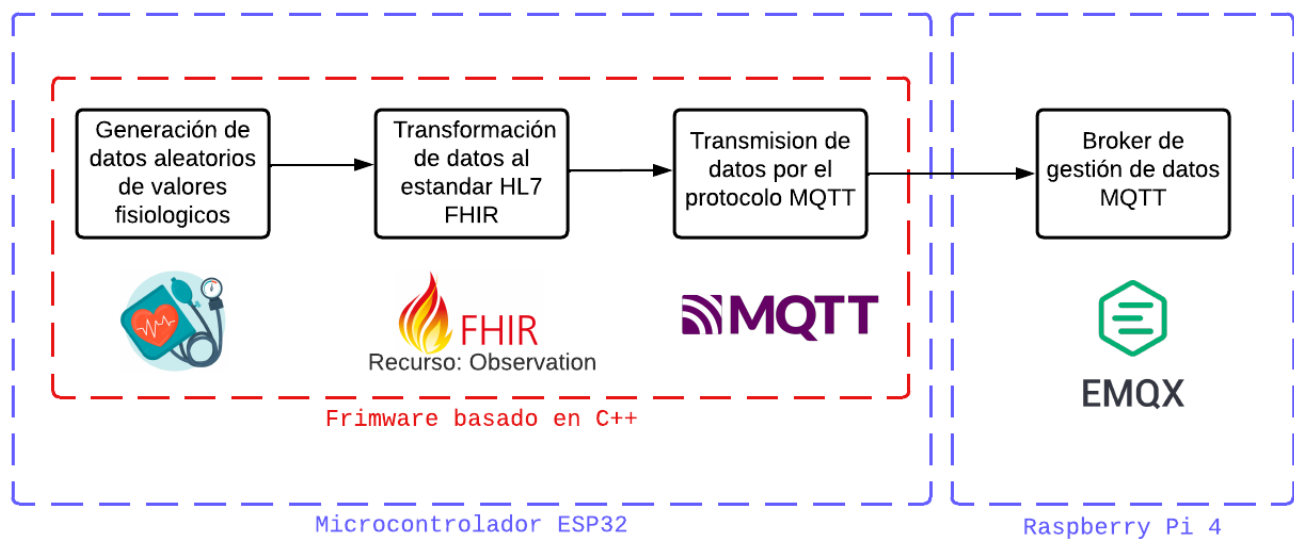
- Sistema Operativo: Debian 12.9 (Bookworm).
- Lenguaje de Programación: Python 3.11.2

- Framework de Gestión: Tryton v6.0 (Serie vinculada a la versión 4.4 de GNU Health).
- Herramientas de Entorno: Bash shell y gestor de paquetes PIP para la resolución de dependencias de Python.

4.3. Implementación del Firmware del Nodo Simulador (Etapa 1)

Esta primera etapa de desarrollo se centra en la construcción de un nodo simulador de prototipo biomédico basado en el microcontrolador ESP32. Su propósito es emular el comportamiento de un dispositivo médico real para validar el flujo de información y permitir su compatibilidad con los estándares internacionales de salud. El diseño funcional de este nodo se ilustra en la Figura 8.

Figura 8. Diseño funcional del nodo simulador del prototipo biomédico.



Fuente: Elaboración propia.

El proceso de desarrollo del firmware se estructuró en tres subfases incrementales y complementarias:

4.3.1 Generación de datos fisiológicos sintéticos

Se desarrolló un algoritmo en C++ diseñado para la generación de variables fisiológicas puntuales. Para efectos de esta investigación, se definieron seis parámetros representativos: temperatura, presión arterial, saturación de oxígeno (SpO2), frecuencia respiratoria, frecuencia cardíaca y glucosa.

La selección de estas variables fisiológicas respondió a criterios tanto clínicos como tecnológicos. En primer lugar, la temperatura corporal, la presión arterial, la saturación

de oxígeno, la frecuencia cardíaca y la frecuencia respiratoria fueron seleccionadas debido a que la literatura las reconoce como los cinco signos vitales tradicionales de mayor importancia para la evaluación del estado de salud del paciente (59). Asimismo, estos parámetros son considerados esenciales para la monitorización clínica y constituyen la base de diversos sistemas de detección temprana del deterioro fisiológico, como los Early Warning Scores (EWS), utilizados para identificar oportunamente pacientes con riesgo de deterioro clínico (60).

Por otro lado, se decidió incorporar la glucosa debido a que constituye un parámetro bioquímico de gran relevancia para el seguimiento de personas con diabetes mellitus y representa una de las variables con mayor desarrollo tecnológico en dispositivos biomédicos de monitorización continua (59). Asimismo, su inclusión resulta pertinente en el contexto nacional, considerando que el Instituto Nacional de Salud reporta que la diabetes presenta una prevalencia del 10,7 % en la población adulta peruana, equivalente aproximadamente a 2,6 millones de personas, y recomienda la evaluación preventiva de los niveles de glucosa al menos una vez al año (61).

Finalmente, además de su importancia clínica, la selección de estas variables permitió representar tanto observaciones de valor único (temperatura, frecuencia cardíaca, frecuencia respiratoria, SpO₂ y glucosa) como observaciones compuestas (presión arterial), validando diferentes representaciones del recurso *Observation* del estándar HL7 FHIR.

Para la validación funcional, se implementó una interfaz de comandos mediante el monitor serial del ESP32. El sistema asocia caracteres alfabéticos específicos a cada variable (por ejemplo, "T" para temperatura), permitiendo al desarrollador disparar la generación del dato y verificar su correcta representación en memoria antes de ser procesado.

4.3.2 Estandarización e Interoperabilidad Semántica (HL7 FHIR)

Con el fin de asegurar que la información sea interpretable por cualquier sistema de salud moderno, se integró el estándar HL7 FHIR. El proceso de normalización de los datos se apoyó en dos herramientas críticas:

- Simplifier.net: Utilizado para la investigación y selección de ejemplos de perfiles adecuados para el recurso *Observation* para cada variable fisiológica.
- FHIR Online Validator: Empleado para la verificación sintáctica de los objetos JSON generados, permitiendo el cumplimiento de las reglas del estándar.

En esta etapa, el ESP32 no solo procesa el valor fisiológico, sino que construye un recurso integral que incluye metadatos de trazabilidad: nombre del paciente, nombre del médico, nombre del dispositivo, número de identidad del paciente y del médico (DNI), identificación única del dispositivo (ID de 4 dígitos) y un estampado cronológico (timestamp) generado mediante una función de tiempo real, el cual constituye un requisito para la validez clínica del dato.

4.3.3 Comunicación y Transmisión mediante Protocolo MQTT

La transmisión de los recursos hacia el servidor central (Raspberry Pi 4) se gestiona a través del broker EMQX. Para facilitar el enrutamiento diferenciado de mensajes entre dispositivos y pacientes, se implementó una arquitectura de tópicos dinámica. Para ello, se configuró el código para que el usuario ingrese los metadatos a través del monitor serial, lo que permite al firmware construir las rutas de comunicación de la siguiente manera:

- ❖ Tópico de entrada:

nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in

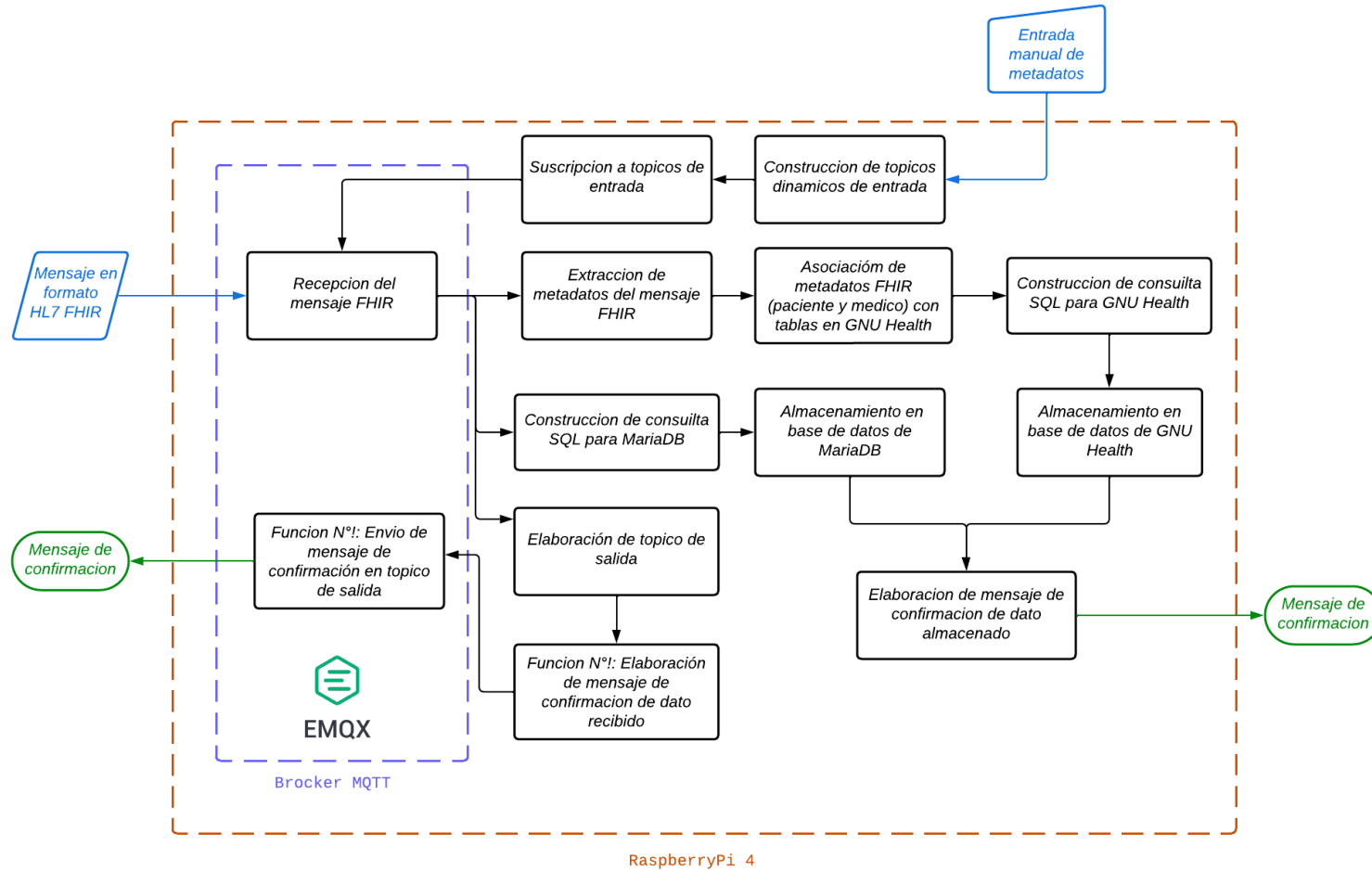
- ❖ Tópico de salida: *nombre_dispositivo/id_paciente/id_dispositivo/out*

La variable *tipo_de_dato* se asigna automáticamente (p. ej., *temperature*, *blood_pressure*), según haya sido la letra ingresada por el usuario en el monitor serial del ESP32. Este enfoque permite que cada mensaje sea unívoco, evitando la sobreescritura y facilitando el enrutamiento en el servidor central. Finalmente, se validó la bidireccionalidad del sistema: el ESP32 publica el recurso en el tópico de entrada y permanece en estado de escucha hasta recibir un mensaje de confirmación (ACK) en el tópico de salida, asegurando la integridad de la entrega de datos.

4.4. Implementación del Middleware de Integración (Etapa 2)

Esta segunda etapa de desarrollo se enfoca en la construcción de la capa intermedia o middleware (API REST) del sistema. Su diseño funcional está orientado a la recepción, procesamiento y gestión estructurada de los datos clínicos que transitan entre el broker MQTT y los repositorios finales de información (base de datos operativa y GNU Health). La arquitectura de esta capa de integración se detalla en la Figura 9.

Figura 9. Diseño funcional del middleware de integración.



Fuente: Elaboración propia

Para asegurar una implementación adecuada, el desarrollo de esta etapa se segmentó en fases progresivas, iniciando con la gestión de la mensajería entrante y el almacenamiento local de los datos.

4.4.1 Desarrollo del Módulo de Recepción y Almacenamiento de Datos

En esta etapa inicial, se desarrolló un servicio en Python encargado de la suscripción activa al broker EMQX, el procesamiento de los recursos HL7 FHIR y el posterior almacenamiento en una base de datos MariaDB.

4.4.1.1 Suscripción Dinámica y Recepción de Mensajería

Para permitir la sincronía con el nodo simulador de prototipo biomédico, el script solicita al inicio los parámetros de configuración que definen la jerarquía de los tópicos: Nombre del dispositivo, ID del paciente e ID del dispositivo. Basándose en estos metadatos, el sistema realiza una suscripción dinámica a seis tópicos específicos bajo la estructura:

nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in

Este enfoque permite que la API permanezca en estado de escucha asíncrona, capturando exclusivamente los flujos de información autorizados y asegurando que los datos de diferentes pacientes o dispositivos se mantengan aislados y segmentados correctamente.

4.4.1.2 Procesamiento de Recursos FHIR y Almacenamiento

Una vez recibido el mensaje en formato JSON, el sistema ejecuta una función de deserialización para extraer la información clínica y técnica contenida en el estándar HL7 FHIR. Los campos extraídos incluyen la identificación completa del paciente y el médico, la trazabilidad del dispositivo, el valor fisiológico medido (incluyendo la distinción entre presión sistólica y diastólica) y las unidades de medida.

Para el almacenamiento de esta información, se diseñó una estructura de tabla en MariaDB (tabla observations), en una base de datos denominada health_monitoring, que permite una trazabilidad total de la medición. Se incluyó un campo de tipo LONGTEXT para almacenar el recurso FHIR completo, permitiendo que, ante cualquier auditoría, se cuente con el mensaje original enviado por el microcontrolador.

Tabla 2. Estructura de la base de datos health_monitoring en MariaDB

Campo	Tipo de Dato	Descripción
patient_name	VARCHAR(255)	Nombre del paciente
patient_id	VARCHAR(20)	ID del paciente (DNI u otro

		identificador)
device_name	VARCHAR(255)	Nombre del dispositivo emisor
device_id	VARCHAR(10)	Código de 4 dígitos asignado al hardware.
doctor_name	VARCHAR(255)	Nombre del médico que realizó la medición
doctor_id	VARCHAR(20)	ID del médico (DNI u otro identificador)
physiological_type	VARCHAR(50)	Variable registrada (ej. <i>temperature</i>).
value	FLOAT	Valor medido (Presión sistólica en caso de PA).
value2	FLOAT	Presión diastólica (solo si aplica).
unit	VARCHAR(20)	Unidad de medida (ej. °C, mmHg, bpm).
full_message	LONGTEXT	Recurso JSON HL7 FHIR original completo.
timestamp	TIMESTAMP	Marca temporal de registro en el servidor.

Fuente: Elaboración propia.

Esta estructura permite que cada medición se registre de forma unívoca, facilitando el análisis temporal de los datos y la trazabilidad de los datos recibidos desde el nodo simulador.

4.4.2 Instalación y puesta en marcha del HMIS de GNU Health

Una vez establecida la capa de almacenamiento intermedio, se procedió con la instalación y puesta en marcha del servidor de GNU Health HMIS v4.4. El despliegue se realizó siguiendo la documentación técnica oficial (instalación tipo Vanilla) (46), asegurando la compatibilidad de los módulos de salud con el entorno de la Raspberry Pi 4.

Luego de la inicialización del servidor, se utilizó el programa GNU Health Client para la gestión administrativa del sistema. Con el fin de validar la interoperabilidad del ecosistema, se procedió con el aprovisionamiento de entidades clínicas, registrando en la base de datos central a 4 pacientes y 2 médicos facultativos a través de GNU Health

Client. En la Figura 10, se observa la interfaz del programa con el listado de las entidades registradas (médicos, pacientes e instituciones) que conforman el entorno de pruebas.

Por otro lado, dada la funcionalidad nativa de GNU Health para la gestión de accesos, se crearon cuentas de usuario específicas para delimitar las acciones dentro de la plataforma. Esta configuración permite que el sistema opere bajo un esquema de permisos predefinidos por el software:

- **Perfiles Médicos:** Se configuraron dos usuarios con rol profesional (medical_criss y medical_richard), los cuales están facultados por el sistema para modificar historias clínicas, crear evaluaciones y gestionar datos de salud.
- **Perfil Administrador:** El usuario admin se reservó para tareas de gestión de infraestructura y visualización de entidades, manteniendo la restricción nativa de no alterar registros clínicos directamente.

En la Tabla N°3, se detallan los usuarios configurados para las pruebas de integración:

Tabla 3. Usuarios configurados en el programa GNU Health Client

Usuario	Contraseña	Perfil asignado
admin	5729	Administrador del Sistema
medical_criss	5729willy	Profesional de la Salud (Médico)
medical_richard	5729willy	Profesional de la Salud (Médico)

Fuente: Elaboración propia.

Por otro lado, se configuraron diversos pacientes y profesionales de la salud en el sistema GNU Health mediante la interfaz de usuario GNU Health Client. En la Figura 10, se presentan las entidades registradas, junto con el rol que desempeñan dentro del HMIS de GNU Health.

Figura 10. Captura de pantalla del listado de entidades en GNU Health Client.

Federation ID	Internal User	Name	Family names	Person	Patient	Health Prof	Institution	Deceased	Receivable To	Payable Today
XXXOIG235H	LABIB	Milagros	Alvites	☒	☒	☐	☐	☐	0,00	0,00
XXXGPP068Z	Medical Criss	Cristopher	Casani	☒	☐	☒	☐	☐	0,00	0,00
XXXUWK024I	Jose	Cebrian		☒	☒	☐	☐	☐	0,00	0,00
XXXYQK737II	Yolanda	Leon		☒	☒	☐	☐	☐	0,00	0,00
XXXSKQ745L	Medical Rich	Richard	Lonconi	☒	☐	☒	☐	☐	0,00	0,00
XXXCDP963C	Willy	Valdivia		☒	☒	☐	☐	☐	0,00	0,00

Fuente: Elaboración propia.

Esta etapa de configuración fue esencial para habilitar el ecosistema de salud, permitiendo que la API tenga un destino estructurado donde alojar la información capturada, respetando la lógica interna del HMIS de GNU Health.

4.4.3 Identificación de estructura de la base de datos de GNU Health

Al haber implementado el GNU Health HMIS, se realizó un proceso de exploración técnica de su base de datos con el propósito de identificar la estructura relacional necesaria para integrar el middleware desarrollado con la HCE. La estrategia de integración adoptada consistió en realizar inserciones controladas sobre la base de datos PostgreSQL de GNU Health, debido a que el objetivo de la investigación fue validar el registro efectivo de la información fisiológica dentro de una HCE funcional. Para ello, fue necesario identificar las tablas, relaciones y restricciones involucradas en el almacenamiento de evaluaciones clínicas, permitiendo que las operaciones implementadas preservaran la consistencia de los datos.

Para este análisis se empleó el gestor DBeaver, estableciendo una conexión con el motor PostgreSQL mediante el usuario de sistema “gnuhealth”, configurado durante la instalación para permitir la administración local del esquema de datos.

Mediante la exploración de los esquemas, dentro de la base de datos denominada **health**, se determinó que la información clínica de carácter fisiológico se centraliza en la tabla

gnuhealth_patient_evaluation. Esta tabla es la encargada de registrar los signos vitales y las observaciones médicas de cada consulta. En la Figura 11, se detalla la estructura técnica de esta entidad, destacando el campo diastolic (destinado a almacenar la presión arterial diastólica).

Figura 11. Captura de pantalla de la estructura de la tabla ***gnuhealth_patient_evaluation*** en DBeaver.

Column Name	#	Tipo de datos	Identidad	Collation	No Nulo	Por defecto
123 id	1	serial4			[v]	nextval(format('%f::text, 'gnuhealth_patient_evaluation_id_seq');)
123 abdominal_circ	2	float8			[f]	
123 abstraction	3	bool			[f]	false
123 appointment	4	int4			[f]	
123 lowni	5	float8			[f]	
123 lpgm	6	int4			[f]	
123 lpgm	7	bool			[f]	false
A4 chief_complaint	8	varchar		default	[f]	
123 cholesterol_total	9	int4		default	[f]	
A4 code	10	varchar		default	[f]	
123 create_date	11	timestamp(6)			[f]	
123 create_uid	12	int4			[f]	
123 dehydration	13	bool			[f]	false
123 derived_from	14	int4			[f]	
123 derived_to	15	int4			[f]	
123 diagnosis	16	int4			[f]	
123 diastolic	17	int4			[f]	

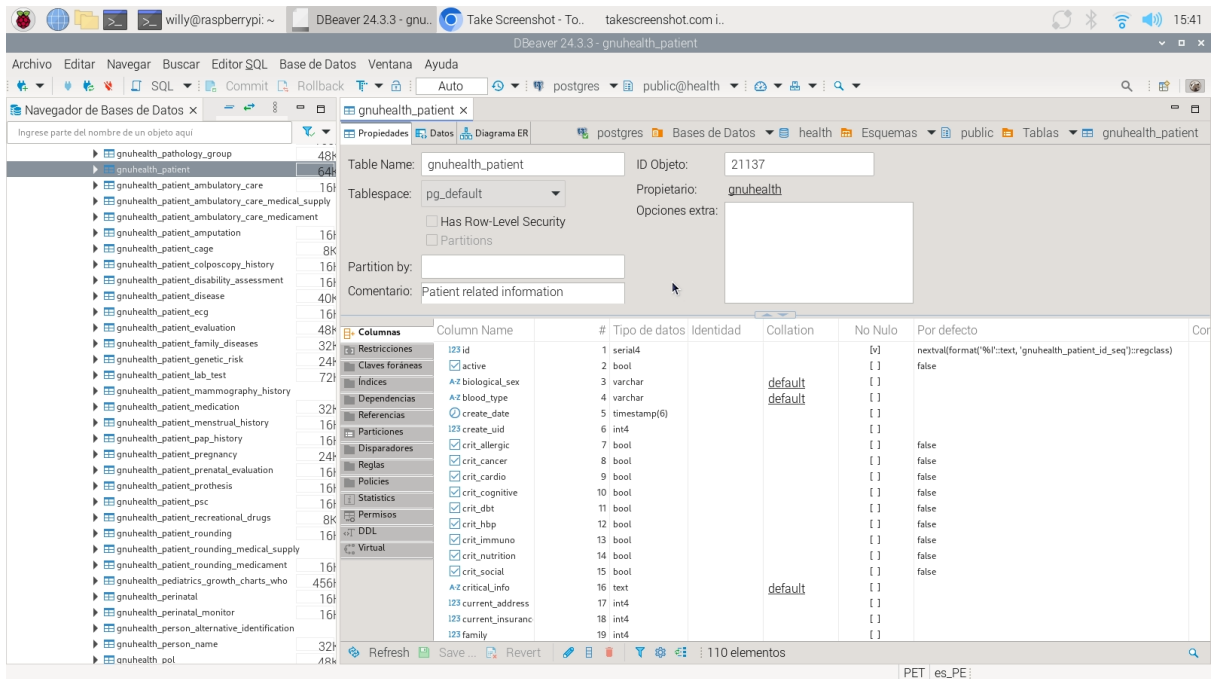
Fuente: Elaboración propia.

Durante el análisis, se identificó que la tabla de evaluaciones no es aislada, sino que depende de un modelo relacional estricto para mantener la integridad de la información. Se hallaron dos dependencias críticas (llaves foráneas) que vinculan cada registro clínico con los actores del sistema:

- **Vínculo con el paciente:** El atributo patient de la tabla de evaluación debe corresponder a un id existente en la tabla ***gnuhealth_patient***.
- **Vínculo con el profesional:** El atributo healthprof debe estar asociado a un id existente en la tabla ***gnuhealth_healthprofessional***.

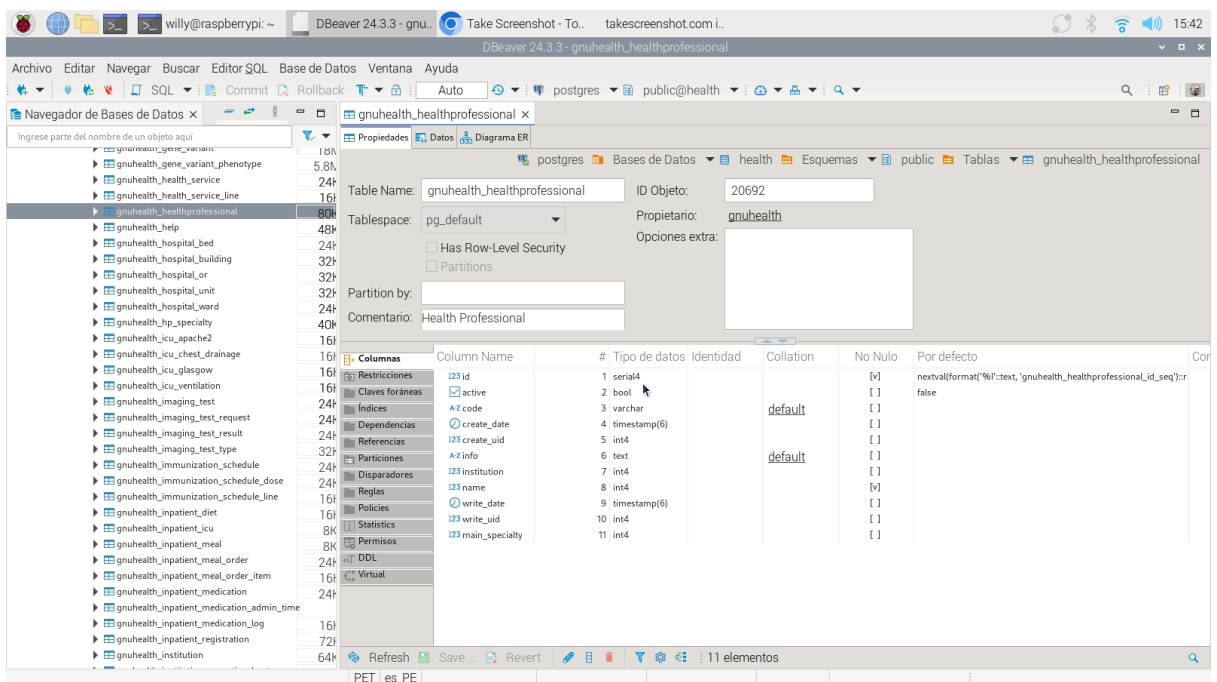
Este esquema asegura que no existan "evaluaciones huérfanas", permitiendo que cada dato biomédico esté estrictamente vinculado a un paciente registrado y a un médico facultativo responsable. En las Figuras 12 y 13, se presentan las estructuras de estas tablas maestras.

Figura 12. Captura de pantalla de la estructura de *gnuhealth_patient* en DBeaver.



Fuente: Elaboración propia.

Figura 13. Captura de pantalla de la estructura de *gnuhealth_healthprofessional* en DBeaver.



Fuente: Elaboración propia.

Esta exploración de la base de datos constituyó un paso importante para el desarrollo de la lógica de integración implementada. El análisis de las tablas, relaciones y restricciones

permitió diseñar inserciones controladas que respetaran la integridad referencial y las reglas de consistencia del modelo de datos de GNU Health durante el registro de las evaluaciones clínicas

4.4.4 Desarrollo de código para el almacenamiento de datos en GNU Health
Con el objetivo de permitir una integración en la tabla *gnuhealth_patient_evaluation*, se diseñó una estrategia de validación en dos etapas: una fase de aprovisionamiento manual y una fase de integración automatizada con el nodo simulador de prototipo biomédico.

4.4.4.1 Pruebas de Concepto: Registro Manual y Mapeo Relacional

Inicialmente, se desarrolló un script en Python destinado a la inserción manual de datos. El propósito de esta etapa fue aislar la lógica de almacenamiento de la capa de transporte (MQTT), permitiendo validar que los datos fueran aceptados por PostgreSQL sin errores de esquema. El flujo de esta funcionalidad contempla la captura de variables críticas: temperatura, frecuencia cardíaca, presión arterial (sistólica/diastólica) y los documentos de identidad (DNI) de los actores involucrados (médico y paciente).

4.4.4.1.1 Gestión de Restricciones y Estampado Temporal

Durante el desarrollo, se identificó que el esquema de GNU Health impone una restricción de "no nulidad" en la columna *discharge_reason* (motivo del alta). Para cumplir con la integridad del esquema sin alterar el flujo clínico, se estableció el valor "home" (domicilio) por defecto. No obstante, se mantuvo la flexibilidad para que el personal médico pueda modificar este campo posteriormente a través del cliente oficial de GNU Health. Asimismo, se integró una función para capturar el tiempo exacto de la medición en formato UTC, asignándole a la columna *evaluation_start*.

4.4.4.1.2 Lógica de Mapeo de Identidades (DNI a ID Interno)

Debido a que GNU Health utiliza identificadores internos numéricos y no el DNI directamente en sus tablas de evaluación, se implementó una lógica de búsqueda y vinculación estructurada en tres niveles:

- 1. Verificación de Identidad Global (tabla *party_party*):** El script consulta la tabla maestra *party_party*, buscando los DNIs ingresados en la columna *ref*. Si los registros existen, se extraen los IDs globales de dicha tabla. Estos IDs identifican a la "persona" de forma general en el sistema. En caso de no hallar alguno de los dos documentos, el programa emite un mensaje de error y detiene la ejecución.

2. **Validación de Roles:** Una vez obtenidos los IDs globales, el sistema debe localizar su equivalencia en las tablas especializadas. Aquí es donde ocurre la vinculación técnica: el ID global de la persona se busca en la columna denominada *name* de las tablas *gnuhealth_healthprofessional* (para médicos) y *gnuhealth_patient* (para pacientes). En la arquitectura de GNU Health, la columna *name* actúa como el enlace (llave foránea) hacia la identidad global del individuo.
3. **Asignación de Llaves Foráneas:** Al validar la coincidencia en la columna *name*, el script extrae los IDs principales (las llaves primarias de la primera columna) de ambas tablas especializadas. Estos valores se asignan a las variables *health_prof* y *patient_id*. Solo con estos identificadores específicos el sistema permite realizar la inserción en la tabla de evaluaciones, permitiendo que el registro sea coherente con el modelo relacional y las reglas de integridad de PostgreSQL.

4.4.4.1.3 Ejecución del almacenamiento de datos (SQL)

Al consolidar todos los valores validados, se ejecuta la sentencia de inserción mediante la librería *psycopg2*, como se muestra en la Figura 14.

Figura 14. Captura de código fuente de la consulta SQL para la inserción de datos.

```
# Consulta SQL para la inserción en la tabla de evaluaciones
query = """
    INSERT INTO gnuhealth_patient_evaluation
    (patient, temperature, bpm, systolic, diastolic,
     discharge_reason, evaluation_start, healthprof,
     respiratory_rate, osat, glycemia)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """
values = (patient_id, temperature, bpm, systolic, diastolic,
          discharge_reason, evaluation_start, health_prof,
          respiratory_rate, osat, glycemia)
cur1.execute(query, values)
```

Fuente: Elaboración propia.

La validación de esta fase fue exitosa al confirmar que los registros insertados manualmente eran plenamente visibles y editables desde el GNU Health Client, demostrando que la base de datos es susceptible a modificaciones externas controladas sin observar errores de consistencia durante las pruebas realizadas.

4.4.4.2 Integración de la Recepción de Datos con el registro en GNU Health

Al validar el correcto registro manual de datos fisiológicos en la base de datos de GNU Health, el siguiente hito fue automatizar el flujo de información. El objetivo fue integrar la recepción de datos en tiempo real desde el broker EMQX con la lógica de almacenamiento de la fase previa. A diferencia de la fase de pruebas iniciales, en esta implementación la API realiza una extracción automatizada de la carga útil (payload) del mensaje JSON, eliminando la necesidad de entrada manual de datos fisiológicos.

4.4.4.2.1 Configuración de Suscripción y Contexto

Para que el middleware procese la información correcta, el usuario debe definir los parámetros de la sesión al iniciar el script (Nombre del dispositivo, DNI del paciente e ID del hardware). Estos datos permiten que la API se suscriba a los tópicos dinámicos correspondientes para recibir los recursos HL7 FHIR Observation específicos.

4.4.4.2.2 Lógica de Discriminación y Mapeo Dinámico

Una vez recibido el recurso FHIR, el sistema ejecuta una función de procesamiento que utiliza el campo **"id"** del JSON como discriminador principal para identificar la naturaleza de la variable. Se implementó una lógica de mapeo condicional que permite gestionar tanto variables simples como recursos compuestos:

- **Variables Simples:** Para temperatura, frecuencia cardíaca, frecuencia respiratoria, saturación de oxígeno y glucemia, el sistema extrae el valor del campo **valueQuantity** y lo asigna a su columna respectiva.
- **Variables Compuestas (Presión Arterial):** Debido a que el estándar FHIR trata la presión arterial como una observación con múltiples componentes, el código navega por el arreglo de componentes del JSON para extraer de forma independiente la presión sistólica (índice 0) y la diastólica (índice 1).

C. Almacenamiento puntual de datos fisiológicos en GNU Health

Para permitir la integridad de la información en el servidor, el script ejecuta una inicialización en estado nulo (None) de todas las variables biomédicas antes del procesamiento de cada mensaje. De esta manera, al recibir una medición específica (por ejemplo, saturación de oxígeno), solo se actualiza el campo pertinente en el registro, manteniendo los demás vacíos. Esta técnica evita la generación de "ruido" o datos falsos en la base de datos, asegurando que cada entrada en la tabla

gnuhealth_patient_evaluation representa una medición puntual, unívoca y exacta, vinculada estrictamente al paciente y al médico responsable.

Un aspecto importante en la integración fue la gestión de la marca de tiempo. Se optó por extraer el campo `effectiveDateTime` del recurso JSON HL7 FHIR para asignarlo a la columna `evaluation_start` de GNU Health. Esta decisión técnica tiene un significado en la interoperabilidad semántica:

- En el estándar HL7 FHIR, el campo ***effectiveDateTime*** representa el instante real en que se capturó la constante vital en el paciente.
- En el ecosistema de GNU Health, la variable ***evaluation_start*** define el inicio del acto clínico o evaluación.

Al sincronizar ambos campos, se permite que el registro histórico en la HCE refleje el momento exacto de la medición clínica y no simplemente la hora en que el servidor recibió el paquete de datos.

4.4.5 Consolidación e Integración Sistémica del Middleware

Luego de la validación funcional de los flujos independientes hacia MariaDB y PostgreSQL, se procedió a la unificación de los servicios en un único script desarrollado en Python. Este script resultante centraliza la lógica de suscripción, procesamiento de recursos FHIR y almacenamiento en ambas bases de datos, permitiendo una gestión integral de la información recibida a través del broker EMQX en la Raspberry Pi 4.

Para asegurar la efectividad del flujo, el sistema solicita al operador el ingreso manual de los metadatos de la sesión: Nombre del dispositivo, ID del dispositivo y DNI del paciente. Estos parámetros son importantes, ya que permiten al middleware construir y suscribirse dinámicamente a la jerarquía de tópicos definida en la primera etapa de desarrollo. Esta coincidencia exacta es lo que permite que el mensaje publicado por el ESP32 sea capturado y procesado sin pérdidas por el servidor central.

También, se integró una función de retroalimentación hacia el nodo emisor. Una vez que el middleware confirma que la data ha sido procesada y almacenada correctamente en los repositorios de MariaDB y PostgreSQL, el sistema genera y envía un mensaje de confirmación (Acknowledgement - ACK) al ESP32.

Esta comunicación se realiza a través del tópico de salida previamente establecido. Esta funcionalidad permite que el microcontrolador tenga certeza del éxito de la operación, cerrando el ciclo de comunicación y permitiendo al dispositivo retomar sus tareas.

4.5. Implementación de la Interfaz Web de Visualización y Gestión (Etapa 3)

Esta tercera etapa de desarrollo se centra en la construcción de la capa de presentación del sistema, diseñada para ofrecer una interacción intuitiva y técnica con la información biomédica procesada. Su implementación se basó en el framework Streamlit, orientado a la creación de dashboards dinámicos. Para permitir un diseño escalable, el desarrollo se centró en distintos aspectos.

En primer lugar, con el fin de restringir el acceso al personal autorizado, se implementó un sistema de gestión de identidades dentro de la base de datos MariaDB. Se diseñó una tabla específica para el almacenamiento de credenciales (nombres de usuario y contraseñas), permitiendo la administración de múltiples perfiles con acceso a la información clínica. Esta funcionalidad asegura que la trazabilidad de los datos esté acompañada de un control de acceso.

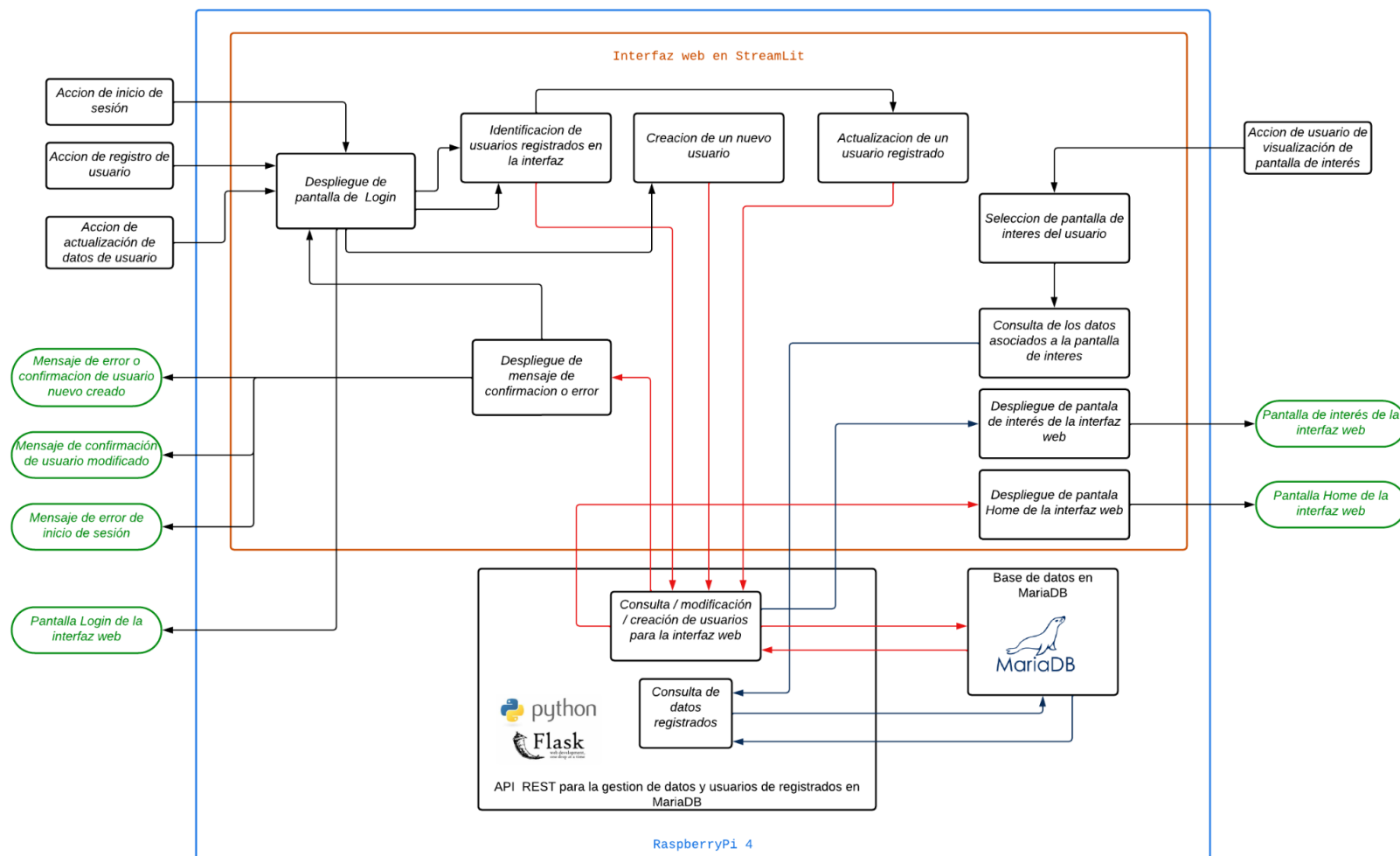
En segundo lugar, una decisión de diseño fue la implementación de una API REST como intermediaria entre la base de datos MariaDB y la interfaz de Streamlit. En lugar de realizar consultas SQL directas desde el frontend, Streamlit consume endpoints específicos mediante peticiones HTTP. Esta arquitectura ofrece tres ventajas estratégicas:

- Separación de Funciones: Streamlit se limita a la renderización de datos (Frontend), mientras que la API gestiona la lógica de negocio y la comunicación con el servidor (Backend).
- Escalabilidad y Flexibilidad: Al exponer la información a través de una API, el sistema queda preparado para que otros servicios o aplicaciones móviles puedan consumir los mismos datos en el futuro.
- Mantenibilidad: Facilita la migración de tecnologías; si en el futuro se optara por cambiar el motor de base de datos, solo sería necesario actualizar la lógica de la API, manteniendo la interfaz de usuario intacta.

La interfaz tiene un doble objetivo: por un lado, asegurar la trazabilidad del flujo de información, verificando que los mensajes JSON lleguen íntegros desde el microcontrolador; y por otro, monitorear en tiempo real las actualizaciones y modificaciones realizadas en las HCE dentro de GNU Health. Como se observa en la Figura 15, el diseño funcional de esta capa de presentación se basa en un modelo de

comunicación bidireccional, donde el usuario interactúa con el entorno gráfico y este delega todo el procesamiento al backend en Flask.

Figura 15. Diagrama funcional de la interfaz web de visualización y gestión.



Fuente: Elaboración propia.

El diagrama ilustra dos flujos principales:

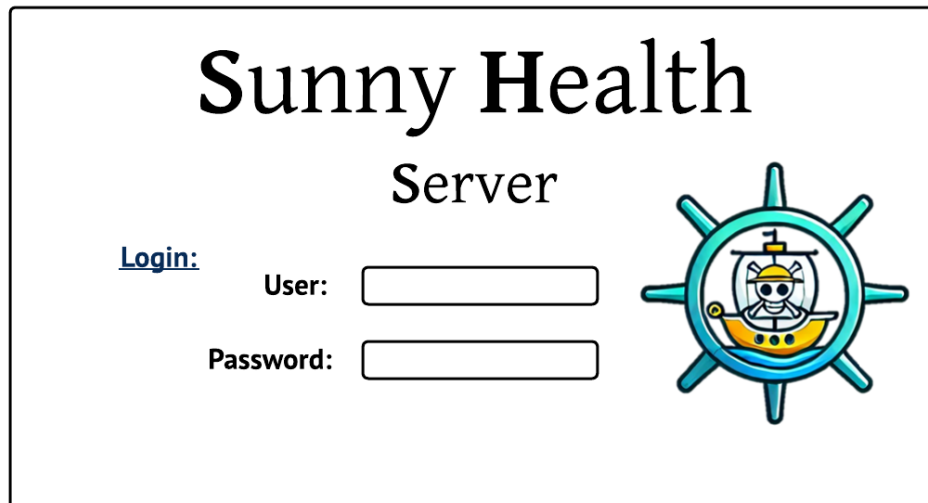
- ❖ **Gestión de Sesión (Líneas Rojas):** Permite el control de acceso mediante la validación, creación y actualización de usuarios en la base de datos MariaDB.
- ❖ **Visualización de Datos (Líneas Azules):** Realiza consultas dinámicas a los registros para desplegar pantallas de interés y monitorear la trazabilidad del sistema.

4.5.1 Diseño de la Interfaz de Usuario (UI)

Previo al desarrollo del código, se definieron los requerimientos visuales y funcionales de la plataforma. El diseño se estructuró en seis pantallas principales, interconectadas para facilitar la navegación clínica y la supervisión técnica del flujo de datos:

- **Pantalla de Inicio de Sesión (Login):** Constituye la puerta de enlace al sistema. Su función es la autenticación del personal mediante credenciales almacenadas. Una validación correcta redirige automáticamente al usuario hacia el panel principal (Home).

Figura 16. Diseño inicial de la interfaz de Login.

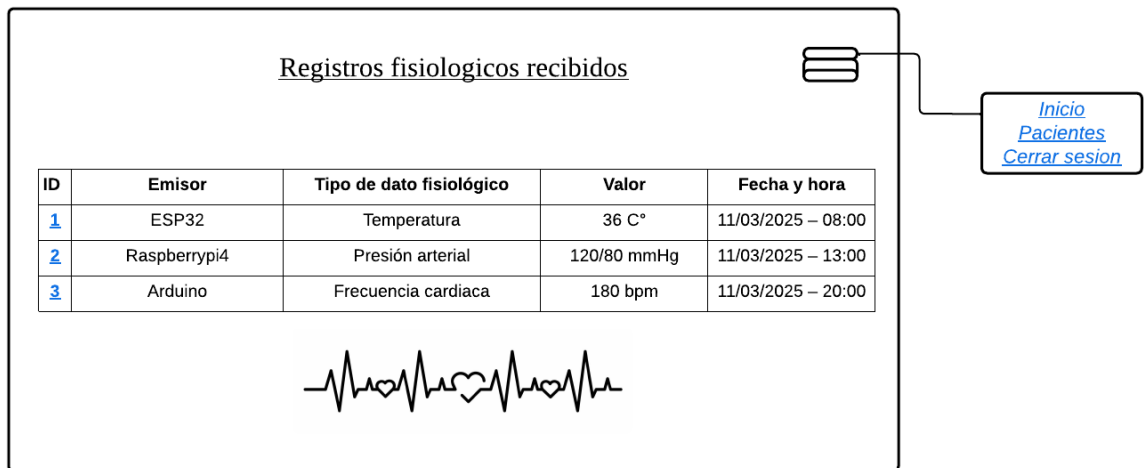


The image shows a login interface for 'Sunny Health Server'. At the top, the text 'Sunny Health' is in a large, black, serif font, with 'Server' below it in a smaller, black, serif font. To the left of the input fields, the word 'Login:' is underlined in blue. Below it, the labels 'User:' and 'Password:' are in black, followed by empty rectangular input boxes. To the right of the input fields is a cartoon illustration of a ship's steering wheel with a yellow and white boat in the center. The entire interface is enclosed in a black rectangular border.

Fuente: Elaboración propia.

- **Pantalla Principal (Home):** Actúa como el centro de control (Dashboard) de la interfaz. Presenta una vista tabular con todos los registros capturados en la base de datos MariaDB. Incluye un menú de navegación lateral (Sidebar) persistente y permite seleccionar registros específicos para una inspección detallada.

Figura 17. Diseño inicial de la interfaz Home.



Fuente: Elaboración propia.

- **Pantalla de Detalles (Details):** Se accede mediante la selección de un registro en la pantalla Home. Expone de forma tabular los metadatos de la medición (Nombre, DNI, médico responsable, etc.) y proporciona una vista técnica del recurso HL7 FHIR original en formato JSON, permitiendo verificar la integridad de la serialización.

Figura 18. Diseño inicial de la interfaz Details.

Registro de valor fisiológico de [Willy Valdivia](#)


ID del registro	2
-----------------	---

Paciente	Willy Valdivia	ID Paciente	72613637
Medico	Cristopher Casani	ID Medico	72613638
Dispositivo	Glucómetro	ID Dispositivo	2048

Tipo de dato fisiológico	Frecuencia cardiaca
Valor	180
Unidad	bpm
Fecha y Hora de Registro	2020-10-14T02:00:00Z

Mensaje completo:

```

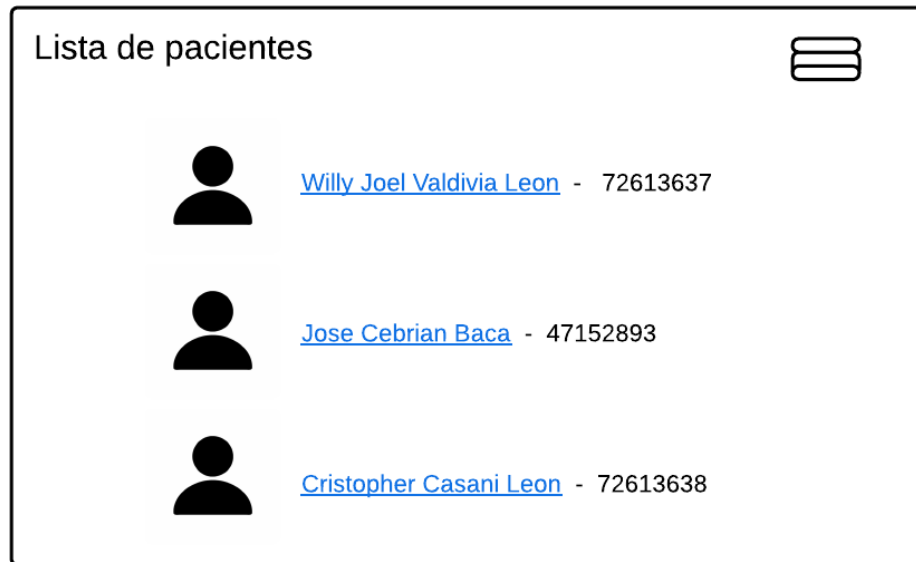
{
  "resourceType": "Observation",
  "id": "respiratory-rate",
  "meta": {
    "profile": [
      "http://hl7.org/fhir/StructureDefinition/resprate"
    ]
  },
  "text": {
    "status": "generated",
    "div": "<div
xmlns=\\\"http://www.w3.org/1999/xhtml\\\">Respiratory rate</div>"
  },
  .....
  .....
  .....

```

Fuente: Elaboración propia.

- **Directorio de Pacientes (Patients):** Accesible desde el menú lateral, esta vista organiza la información bajo un enfoque centrado en el usuario. Despliega un listado de todos los pacientes que cuentan con registros históricos en el sistema, sirviendo como punto de entrada para el análisis clínico individualizado.

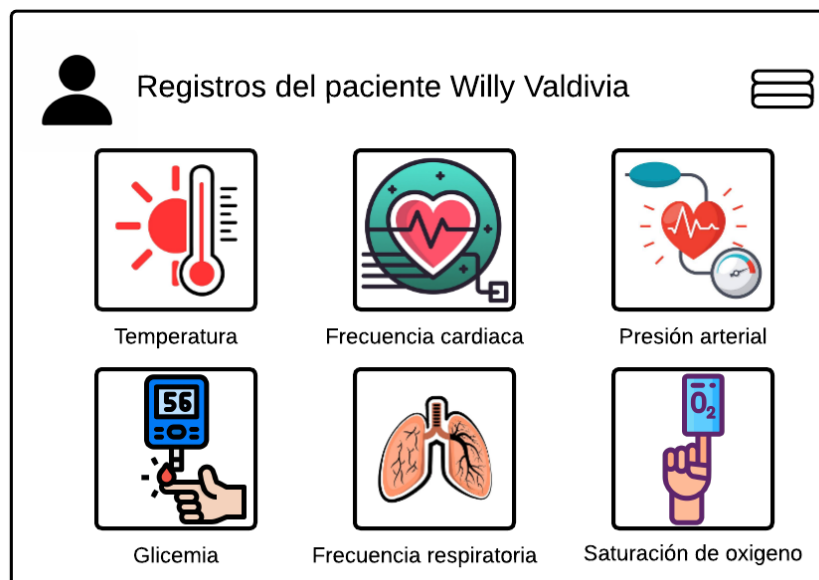
Figura 19. Diseño inicial de la interfaz Patients.



Fuente: Elaboración propia.

- **Detalles por Paciente (Patients-details):** Luego de seleccionar un paciente, esta pantalla permite discriminar la información según las seis variables fisiológicas definidas en la tesis. El usuario puede filtrar y seleccionar el parámetro de interés (ej. glucemia o temperatura) para acceder a su historial específico.

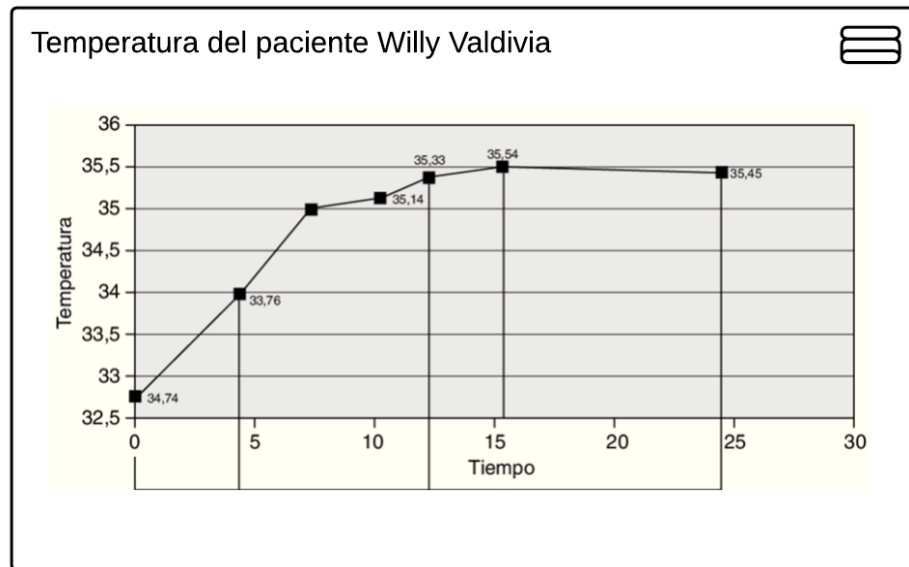
Figura 20. Diseño inicial de la interfaz Patients_details.



Fuente: Elaboración propia.

- **Visualización de Tendencias (Data-view):** Es el módulo de análisis temporal. Presenta los valores de la variable seleccionada mediante gráficas de series de tiempo, facilitando la detección de anomalías o tendencias clínicas a lo largo del periodo de monitoreo.

Figura 21. Diseño inicial de la interfaz Data_view.



Fuente: Elaboración propia.

Estas pantallas fueron diseñadas bajo principios de usabilidad, asegurando que el personal de salud pueda transitar desde una visión general del sistema hasta el análisis detallado de una constante vital específica de forma ágil.

4.5.2 Desarrollo de pantallas principales

Para agilizar el ciclo de desarrollo y asegurar la correcta visualización de la información clínica, se optó por un enfoque de prototipado incremental. En esta fase inicial, se priorizó la construcción de las pantallas de visualización de datos sobre los módulos de autenticación, permitiendo validar la integridad de la información en la interfaz de forma inmediata.

4.5.2.1 Gestión de Estado y Enrutamiento Modular

La arquitectura de la interfaz se basa en un modelo de programación reactiva proporcionado por Streamlit. Se desarrolló un script principal que actúa como enrutador del sistema, utilizando la variable de entorno `st.session_state["page"]` como la fuente única de verdad para la navegación. La transición entre pantallas se gestiona mediante una lógica de cambio de estado: al interactuar con botones o menús, el sistema actualiza

el puntero de la página y ejecuta un renderizado (`st.rerun()`) para desplegar el módulo correspondiente. Este diseño permite una navegación fluida y modular, donde cada pantalla está contenida en su propio script importado, facilitando el mantenimiento del código.

4.5.2.2 Metodología de Acceso a Datos y Visualización

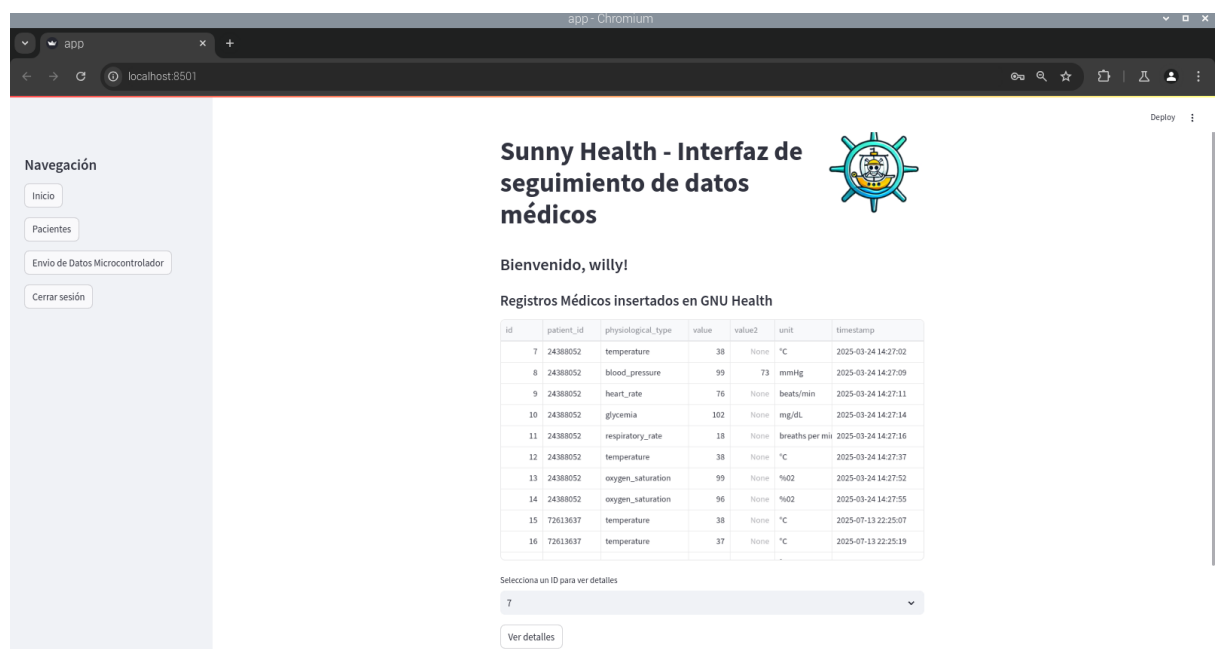
Durante esta etapa de desarrollo, y con el fin de acelerar las pruebas de estrés en la UI, se implementó una conexión directa a la base de datos MariaDB mediante la librería `mysql.connector`. Aunque la arquitectura final contempla el uso de una API REST, este acceso directo permitió validar rápidamente el mapeo de los registros. Para el tratamiento y presentación de la información, se integraron las siguientes herramientas:

- **Pandas:** Utilizada para la estructuración y manipulación de los datos en marcos de datos (dataframes), permitiendo una visualización tabular organizada y eficiente.
- **Plotly:** Implementada en la interfaz Data-view para la generación de gráficos interactivos de series de tiempo, permitiendo al usuario realizar zoom y seguimiento detallado de las tendencias fisiológicas.

4.5.2.3 Diseño final de las pantallas principales

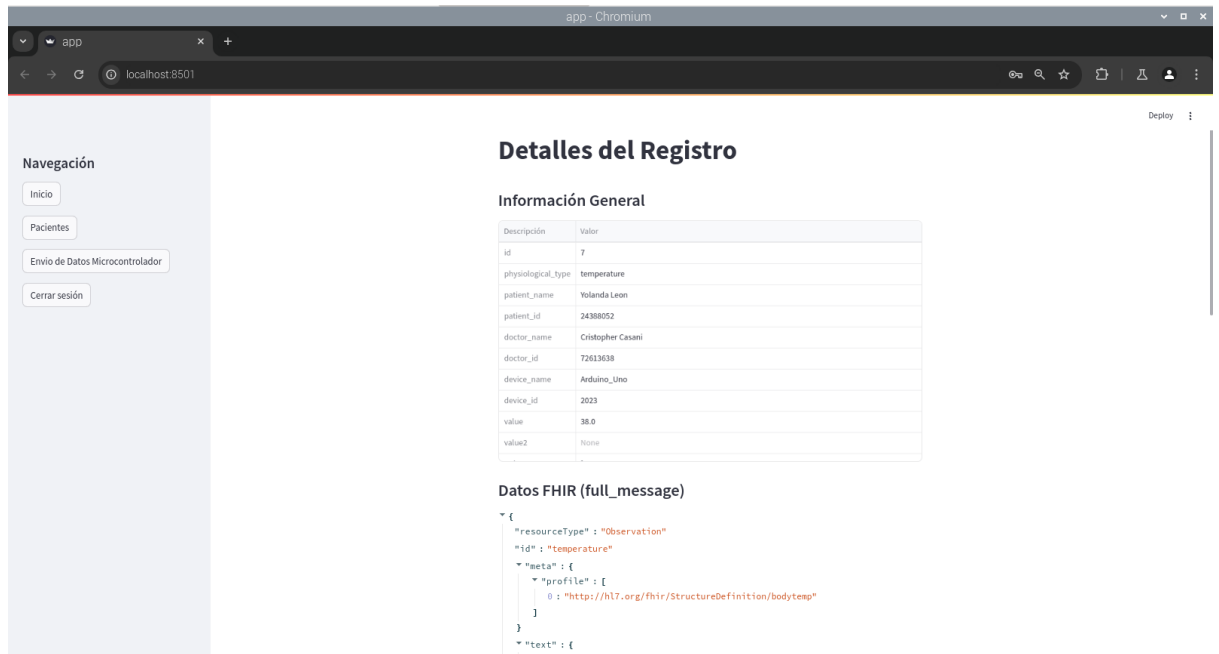
Al haber implementado las herramientas antes descritas, a continuación, se presenta el diseño final de las pantallas principales de la interfaz web desarrollada en Streamlit.

Figura 22. Diseño final de la pantalla Home



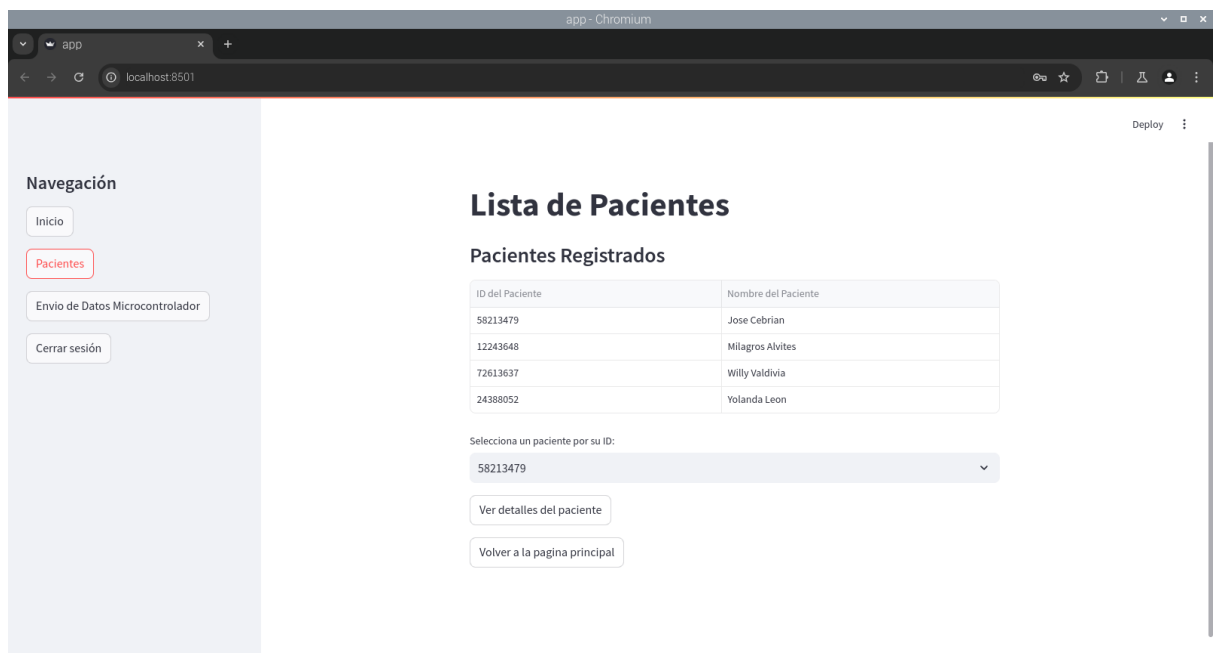
Fuente: Elaboración propia

Figura 23. Diseño final de la pantalla Details



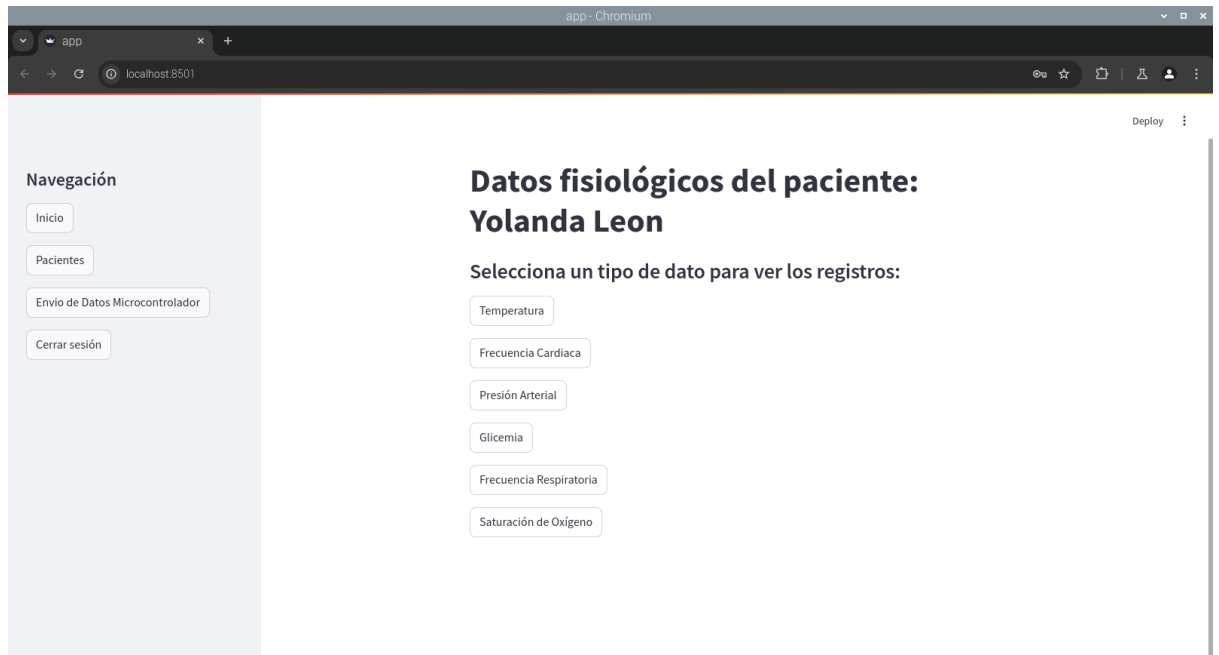
Fuente: Elaboración propia

Figura 24. Diseño final de la pantalla Patients



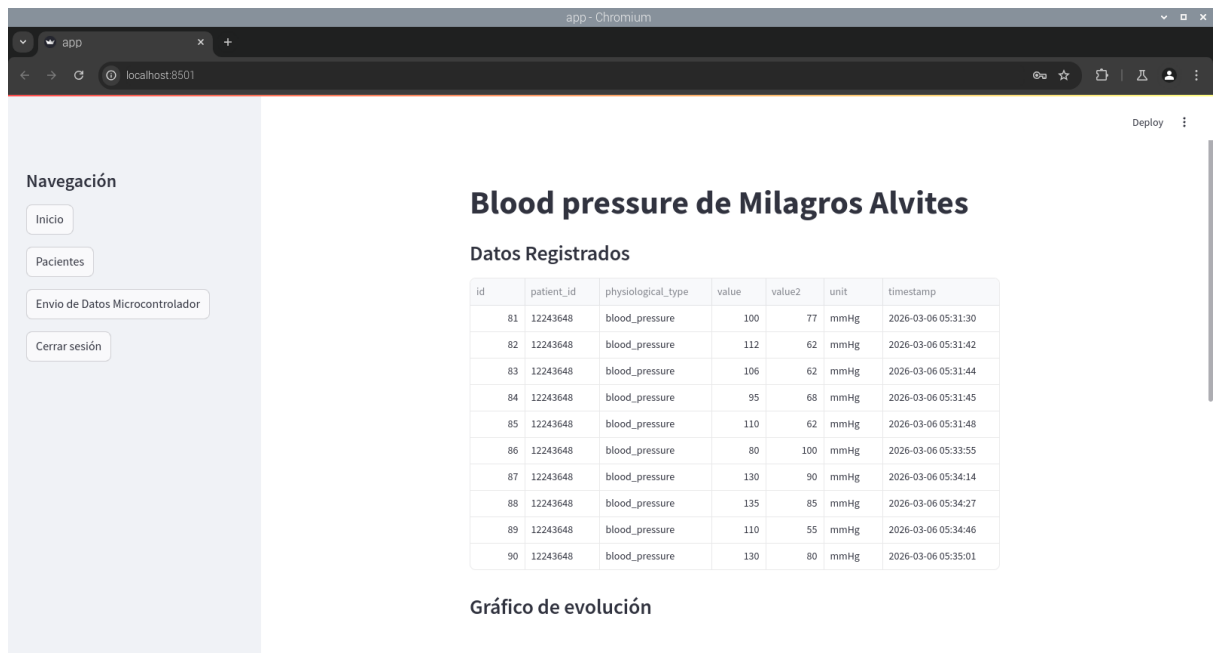
Fuente: Elaboración propia

Figura 25. Diseño final de la pantalla Patients_details



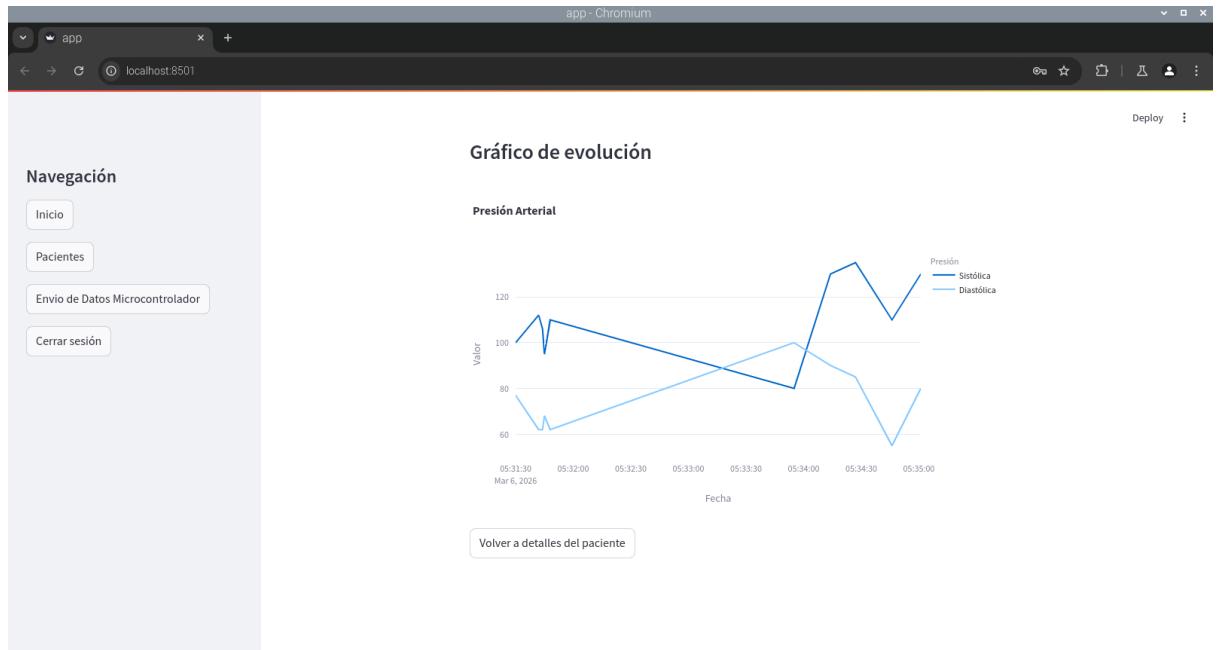
Fuente: Elaboración propia

Figura 26. Diseño final de la pantalla Data_view (Parte 1)



Fuente: Elaboración propia

Figura 27. Diseño final de la pantalla Data_view (Parte 2)

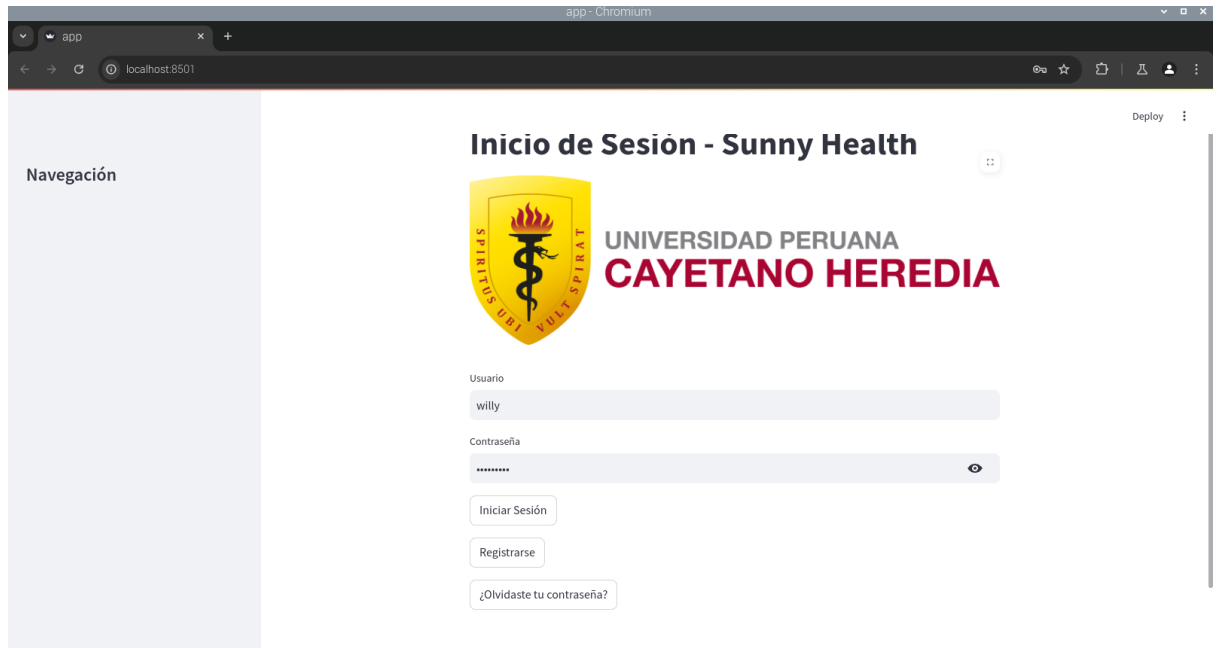


Fuente: Elaboración propia

4.5.3 Desarrollo de pantallas de acceso

Una vez validada la funcionalidad de las pantallas de visualización de datos, se procedió a implementar la capa de seguridad y control de acceso. El mecanismo de autenticación se apoya en una tabla específica denominada *profile* dentro de la base de datos MariaDB, la cual gestiona el almacenamiento de las identidades autorizadas para operar la interfaz. A continuación, se muestra el diseño final de la interfaz Login que gestiona el ingreso a usuarios:

Figura 28. Diseño final de la pantalla Login



Fuente: Elaboración propia

4.5.3.1 Criptografía y Protección de Credenciales

Bajo estándares de seguridad informática, se determinó que el sistema no almacenará contraseñas en texto plano. En su lugar, se integró la librería `werkzeug.security` para la generación de hashes. Mediante el método `generate_password_hash`, cada clave es transformada en una cadena alfanumérica irreversible antes de su inserción en la base de datos, lo que permite la confidencialidad de la información ante posibles vulneraciones del repositorio. Cabe resaltar que, para validar este funcionamiento se creó un usuario con el nombre “willy” y contraseña “5729willy”,

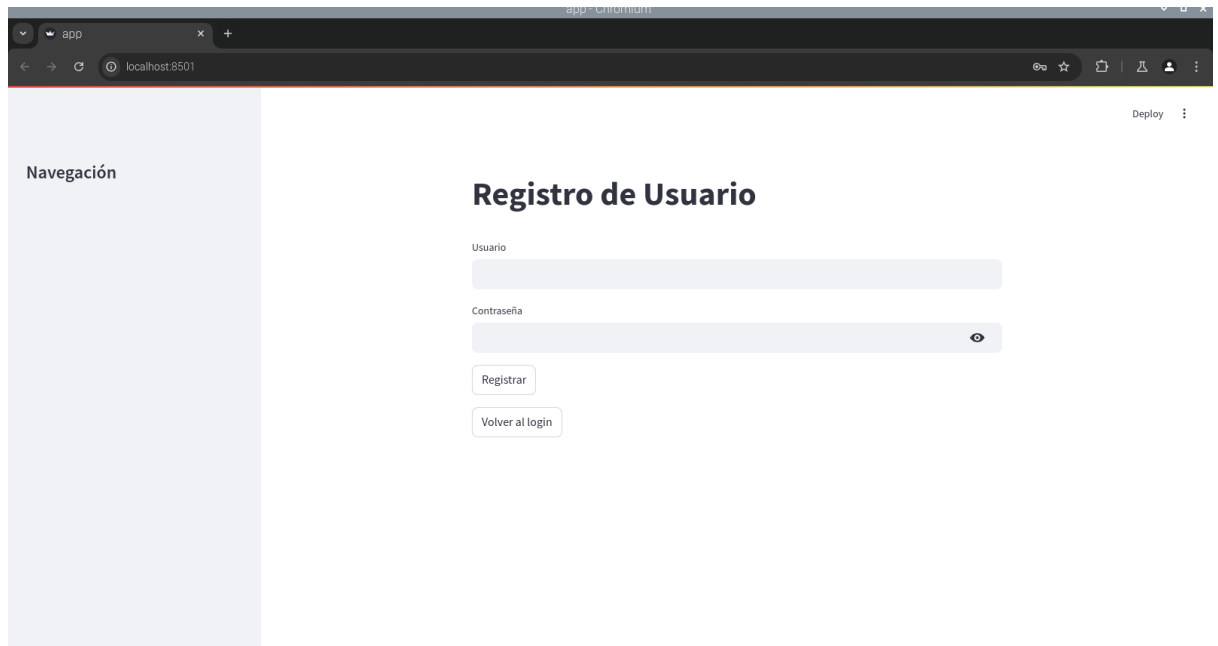
4.5.3.2 Gestión de Registro (Pantalla Register)

Por otro lado, se decidió desarrollar una pantalla de registro, la cual permite el aprovisionamiento de nuevos usuarios médicos o administrativos. Para asegurar la calidad de las credenciales, se programaron tres filtros de validación obligatorios:

- **Unicidad de Usuario:** El sistema verifica que el nombre de usuario no se encuentre registrado previamente en la tabla `profile`.
- **Longitud Mínima:** Se exige una extensión superior a 8 caracteres.
- **Complejidad Alfanumérica:** Se requiere una combinación de letras y números para aumentar la resistencia ante ataques de fuerza bruta.

A continuación, se muestra el diseño final de la pantalla register que permite agregar nuevos usuarios:

Figura 29. Diseño final de la pantalla Register



The screenshot shows a web browser window with the address bar displaying 'localhost:8501'. The page has a dark theme. On the left, there is a light gray sidebar with the text 'Navegación'. The main content area is titled 'Registro de Usuario' in bold. Below the title, there are two input fields: 'Usuario' and 'Contraseña'. The 'Contraseña' field has an eye icon on the right side. Below the input fields, there are two buttons: 'Registrar' and 'Volver al login'.

Fuente: Elaboración propia

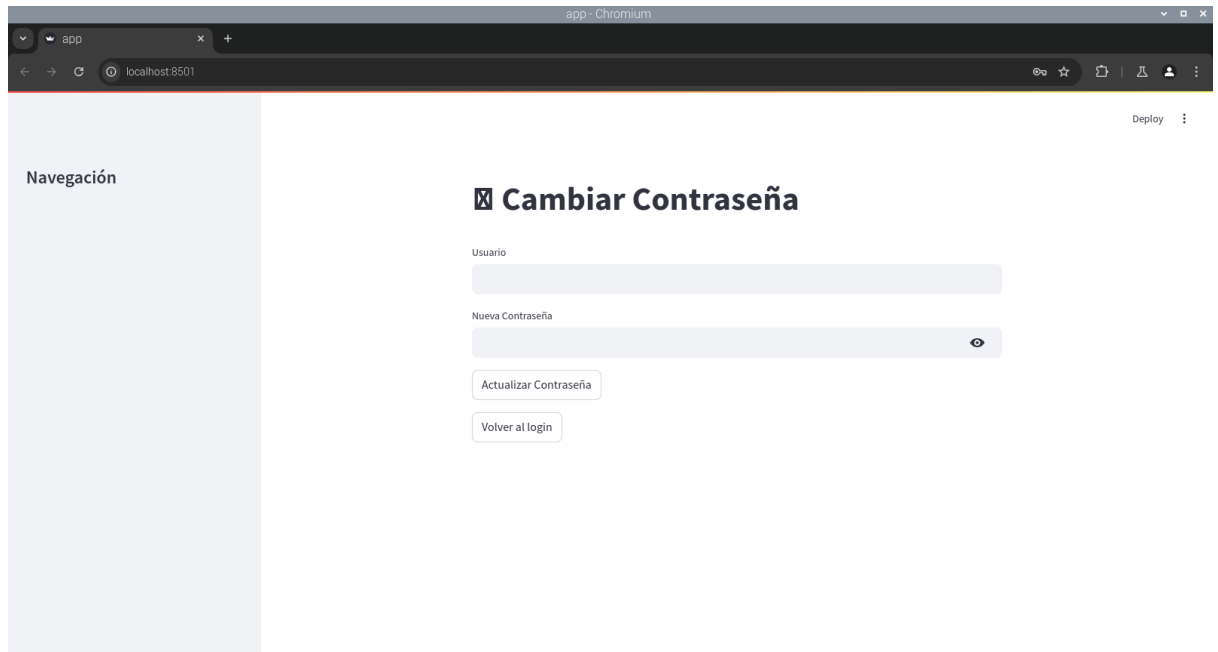
4.5.3.3 Recuperación y Actualización (Pantalla Change-password)

Se implementó una interfaz de gestión de credenciales que permite la actualización de contraseñas existentes. Esta pantalla actúa como un mecanismo de recuperación en caso de olvido o por políticas de rotación de seguridad.

Al igual que en la interfaz de registro, este módulo aplica la validación de longitud y el proceso de re-hasheo antes de actualizar el registro en MariaDB. Al completar satisfactoriamente el proceso de autenticación en la pantalla de Login, el sistema utiliza el estado de la sesión (*st.session_state*) para otorgar acceso y redirigir automáticamente al usuario hacia la interfaz principal (pantalla Home), habilitando así el resto de las funcionalidades de la plataforma.

A continuación, se muestra el diseño final de la pantalla change_password que permite actualizar las contraseñas de los usuarios ya registrados:

Figura 30. Diseño final de la pantalla Change-password



Fuente: Elaboración propia

4.5.4 Desarrollo de API RESTful para interfaz web

La arquitectura de la interfaz se sustenta en una API RESTful que actúa como middleware o capa intermedia entre los repositorios de datos y el frontend en Streamlit. Para su implementación, se seleccionó el micro-framework Flask, configurando diversos endpoints bajo una estructura puramente REST. Esta decisión técnica responde a tres objetivos:

- Modularidad: Permite el crecimiento modular del sistema sin afectar la interfaz de usuario.
- Mantenibilidad: La segmentación de funciones en endpoints específicos facilita la detección y corrección aislada de errores.
- Interoperabilidad: Al seguir el estándar REST, se permite que otros dispositivos o servicios externos puedan consumir la información almacenada en MariaDB de forma segura y uniforme.

Para asegurar un acceso estructurado a la información biomédica, la API se organizó en cuatro módulos funcionales, cuyos resultados se entregan en formato JSON. Se utilizó la estructura `OrderedDict` para preservar el orden semántico de los campos clínicos, permitiendo la legibilidad de la data.

Tabla 4. Módulos funcionales de la API

Módulo	Funcionalidad Principal	Métodos HTTP
Observaciones	Recuperación masiva y detallada de registros fisiológicos.	GET /data, GET /data/<id>
Identidad Clínica	Filtrado dinámico de la data fisiológica por paciente, médico o dispositivo.	GET /patients, GET /doctors, GET /devices
Análisis Temporal	Obtención de la última medición o historial por tipo de variable.	GET /patients/<id>/<tipo>/latest
Gestión de Acceso	Registro, validación y actualización de credenciales.	POST /users, GET /users, PATCH /users

Fuente: Elaboración propia.

Por otro lado, se definió una ruta raíz (/) que funciona como documentación dinámica, desplegando un mapa de todas las rutas disponibles y sus funciones específicas. Asimismo, la API realiza un pre-procesamiento de los datos antes de su entrega final:

- **Serialización "Pretty Print":** Las respuestas incluyen indentación y soporte completo para caracteres UTF-8, facilitando la inspección técnica del recurso HL7 FHIR original contenido en el campo full_message.
- **Normalización Temporal:** Mediante la librería pytz, la API estandariza los timestamps de las mediciones al formato YYYY-MM-DD HH:MM:SS, eliminando ambigüedades horarias en la interfaz.

4.5.5 Integración de API RESTful con la interfaz de usuario

La fase final del desarrollo de la interfaz de visualización y gestión consistió en la vinculación dinámica del frontend desarrollado en Streamlit con la API. En esta fase, se realizó una reestructuración del código base: se eliminó el acceso directo de la interfaz a la base de datos MariaDB, sustituyéndolo por un modelo de consumo de servicios web. Este cambio permite que la interfaz de usuario sea independiente de la infraestructura de datos, facilitando el mantenimiento y la seguridad.

4.5.5.1 Protocolo de Comunicación y Flujo de Datos

La comunicación se centralizó en el protocolo HTTP, empleando la librería *requests* para el intercambio de información. Para asegurar una experiencia de usuario fluida y la integridad de los datos clínicos, se implementó un flujo de procesamiento dividido en tres etapas:

- **Petición de Recursos (Request):** La interfaz solicita datos específicos (ej. /patients para el listado o /data para registros generales).
- **Validación de Respuesta:** El sistema evalúa los códigos de estado HTTP. Si el servidor responde con un código 200 OK, se procede al procesamiento; de lo contrario, se activan alertas mediante `st.error` (errores de servidor) o `st.warning` (registros no encontrados).
- **Normalización y Deserialización:** Los objetos JSON recibidos son transformados en estructuras Pandas (DataFrames). Esta normalización permite que los datos médicos se presenten en tablas interactivas y se utilicen en filtros de búsqueda en tiempo real.

4.5.5.2 Gestión de Estado mediante la variable `st.session_state`

Debido a que Streamlit se ejecuta de forma lineal en cada interacción, se utilizó el objeto *st.session_state* como una capa de memoria compartida entre interfaces. Este mecanismo permite:

- Seguridad de sesión: Almacena la identidad del usuario y su estado de autenticación de forma global.
- Transferencia de Contexto: Al seleccionar un paciente en la vista general, su identificador único (id) persiste en el estado de la sesión, permitiendo que las pantallas de detalle consulten el endpoint correcto sin perder la trazabilidad.

4.5.5.3. Operaciones de Autenticación y Gestión de Usuarios

La integración del sistema contempla la manipulación segura de la tabla `profile` en MariaDB, utilizando la API como una capa de abstracción que impide el acceso directo desde el frontend. Cabe resaltar que, la interfaz reacciona dinámicamente según los códigos de estado HTTP recibidos, informando al usuario sobre conflictos de duplicidad o errores de conexión. Para gestionar el ciclo de vida de las credenciales, se implementaron tres operaciones:

- **Registro de Usuarios (Método POST):** Para el aprovisionamiento de nuevas cuentas, la interfaz envía una petición POST al endpoint /users. Esta operación integra la lógica descrita en la sección 4.5.3.2, asegurando que antes de la inserción en la base de datos, los datos cumplan con los filtros de unicidad y complejidad alfanumérica. La API transforma la contraseña en un password_hash irreversible mediante la librería Werkzeug.
- **Validación de Acceso (Método GET):** El proceso de Login se apoya en el método GET hacia el recurso /users/<username>. Este endpoint devuelve el hash almacenado. Streamlit utiliza check_password_hash para comparar la entrada con el registro. Si es correcta, se inicializa el st.session_state y se permite la navegación a la pantalla Home.
- **Actualización de Credenciales (Método PATCH):** En cumplimiento con las funcionalidades de rotación de seguridad y recuperación de cuentas, se implementó el método PATCH para el endpoint /users/<username>. Este método permite una actualización parcial del registro, modificando únicamente el campo password_hash sin alterar otros atributos del perfil. Al igual que en el registro, la nueva contraseña es validada y procesada antes de ser almacenada en MariaDB.

4.5.5.4 Visualización de Datos Clínicos y Navegación Dinámica

Una vez superada la capa de autenticación, el sistema despliega la información médica a través de un flujo de navegación jerárquico que consume los módulos de Observaciones e Identidad Clínica de la API. Aplicando los principios de comunicación y almacenamiento de estado previamente descritos (secciones 4.5.5.1 y 4.5.5.2), esta integración se divide en cuatro niveles operativos:

- **Vista Global (Módulo de Observaciones):** Al iniciar, la pantalla principal ejecuta una petición GET /data. La interfaz procesa el arreglo JSON recibido y, mediante la librería Pandas, genera una tabla interactiva. Esto permite al personal médico visualizar de forma consolidada todos los registros capturados por los dispositivos IoT.
- **Vista de Detalle e Interoperabilidad:** Al seleccionar un registro, se realiza un GET /data/<id>. El sistema extrae el full_message y lo presenta estructurado con st.json(). Esto permite auditar los valores procesados y el recurso estándar HL7 FHIR original, permitiendo la trazabilidad.

- **Filtros por Identidad:** En la sección de pacientes, la interfaz consume el endpoint GET /patients. Al seleccionar un paciente del listado, el sistema utiliza el mecanismo st.session_state para capturar su ID y filtrar automáticamente todas las constantes fisiológicas (como temperatura, frecuencia cardíaca o saturación) asociadas a ese perfil clínico, optimizando el tiempo de consulta.
- **Vista Analítica y Gráfica Evolutiva (Data View):** Al seleccionar un paciente y una constante fisiológica, se consume el endpoint GET /patients/<id>/<tipo>. La API responde con un JSON pre-formateado mediante OrderedDict. La interfaz convierte estos datos en un DataFrame y emplea Plotly Express para renderizar gráficos de líneas interactivas.

Cabe destacar el tratamiento especial en la Vista Analítica para variables complejas como la presión arterial. Si el sistema detecta el campo value2 asociado a blood_pressure, el DataFrame se reestructura utilizando la función melt de Pandas. Esto transforma los datos a un formato largo (long format), permitiendo que Plotly grafique simultáneamente la presión sistólica y diastólica en una misma línea temporal, facilitando el análisis clínico de la evolución del paciente.

Con la culminación de esta fase de visualización y consumo de servicios web, la interfaz de visualización y gestión queda finalizada y plenamente operativa. El sistema cuenta ahora con una arquitectura preparada para recibir los datos generados por el nodo simulador dentro del flujo implementado.

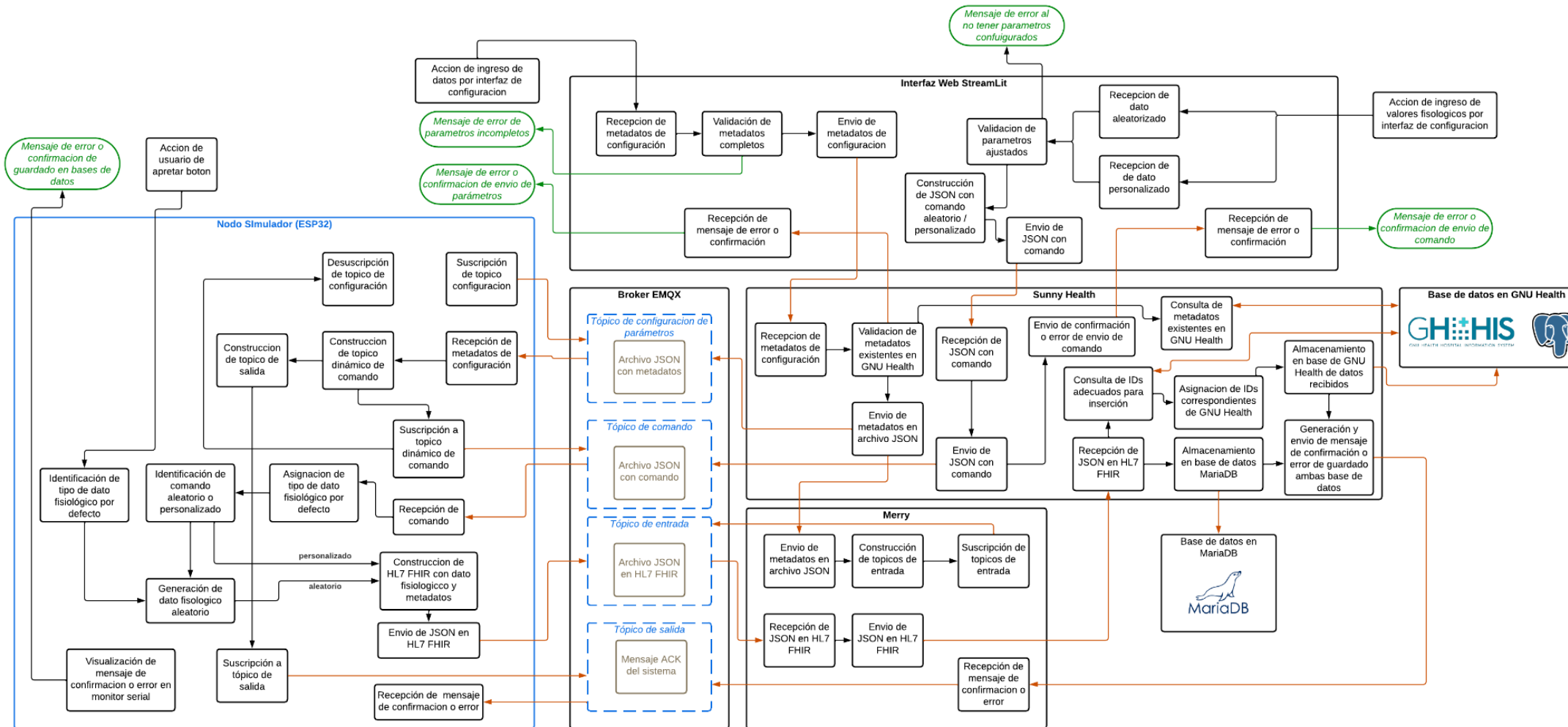
4.6 Implementación del Sistema Integrado (Etapa 4)

En la cuarta y última etapa de desarrollo, se procedió a la consolidación del sistema tecnológico. Para lograr un flujo de información bidireccional y estructurado, se implementaron funciones adicionales que unificaron los módulos desarrollados en las fases previas. Esta etapa permite una correcta interoperabilidad entre el nodo simulador, el entorno de procesamiento y las bases de datos finales.

Como se evidencia en el diagrama funcional general (Figura 31), el sistema opera bajo un modelo de comunicación asíncrona centralizado por el broker EMQX, el cual gestiona cuatro canales principales (tópicos de configuración, comando, entrada y salida). El flujo de información inicia en la Interfaz Web (Streamlit), donde el usuario ingresa los metadatos de configuración o comandos de generación de datos. Esta información es validada y enrutada por la capa de integración hacia el broker MQTT. El Nodo Simulador

(ESP32) recibe estos parámetros, se suscribe dinámicamente y, tras la acción física (botón) o el comando remoto, genera el dato fisiológico en formato HL7 FHIR. Este archivo es capturado por el entorno de procesamiento (Merry y Sunny), el cual consulta las identidades correspondientes y ejecuta el almacenamiento simultáneo en GNU Health y MariaDB, cerrando el ciclo mediante un mensaje de confirmación en la interfaz de usuario.

Figura 31. Diagrama funcional del sistema integrado final.



Fuente: Elaboración propia.

4.6.1 Estrategia de separación de funciones en la arquitectura de la API final

Para la consolidación del sistema, se estableció que la API desarrollada en la tercera etapa actuaría como el núcleo funcional de toda la arquitectura. Durante la Etapa 2, el almacenamiento de datos en GNU Health se logró mediante un script independiente en Python. Para centralizar las operaciones, este script se integró a la API REST encargada del frontend.

Adicionalmente, se identificó una oportunidad de mejora respecto a la Etapa 1: la configuración de los parámetros de envío HL7 FHIR del nodo simulador dependía de una conexión física mediante un monitor serial. Para optimizar este proceso, se incorporó un módulo en la interfaz web que permite enviar metadatos de configuración al microcontrolador de forma remota a través de MQTT.

Con el objetivo de separar la gestión de la mensajería MQTT de la lógica principal de procesamiento y almacenamiento, se adoptó una arquitectura basada en dos servicios independientes. De esta manera, se subdividió las responsabilidades en dos interfaces independientes: el servicio Merry (orientado a la mensajería) y el servicio Sunny (orientado a la lógica de negocio), tal como se ilustró previamente en el diagrama funcional de la Figura 31.

4.6.1.1 Desarrollo del servicio Merry (API de Enrutamiento y Mensajería MQTT)

Este microservicio opera como la pasarela de comunicación asíncrona del sistema. Su función principal es la escucha constante del tópico de entrada asociado al nodo simulador. La dinámica operativa inicia cuando la API principal (Sunny) le transfiere los metadatos necesarios para construir y suscribirse dinámicamente a los tópicos MQTT correspondientes.

Cuando el servicio Merry detecta el ingreso de un mensaje con formato HL7 FHIR en el broker, captura la carga útil (payload) y la retransmite inmediatamente a la API principal mediante una petición HTTP. Una vez que Sunny procesa la información, devuelve a Merry una respuesta de confirmación o un código de error, el cual es finalmente publicado en el tópico de salida del nodo simulador para cerrar el ciclo de retroalimentación.

4.6.1.2 Desarrollo del servicio Sunny Health (API Principal de Integración y Lógica de Negocio)

Este servicio opera como el backend central del sistema. Sus responsabilidades abarcan la gestión de la información renderizada en la interfaz web de Streamlit, el control de acceso y gestión de usuarios, la configuración remota de los nodos simuladores y la ejecución de las inserciones de almacenamiento en las bases de datos.

Como se mencionó en la Etapa 3 de la metodología, las funcionalidades de gestión de identidades (incluyendo el hashing de contraseñas) y la provisión de datos al frontend ya habían sido validadas mediante una arquitectura RESTful preliminar, cuyas capacidades fueron absorbidas e integradas en su totalidad por la API Sunny en esta fase final.

Por otro lado, en lo que respecta a la interoperabilidad clínica, Sunny actúa como receptor final del flujo de datos. La API espera la llegada de un documento JSON (estructurado bajo el estándar HL7 FHIR) proveniente del middleware Merry a través de una petición HTTP POST. Al interceptar la carga útil, el servicio procesa la estructura, mapea los identificadores internos correspondientes y ejecuta la inserción oficial del registro clínico en la base de datos PostgreSQL del entorno GNU Health.

De manera simultánea, Sunny replica el almacenamiento de estos valores en la tabla observations de la base de datos MariaDB. Esta estrategia de diseño arquitectónico es importante para el rendimiento del sistema: MariaDB actúa como una base de datos espejo de alta velocidad dedicada exclusivamente a alimentar los paneles de visualización (Dashboards) de Streamlit. Al derivar las consultas de lectura hacia esta base de datos más ligera, se evitan accesos continuos al sistema de GNU Health.

4.6.2 Integración del Nodo Simulador al ecosistema final

Como se mencionó anteriormente, se implementaron funciones para la integración de los diferentes módulos. De esta manera, para la integración del nodo simulador se implementaron funciones en la interfaz, API y microcontrolador. A continuación, se describen esas implementaciones.

4.6.2.1 Desarrollo del módulo de configuración remota (Pagina Settings)

En la interfaz web desarrollada en Streamlit, se implementó un módulo de configuración remota accesible desde el menú lateral de navegación. Esta sección, diseñada con instrucciones claras para el usuario, permite centralizar el control operativo del microcontrolador sin necesidad de conexiones físicas.

4.6.2.1.1 Interfaz de entrada de metadatos clínicos y técnicos

Se estructuró un formulario web para capturar los identificadores (ID) y nombres del paciente, el médico tratante y el dispositivo biomédico. Para permitir la integridad de los mensajes MQTT, se programó una validación en el frontend que bloquea la ejecución si detecta campos vacíos.

Una vez completado el formulario, al presionar el botón de envío, la interfaz consume el endpoint ***/mqtt/setting*** de la API Sunny. Como medida de seguridad e integridad de datos, este servicio backend verifica primero la existencia y validez de los IDs ingresados (paciente y médico) contra la base de datos de GNU Health, aprovechando la lógica de validación desarrollada en la Etapa 2. Luego de una confirmación, la API ejecuta dos acciones simultáneas: establece conexión con el broker MQTT para publicar un archivo JSON con los metadatos en el tópico global ***sunnyhealth/setting*** (dirigido al microcontrolador), y realiza una petición HTTP POST hacia el servicio Merry para transferir estos mismos parámetros, permitiendo que dicha API construya sus propios tópicos de escucha.

4.6.2.1.2 Interfaz de control para la generación de datos

Debajo del panel de metadatos, se diseñó un módulo interactivo para accionar la transmisión de datos desde el nodo simulador. Se implementaron dos modalidades mediante botones de acción: envío de dato aleatorio y envío de dato personalizado. En ambos casos, un menú desplegable permite seleccionar el tipo de variable fisiológica a simular (por ejemplo, frecuencia cardíaca o saturación de oxígeno). La modalidad personalizada habilita además un campo de texto para ingresar un valor clínico específico (o dos valores separados por un espacio, en el caso de variables compuestas como la presión arterial sistólica y diastólica).

Para asegurar la consistencia del ecosistema, estos controles de generación permanecen estrictamente inhabilitados (mediante la gestión de la variable de estado ***session_state.configurado***) hasta que los metadatos de configuración hayan sido enviados y validados. Esto evita la generación de información clínica huérfana (sin un paciente o dispositivo asociado).

Al ejecutar un envío de comando, la interfaz web invoca el endpoint ***/mqtt/command*** de la API principal. El backend recibe el tipo de dato y, de forma opcional, el valor asignado. Con esta información construye el payload y genera un tópico MQTT dinámico y

exclusivo para el nodo objetivo. Según el diseño del código, este canal de comando adopta la siguiente estructura jerárquica:

sunnyhealth/nombre_dispositivo/id_dispositivo/command

4.6.2.1.3 Lógica de tópicos dinámicos y solución de escalabilidad

La implementación de tópicos dinámicos para los comandos, en lugar de un tópico global, responde a una decisión arquitectónica crítica para evitar la colisión de datos. Si se utilizara un tópico común, todos los dispositivos conectados a la red reaccionarían al mismo comando simultáneamente.

No obstante, el tópico inicial de configuración (***sunnyhealth/setting***) sí es de carácter global. Para evitar que la configuración de un nuevo equipo altere a los nodos ya operativos, se diseñó una regla de software en el flujo del sistema: una vez que el nodo simulador recibe y procesa sus parámetros de configuración por el tópico global, se desuscribe automáticamente del mismo y pasa a escuchar únicamente su tópico dinámico de comandos. Esta lógica permite aislar los canales de comunicación de cada nodo después de su configuración. Sin embargo, la asignación inicial de parámetros debe realizarse de manera secuencial, ya que el funcionamiento concurrente con múltiples dispositivos no fue evaluado experimentalmente.

4.6.2.2 Actualización de funciones en el firmware del Nodo Simulador

Durante la primera etapa del proyecto, se desarrolló un nodo simulador cuya configuración inicial dependía de una conexión física a través del monitor serial del entorno de desarrollo (IDE). Para integrar este dispositivo a la nueva arquitectura de control remoto, se modificó el firmware original escrito en C++ para el microcontrolador ESP32. Se incorporaron librerías de manejo de JSON (ArduinoJson), almacenamiento persistente (EEPROM) y sincronización temporal (NTPClient), resultando en las siguientes mejoras operativas:

4.6.2.2.1 Implementación de la suscripción a tópicos MQTT (configuración, comando y respuesta)

Se programó el firmware para que, al encenderse y establecer conexión Wi-Fi, el nodo simulador se suscriba por defecto al tópico global de configuración: ***sunnyhealth/setting***. Al recibir un payload en formato JSON con los metadatos de inicialización, el microcontrolador procesa la información y ejecuta tres acciones de enrutamiento:

1. Construye y se suscribe a su tópico dinámico de comandos: *sunnyhealth/nombre_dispositivo/id_dispositivo/command*.
2. Construye y se suscribe a un tópico dinámico de salida (*nombre_dispositivo/id_paciente/id_dispositivo/out*) para escuchar los mensajes de confirmación (ACK) o errores emitidos por la API Merry.
3. Se desuscribe automáticamente del tópico global *sunnyhealth/setting*.

Como se justificó en la sección anterior, esta desuscripción aísla al dispositivo, permitiendo que futuras configuraciones de otros nodos en la red no sobrescriban sus parámetros locales.

4.6.2.2.2 Lógica de procesamiento y recepción de instrucciones

El microcontrolador se mantiene a la escucha en su tópico de comandos privado. Cuando recibe una instrucción, extrae el campo *tipo_dato_fisiologico* (codificado mediante una letra, ej., 'T' para temperatura) y verifica la existencia del campo opcional *valor*.

Para permitir la resiliencia del dispositivo frente a cortes de energía, el tipo de dato fisiológico seleccionado se almacena en la memoria no volátil del ESP32 (EEPROM). De este modo, si el dispositivo se reinicia, recupera automáticamente su última configuración operativa.

Posteriormente, la lógica evalúa si el comando incluye un valor personalizado. En caso de no poseerlo, el algoritmo genera un valor aleatorio coherente dentro de rangos clínicos preestablecidos (por ejemplo, 36 °C a 38 °C para temperatura). Si el comando incluye un valor (o dos, en el caso de la presión arterial sistólica y diastólica), el sistema asimila dichas variables.

Finalmente, el nodo construye un tópico dinámico de entrada (*nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in*) y ensambla la estructura del mensaje bajo el estándar HL7 FHIR. Durante este ensamblaje, el ESP32 consulta un servidor de tiempo de red (NTP) para inyectar una marca de tiempo real y precisa (*effectiveDateTime*). Una vez estructurado, el JSON es publicado hacia el broker MQTT.

4.6.2.2.3 Configuración del disparador físico (botón) e indicadores de estado

Para simular el comportamiento de un equipo biomédico real con interfaz de usuario en hardware, se configuró un pulsador físico conectado a los pines del ESP32, acompañado

de un indicador LED para retroalimentación visual. Se programaron dos rutinas operativas basadas en el tiempo de pulsación:

- **Envío manual (Pulsación corta):** Al presionar y soltar el botón, el microcontrolador recupera el tipo de variable fisiológica almacenado en la EEPROM, genera automáticamente un valor aleatorio dentro de los límites clínicos y lo transmite al sistema en formato HL7 FHIR, haciendo parpadear el LED de estado. Esto permite la generación de datos sin depender de la interfaz web.
- **Restablecimiento de fábrica (Pulsación larga):** Se implementó un mecanismo de recuperación ante fallos. Si el usuario mantiene presionado el botón por más de tres segundos, el firmware borra la configuración almacenada en la EEPROM, reinicia las variables de estado y emite cinco parpadeos rápidos en el LED para confirmar el reseteo. Luego de esta acción, el dispositivo vuelve a suscribirse al tópico global de configuración, quedando listo para ser reasignado a un nuevo paciente o entorno.

4.6.3 Integración del Middleware y repositorios al sistema final

Una vez concretada la integración del Nodo Simulador, la fase final consistió en unificar las funciones de almacenamiento de datos (desarrolladas mediante scripts de Python en la Etapa 2) dentro de la arquitectura principal de la API Sunny. Asimismo, dada la estrategia de microservicios adoptada, fue necesario establecer los canales de comunicación HTTP necesarios para que los servicios Sunny y Merry operen de forma sincronizada.

En esta etapa de desarrollo, la API Sunny ya contaba con la capacidad de renderizar la interfaz gráfica web y poseía los endpoints orientados al control remoto del microcontrolador (página Settings). En consecuencia, las funciones incorporadas para consolidar el middleware se centraron en la orquestación interna, las cuales se detallan a continuación:

4.6.3.1 Lógica de comunicación inter-servicios para la configuración de parámetros

Durante el proceso de configuración, la API Sunny no solo publica los metadatos en el broker MQTT, sino que actúa como orquestador al enviar simultáneamente esta misma

carga útil (JSON) hacia la API Merry mediante una petición HTTP POST dirigida al endpoint */suscribir*.

Al recibir estos parámetros (ID de paciente, ID de dispositivo, etc.), el servicio Merry ejecuta una rutina de generación dinámica de rutas. De forma automática, construye y se suscribe a una matriz de seis tópicos de entrada específicos, correspondientes a las variables fisiológicas soportadas por el ecosistema (temperatura, presión arterial, glucemia, frecuencia respiratoria, saturación de oxígeno y frecuencia cardíaca). Esta automatización prepara pasivamente al servicio para interceptar cualquier medición futura proveniente de ese nodo específico, y preconfigura la estructura del tópico de salida (ACK) para mantener la comunicación bidireccional.

4.6.3.2 Lógica de procesamiento bidireccional para la inserción de datos

El ciclo de vida del dato biomédico se completa cuando un mensaje HL7 FHIR ingresa por alguno de los tópicos dinámicos de escucha. La API Merry intercepta la carga útil y la transfiere inmediatamente a la API Sunny a través del endpoint */datos/ingresar* (método POST).

Una vez que Sunny recibe el registro, procede a desempaquetarlo y ejecuta la lógica de almacenamiento descrita en la Etapa 2, intentando registrar la información de manera paralela en PostgreSQL (GNU Health) y MariaDB (base de datos espejo). Un aspecto destacado de esta integración es la implementación de una evaluación global de estado transaccional dentro de la API Sunny:

- Si la inserción es correcta en ambas bases de datos, el sistema retorna un código HTTP 200 (Success).
- Si ocurre un fallo asimétrico (por ejemplo, éxito en MariaDB, pero error en GNU Health), el sistema captura las excepciones y retorna un código HTTP 207 (Multi-Status), indicando un guardado parcial.
- Si la inserción falla en ambos repositorios, se deniega la operación con un código HTTP 400 (Bad Request).

Sunny consolida este diagnóstico en un mensaje JSON detallado y lo devuelve a Merry. Finalmente, Merry extrae los identificadores del tópico original de entrada (*nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in*), transforma la terminación de la ruta y publica el JSON de respuesta en el tópico dinámico de salida del microcontrolador (*nombre_dispositivo/id_paciente/id_dispositivo/out*).

Con la validación de estas rutinas, el middleware de integración quedó completamente unificado al flujo general de información. De este modo, se dio por concluido el desarrollo del ecosistema tecnológico, donde se validó funcionalmente el flujo de interoperabilidad entre los módulos bajo las condiciones experimentales definidas. Cabe añadir que, los códigos desarrollados finales para cada módulo del sistema se presentan en el Anexo 2 del documento.

4.6.4 Pruebas de consistencia de datos de extremo a extremo

Con el propósito de validar la interoperabilidad integral del sistema, desde la generación del dato en el hardware hasta su integración clínica en la HCE de GNU Health, se diseñó un protocolo de pruebas de extremo a extremo (End-to-End). Para una evaluación del sistema, el protocolo se dividió en dos escenarios de validación complementarios.

4.6.4.1 Escenario 1: Análisis de Trazabilidad Transaccional (Prueba Unitaria)

El objetivo de esta primera prueba fue auditar el ciclo de vida transaccional de un único dato fisiológico para verificar el correcto funcionamiento del middleware y el enrutamiento MQTT. Para ello, se configuró el microcontrolador con un solo paciente, un médico tratante y un dispositivo de forma remota, a través de la interfaz web en Streamlit. La validación de este escenario se sustentó en la recolección de evidencia técnica y visual a través de tres niveles de auditoría:

- Auditoría de red y procesamiento: Se analizó el monitor serial del Nodo Simulador (ESP32) y los registros de eventos (logs) en las terminales de ejecución de las APIs Merry y Sunny durante la captura en el hardware, la transformación de protocolos y la emisión del mensaje de confirmación.
- Auditoría de almacenamiento (Bases de datos): Se utilizó el gestor de bases de datos DBeaver para constatar la llegada del formato JSON (basado en el estándar HL7 FHIR) y su correcta inserción estructural en las tablas correspondientes de PostgreSQL (entorno GNU Health) y MariaDB (base de datos espejo).
- Auditoría de visualización (Interfaz web y HCE): Se empleó la plataforma web desarrollada en Streamlit, específicamente las pantallas principales (Home) y de detalles (Details), para observar el dato recibido y auditar su trazabilidad desde la base de datos espejo. En paralelo, se ingresó a la aplicación de escritorio GNU

Health Client para comprobar la integración y correcta visualización del registro clínico dentro de la HCE del paciente configurado.

4.6.4.2 Escenario 2: Evaluación de Consistencia y Segregación de Datos mediante Matriz Combinatoria (N=120)

Al superar la validación transaccional unitaria, la segunda prueba se enfocó en evaluar la consistencia funcional del sistema, la correcta segregación de la información biomédica y la ausencia de pérdidas de tramas bajo un flujo estructurado de datos entre múltiples sujetos en un entorno controlado de red local. Para esta evaluación, se seleccionaron dos perfiles de pacientes previamente registrados en el sistema GNU Health, con el objetivo de verificar el enrutamiento dinámico y descartar la existencia de asignaciones cruzadas de parámetros clínicos en la base de datos de la HCE.

Con el propósito de definir de manera sistemática los escenarios de prueba y evitar una selección arbitraria de las entradas del sistema, se empleó la metodología de Partición del Espacio de Entrada (Input Space Partitioning, ISP) (58), la cual consiste en dividir el dominio de entrada en particiones o bloques, asumiendo que los valores pertenecientes a un mismo bloque presentan un comportamiento funcional equivalente durante la ejecución de las pruebas. A partir del modelo de partición del espacio de entrada se identificaron tres características de entrada independientes para el presente estudio:

1. **Perfiles de Configuración del Nodo Simulador (2 bloques):** Perfil A (Paciente A, Médico A, Dispositivo A) y Perfil B (Paciente B, Médico B, Dispositivo B), diseñados para evaluar la segregación y el mapeo relacional en PostgreSQL.
2. **Variables Fisiológicas (6 bloques):** Procesamiento de los seis parámetros vitales implementados (temperatura, presión arterial, glucemia, frecuencia respiratoria, saturación de oxígeno y frecuencia cardíaca).
3. **Modalidades de transmisión del Payload (2 bloques):** Modalidad Aleatoria y Modalidad Personalizada.

Al definir las particiones del espacio de entrada, se aplicó el criterio de Cobertura por Todas las Combinaciones (All Combinations Coverage, ACoC) descrito por Ammann y Offutt (58), el cual establece que deben evaluarse todas las combinaciones posibles entre los bloques definidos para cada característica. En consecuencia, la combinación completa de los tres factores considerados (2 perfiles $\times$ 6 variables fisiológicas $\times$ 2 modalidades de transmisión) dio lugar a 24 escenarios operativos únicos e independientes, permitiendo la

cobertura de todas las combinaciones definidas para los factores considerados en esta evaluación. La estrategia de evaluar todas las combinaciones definidas para los tres factores de entrada también se encuentra respaldada por el trabajo de Kuhn et al. (62), quienes demostraron, mediante el análisis empírico de fallos en distintos dominios de software, que la mayoría de los defectos son ocasionados por la interacción de un número reducido de parámetros, por lo que las pruebas combinatorias permiten obtener una elevada cobertura de defectos sin necesidad de evaluar exhaustivamente todas las posibles entradas del sistema.

Adicionalmente, con la finalidad de evaluar la repetibilidad y la estabilidad operativa del sistema frente a posibles fluctuaciones transitorias en la red local y durante el establecimiento, mantenimiento y cierre de las conexiones (MQTT, API y PostgreSQL), se estableció un protocolo de cinco ejecuciones consecutivas para cada uno de los 24 escenarios combinatorios. Esta repetición secuencial tuvo como propósito verificar que el nodo simulador, el middleware y la HCE mantuvieran un comportamiento funcional consistente durante transmisiones sucesivas, permitiendo comprobar la ausencia de fallos intermitentes asociados con la comunicación, el procesamiento y el almacenamiento de la información biomédica durante la ejecución repetida de cada escenario en un entorno controlado. De este modo, el protocolo experimental comprendió un total de 120 transmisiones estructuradas ($24 \text{ escenarios} \times 5 \text{ repeticiones}$), correspondientes a 60 transmisiones para cada uno de los dos perfiles de pacientes evaluados.

V. Resultados

5.1 Trazabilidad de un ciclo de transmisión unitario (Prueba Unitaria)

Con el objetivo de demostrar la correcta interoperabilidad y transmisión de los datos fisiológicos entre los distintos módulos del sistema, se procedió a aislar un único evento de transmisión. Esta estrategia permite capturar evidencias en cada etapa del ciclo, demostrando de forma detallada que el flujo de información es íntegro, direccional y trazable.

Para esta prueba de concepto, se seleccionó la transmisión de un dato fisiológico compuesto: la presión arterial, operando bajo la modalidad de envío "personalizado". En este caso, el usuario ingresó manualmente los valores clínicos simulados: 110 mmHg para la presión sistólica y 70 mmHg para la presión diastólica. A continuación, se desglosa el comportamiento del sistema en cada una de las fases necesarias para insertar este registro en la HCE de GNU Health.

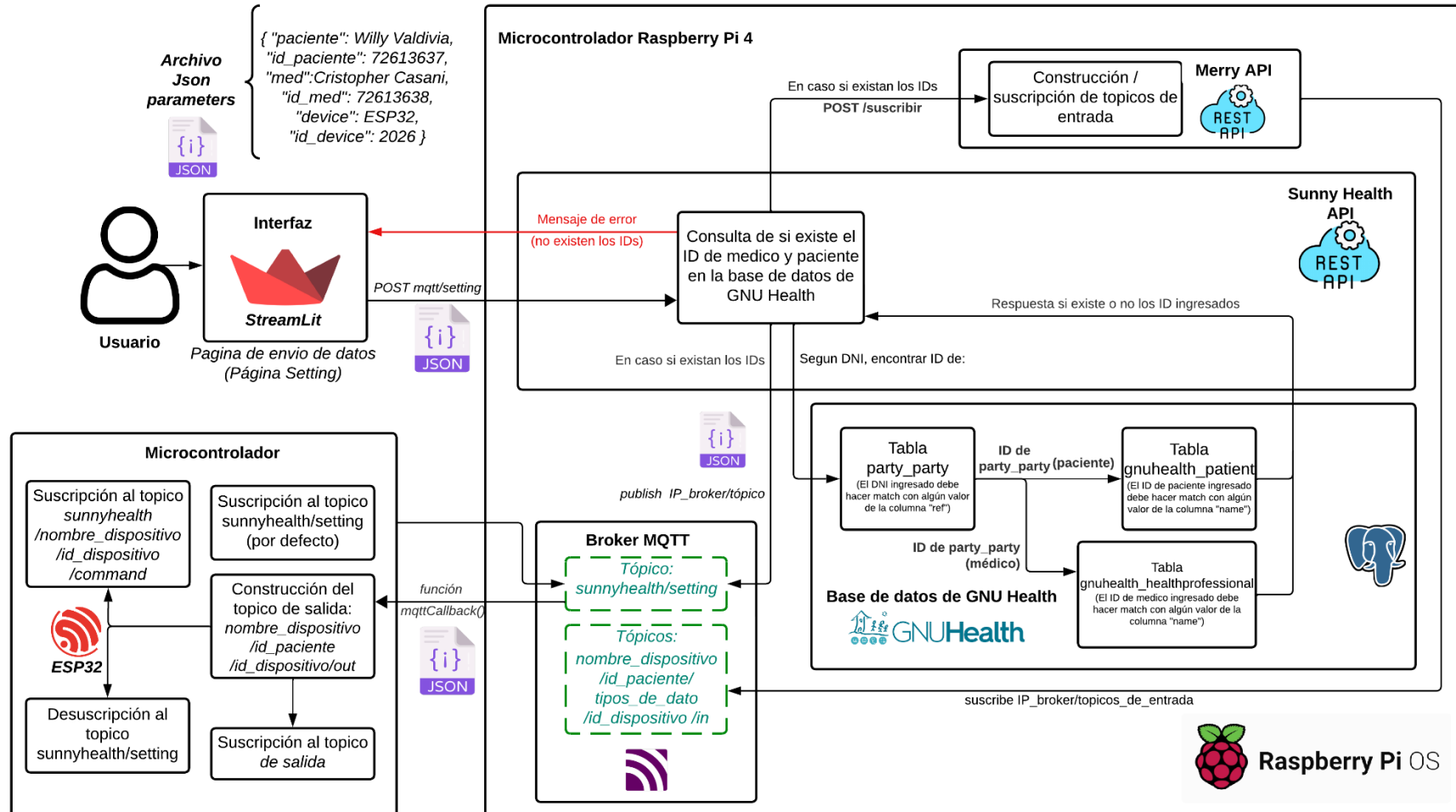
5.1.1 Generación de parámetros y configuración remota

Para la ejecución de este caso de prueba, se definieron los siguientes metadatos de configuración, correspondientes a entidades previamente registradas en el sistema hospitalario:

- Paciente: Willy Valdivia (ID/DNI: 72613637).
- Médico: Cristopher Casani (ID/DNI: 72613638).

Por otro lado, se decidió usar como nombre del dispositivo ESP32, ya que este microcontrolador será el que actuará como nodo simulador y se asoció el código 2026 como su ID único. El comportamiento del sistema durante esta fase de configuración se ilustra en el siguiente diagrama de flujo.

Figura 32. Diagrama de flujo del proceso de configuración remota de parámetros



Fuente: Elaboración propia.

5.1.1.1 Descripción del flujo de configuración

Como detalla el diagrama, el proceso inicia en la interfaz web (Streamlit) dentro de la página Settings. Al presionar el botón de envío, la interfaz empaqueta los metadatos ingresados por el usuario en un archivo JSON y realiza una petición HTTP POST al endpoint */mqtt/setting* de la API principal (Sunny).

Al recibir la carga útil, Sunny ejecuta una rutina de validación contra la base de datos PostgreSQL de GNU Health. Primero, consulta la tabla central *party_party* buscando coincidencias de los DNIs en la columna *ref*. De ser exitosa la búsqueda, extrae los identificadores internos y valida su correspondencia cruzada en las tablas *gnuhealth_patient* y *gnuhealth_healthprofessional* verificando la columna *name*. Si no existe coincidencia, el sistema aborta la operación y retorna un error a la interfaz web.

Luego de una validación, la API Sunny enruta el JSON validado hacia dos destinos simultáneos:

1. Lo publica en el broker MQTT (puerto 1883) bajo el tópico global *sunnyhealth/setting*.
2. Se envía a la API Merry mediante una petición POST al endpoint */suscribir*. Al recibirlo, Merry construye y se suscribe inmediatamente a todos los tópicos dinámicos de entrada posibles para ese nodo, con la estructura:

nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in.

En paralelo, el microcontrolador (Nodo Simulador), que se encuentra a la escucha por defecto en el tópico *sunnyhealth/setting*, recibe el JSON. A partir de estos metadatos, el ESP32 construye y se suscribe a su tópico dinámico de salida (*nombre_dispositivo/id_paciente /id_dispositivo/out*) y a su tópico de comandos (*sunnyhealth/nombre_dispositivo /id_dispositivo/command*), para finalmente desuscribirse del tópico global por defecto, quedando aislado y listo para operar.

5.1.1.2 Evidencia de ejecución de configuración de parámetros

Una vez definido el flujo, se ejecutó la prueba ingresando los parámetros en la interfaz gráfica, tal como se evidencia a continuación en las siguientes figuras:

Figura 33. Captura de pantalla de la configuración de metadatos en la página Settings en Streamlit (Parte 1).

The screenshot shows a web browser window with the URL 'localhost:8501'. On the left is a sidebar with a 'Navegación' section containing buttons for 'Inicio', 'Pacientes', 'Envío de Datos Microcontrolador', and 'Cerrar sesión'. The main content area is titled 'Configuración del paciente, médico y dispositivo'. It contains several input fields: 'Nombre del paciente' (Willy Valdivia), 'ID del paciente' (72613637), 'Nombre del médico' (Cristopher Casani), 'ID del médico' (72613638), 'Nombre del dispositivo' (ESP32), and 'ID del dispositivo' (2026). Below these fields is a red 'Enviar parámetros' button. A green success message at the bottom states '✓ Parámetros configurados correctamente.'

Fuente: Elaboración propia.

Figura 34. Captura de pantalla de la configuración de metadatos en la página Settings en Streamlit (Parte 2).

This screenshot shows the same Streamlit interface as Figure 33, but with the 'Enviar parámetros' button highlighted. Below the button, a green success message reads '✓ Parámetros configurados correctamente.' Below this, a JSON object is displayed, representing the response from the backend. The JSON structure is as follows:

```
{
  "merry_answer": {
    "status": "ok"
  },
  "topics": [
    0: "ESP32/72613637/temperature/2026/in"
    1: "ESP32/72613637/blood_pressure/2026/in"
    2: "ESP32/72613637/glycemia/2026/in"
    3: "ESP32/72613637/respiratory_rate/2026/in"
    4: "ESP32/72613637/oxygen_saturation/2026/in"
    5: "ESP32/72613637/heart_rate/2026/in"
  ]
},
  "message": "Parámetros enviados correctamente a Merry API"
  "mqtt": "Parametros publicados en topic: sunnyhealth/setting"
}
```

Fuente: Elaboración propia.

Como respuesta a esta acción del usuario, los servicios backend reaccionaron según la arquitectura diseñada. Las siguientes capturas de las terminales de ejecución de Python

demuestran la recepción de la petición POST /mqtt/setting por parte de Sunny, y la petición POST /suscribir procesada por Merry, evidenciando la suscripción automática a los seis tópicos dinámicos de entrada:

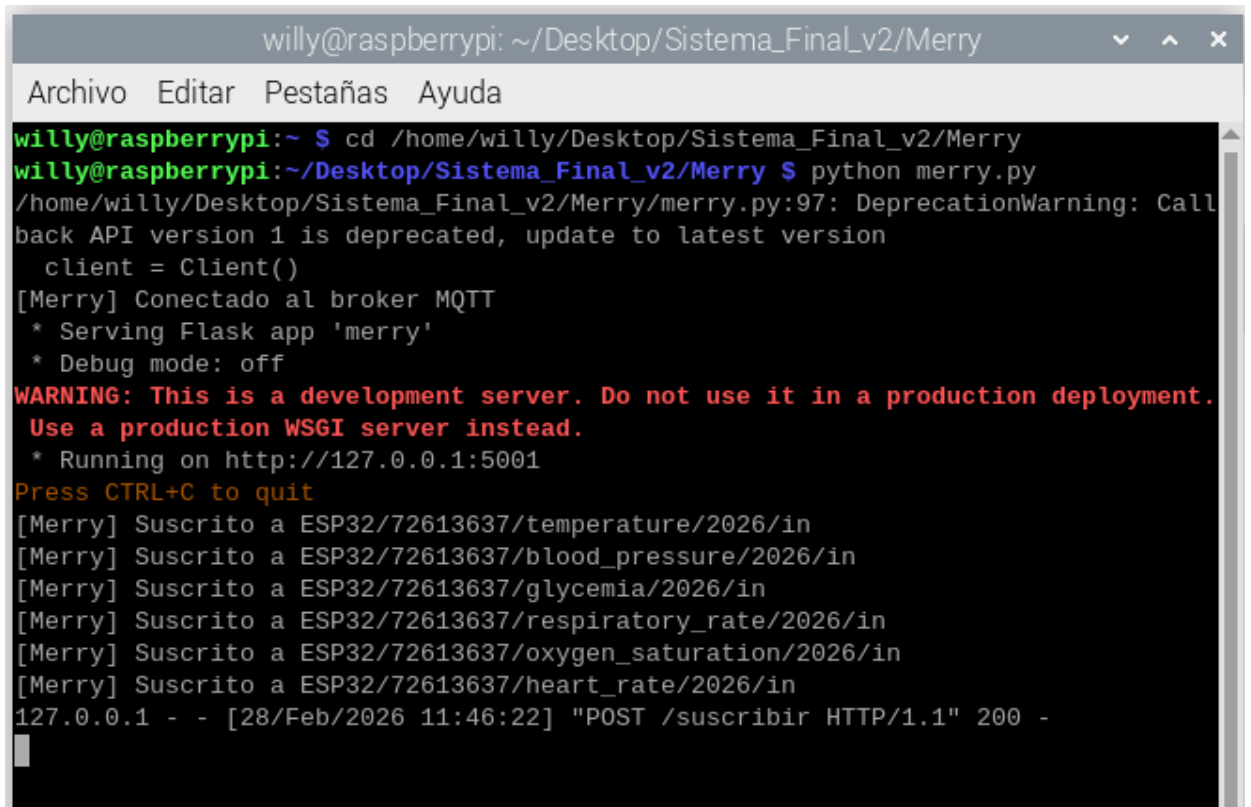
Figura 35. Registro de ejecución (logs) en la terminal de la API principal Sunny Health.

```
willy@raspberrypi: ~/Desktop/Sistema_Final_v2/API
Archivo Editar Pestañas Ayuda
willy@raspberrypi:~$ cd /home/willy/Desktop/Sistema_Final_v2/API
willy@raspberrypi:~/Desktop/Sistema_Final_v2/API$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/exec_code.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/exec_code.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /users/willy HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:46:22] "POST /mqtt/setting HTTP/1.1" 200 -
```

Fuente: Elaboración propia.

Figura 36. Registro de ejecución (logs) y suscripción dinámica en la terminal de la API

Merry

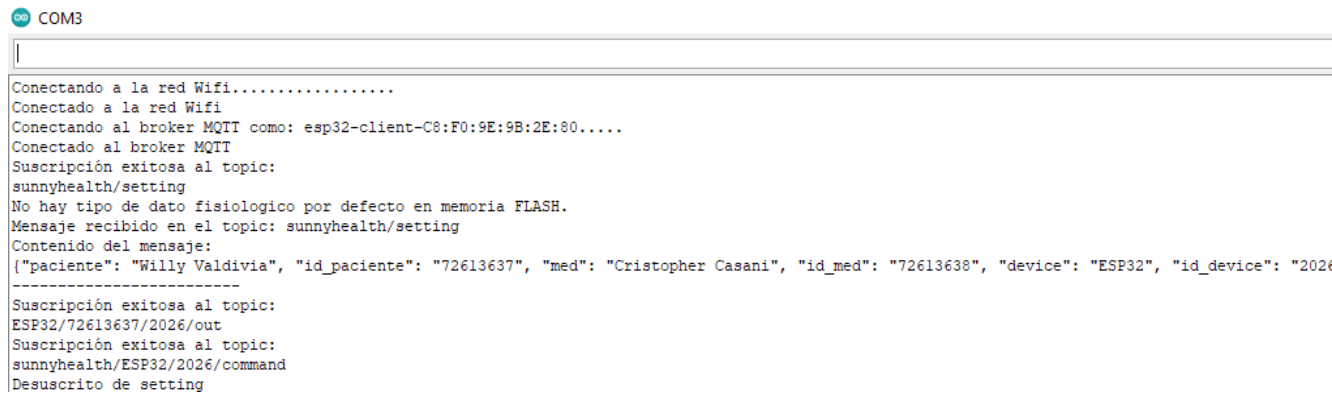


```
willy@raspberrypi: ~/Desktop/Sistema_Final_v2/Merry
Archivo Editar Pestañas Ayuda
willy@raspberrypi:~ $ cd /home/willy/Desktop/Sistema_Final_v2/Merry
willy@raspberrypi:~/Desktop/Sistema_Final_v2/Merry $ python merry.py
/home/willy/Desktop/Sistema_Final_v2/Merry/merry.py:97: DeprecationWarning: Call
back API version 1 is deprecated, update to latest version
  client = Client()
[Merry] Conectado al broker MQTT
* Serving Flask app 'merry'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5001
Press CTRL+C to quit
[Merry] Suscrito a ESP32/72613637/temperature/2026/in
[Merry] Suscrito a ESP32/72613637/blood_pressure/2026/in
[Merry] Suscrito a ESP32/72613637/glycemia/2026/in
[Merry] Suscrito a ESP32/72613637/respiratory_rate/2026/in
[Merry] Suscrito a ESP32/72613637/oxygen_saturation/2026/in
[Merry] Suscrito a ESP32/72613637/heart_rate/2026/in
127.0.0.1 - - [28/Feb/2026 11:46:22] "POST /suscribir HTTP/1.1" 200 -
```

Fuente: Elaboración propia.

Finalmente, la integración a nivel de hardware se comprobó mediante el monitor serial del entorno de desarrollo Arduino IDE. La Figura 37 demuestra la secuencia interna del microcontrolador ESP32: su conexión exitosa al broker MQTT, la intercepción del JSON en el tópico global, la suscripción a sus nuevos tópicos dinámicos (out y command) y su posterior desuscripción del canal de configuración. Adicionalmente, se observa que la variable de tipo de dato fisiológico alojada en la memoria flash (EEPROM) se encuentra vacía, confirmando que el dispositivo está a la espera de su primera instrucción.

Figura 37. Secuencia de inicialización y enrutamiento dinámico en el monitor serial del ESP32



```
COM3
Conectando a la red Wifi.....
Conectado a la red Wifi
Conectando al broker MQTT como: esp32-client-C8:F0:9E:9B:2E:80.....
Conectado al broker MQTT
Suscripción exitosa al topic:
sunnyhealth/setting
No hay tipo de dato fisiologico por defecto en memoria FLASH.
Mensaje recibido en el topic: sunnyhealth/setting
Contenido del mensaje:
{"paciente": "Willy Valdivia", "id_paciente": "72613637", "med": "Cristopher Casani", "id_med": "72613638", "device": "ESP32", "id_device": "202613637"}
Suscripción exitosa al topic:
ESP32/72613637/2026/out
Suscripción exitosa al topic:
sunnyhealth/ESP32/2026/command
Desuscrito de setting
```

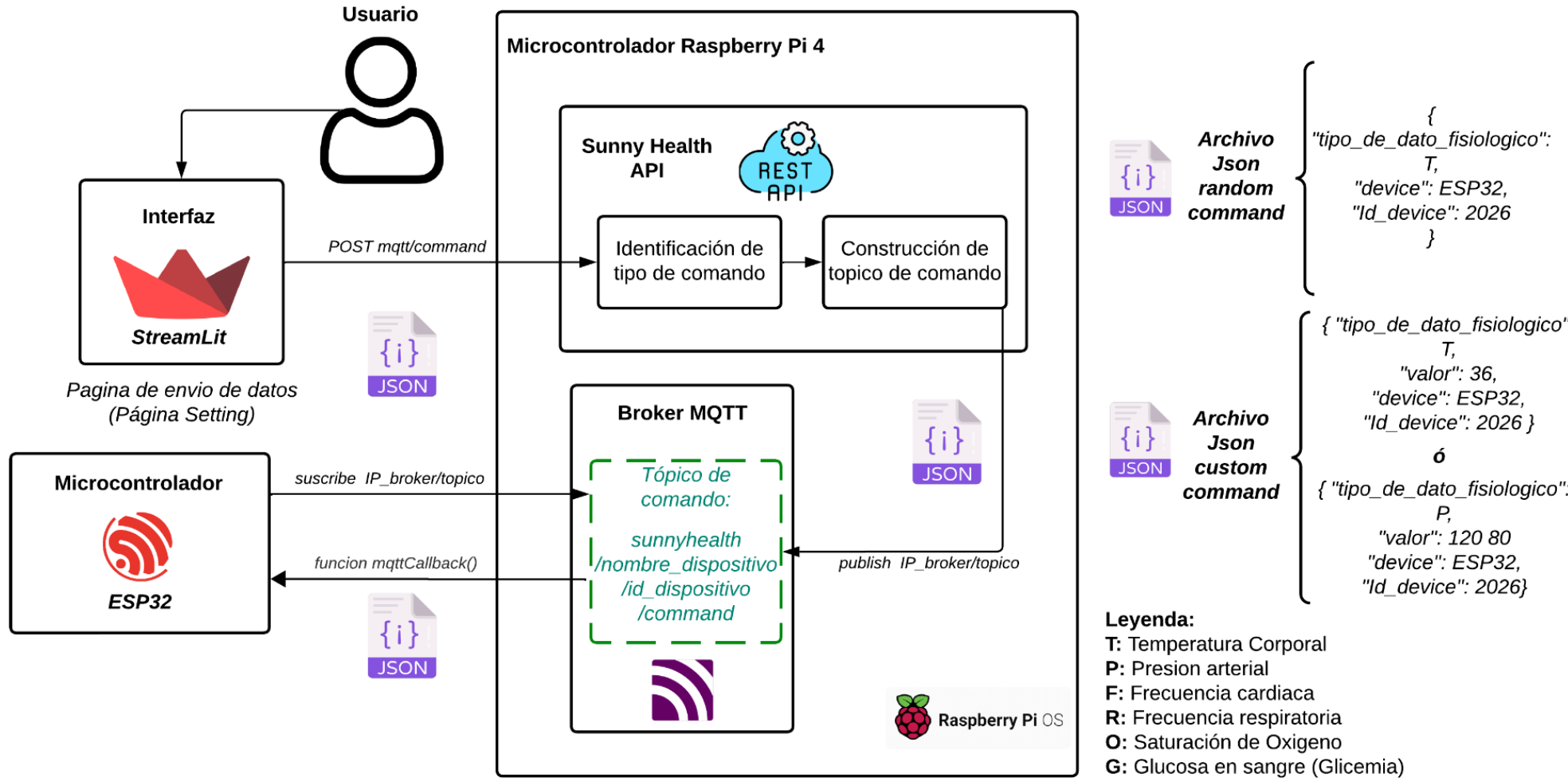
Fuente: Elaboración propia.

De esta manera, se demuestra de forma trazable el flujo de configuración remota, confirmando la sincronización de extremo a extremo entre la interfaz web, el middleware y el firmware del nodo simulador.

5.1.2 Captura del comando desde el Nodo Simulador

Una vez que el Nodo Simulador se encuentra configurado y aislado en su canal de comunicación, el sistema está listo para recibir instrucciones de operación. A continuación, se presenta el diagrama de flujo que ilustra la interacción entre los módulos durante la emisión de un comando hacia el ESP32:

Figura 38. Diagrama de flujo del proceso de envío y enrutamiento de comandos al Nodo Simulador.



Fuente: Elaboración propia

5.1.2.1 Descripción del flujo de enrutamiento de comandos

De acuerdo con el diagrama ilustrado, el proceso se origina en la página Settings de la interfaz web (Streamlit). En esta etapa, el usuario determina el tipo de variable fisiológica que el Nodo Simulador debe procesar. Para ello, el frontend estructura un archivo JSON que contiene el código de la variable, el nombre del dispositivo y su identificador (estos dos últimos previamente definidos en la fase de configuración).

El sistema admite dos modalidades de comando: envío de comando aleatorio (sin valor clínico adjunto) y envío de comando personalizado (con un valor clínico específico definido por el usuario). Al interactuar con el botón de envío en cualquiera de las modalidades, la interfaz ejecuta una petición HTTP POST dirigida al endpoint */mqtt/command* de la API principal (Sunny), adjuntando la carga útil.

Al interceptar este JSON, la API Sunny identifica los parámetros y ensambla dinámicamente el tópico de comandos exclusivo del dispositivo objetivo. Dicho tópico adopta la nomenclatura jerárquica: *sunnyhealth/nombre_dispositivo/id_dispositivo/command*. Inmediatamente, la API publica el comando a través del broker MQTT en dicho canal. Dado que el microcontrolador fue programado en la etapa anterior para escuchar exclusivamente esta ruta, recibe la instrucción a través del tópico asignado al dispositivo.

5.1.2.2 Evidencia de ejecución de captura de comando desde el Nodo simulador

Para dar continuidad al caso de prueba, se procedió a ejecutar la modalidad de envío "personalizado". A través de la interfaz web, se seleccionó la variable "Presión Arterial" y se ingresaron los valores clínicos establecidos para esta prueba (110 y 70), separados por un espacio, tal como exige la lógica del sistema para datos compuestos.

Figura 39. Captura de pantalla del envío de comando personalizado en la página Settings de Streamlit.

The screenshot shows a web browser window with the address bar displaying 'localhost:8501'. The page is titled 'Enviar dato fisiológico con valor ingresado'. On the left, there is a sidebar with the heading 'Navegación' and four buttons: 'Inicio', 'Pacientes', 'Envío de Datos Microcontrolador', and 'Cerrar sesión'. The main content area has a title 'Enviar dato fisiológico con valor ingresado' and a dropdown menu for 'Tipo de dato a ingresar con valor' set to 'Presión arterial'. Below this is a text input field containing '110 70'. A button labeled 'Enviar dato personalizado' is visible. A green success message states 'Dato con valor enviado correctamente.' Below this, a JSON object is displayed:

```
{  "contenido_enviado": {    "tipo_dato_fisiológico": "p",    "valor": "110 70"  },  "message": "Comando enviado correctamente al topic segun configuracion"}
```

 At the bottom, there is a button labeled 'Volver a la página principal'.

Fuente. Elaboración propia

Como se visualiza en la Figura 39, la interfaz notifica al usuario el éxito de la operación. De forma concurrente, en la Figura 40, el registro de eventos (log) en la terminal de la API Sunny confirma que el endpoint `/mqtt/command` fue accionado y procesado correctamente:

Figura 40. Registro de ejecución en la terminal de la API Sunny durante el enrutamiento del comando.

```
willy@raspberrypi: ~/Desktop/Sistema_Final_v2/API
Archivo Editar Pestañas Ayuda
willy@raspberrypi:~$ cd /home/willy/Desktop/Sistema_Final_v2/API
willy@raspberrypi:~/Desktop/Sistema_Final_v2/API$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/exec_code.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/exec_code.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /users/willy HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:46:22] "POST /mqtt/setting HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:49:30] "POST /mqtt/command HTTP/1.1" 200 -
```

Fuente. Elaboración propia

Finalmente, la validación a nivel de hardware se constata en el monitor serial del ESP32. La Figura 41 evidencia el momento exacto en el que el comando ingresa por el tópico dinámico correspondiente. Se observa que la estructura del canal de recepción coincide con la arquitectura diseñada y que la carga útil (payload) preserva íntegramente los valores (110 y 70) enviados desde la interfaz web.

Figura 41. Recepción del comando personalizado en el monitor serial del ESP32

```
COM3
Conectando a la red Wifi.....
Conectado a la red Wifi
Conectando al broker MQTT como: esp32-client-C8:F0:9E:9B:2E:80.....
Conectado al broker MQTT
Suscripción exitosa al topic:
sunnyhealth/setting
No hay tipo de dato fisiologico por defecto en memoria FLASH.
Mensaje recibido en el topic: sunnyhealth/setting
Contenido del mensaje:
{"paciente": "Willy Valdivia", "id_paciente": "72613637", "med": "Cristopher Casani", "id_med": "72613638", "device": "ESP32", "id_device": "2026"}
Suscripción exitosa al topic:
ESP32/72613637/2026/out
Suscripción exitosa al topic:
sunnyhealth/ESP32/2026/command
Desuscripto de setting
Mensaje recibido en el topic: sunnyhealth/ESP32/2026/command
Contenido del mensaje:
{"tipo_dato_fisiologico": "P", "valor": "110 70"}
-----
```

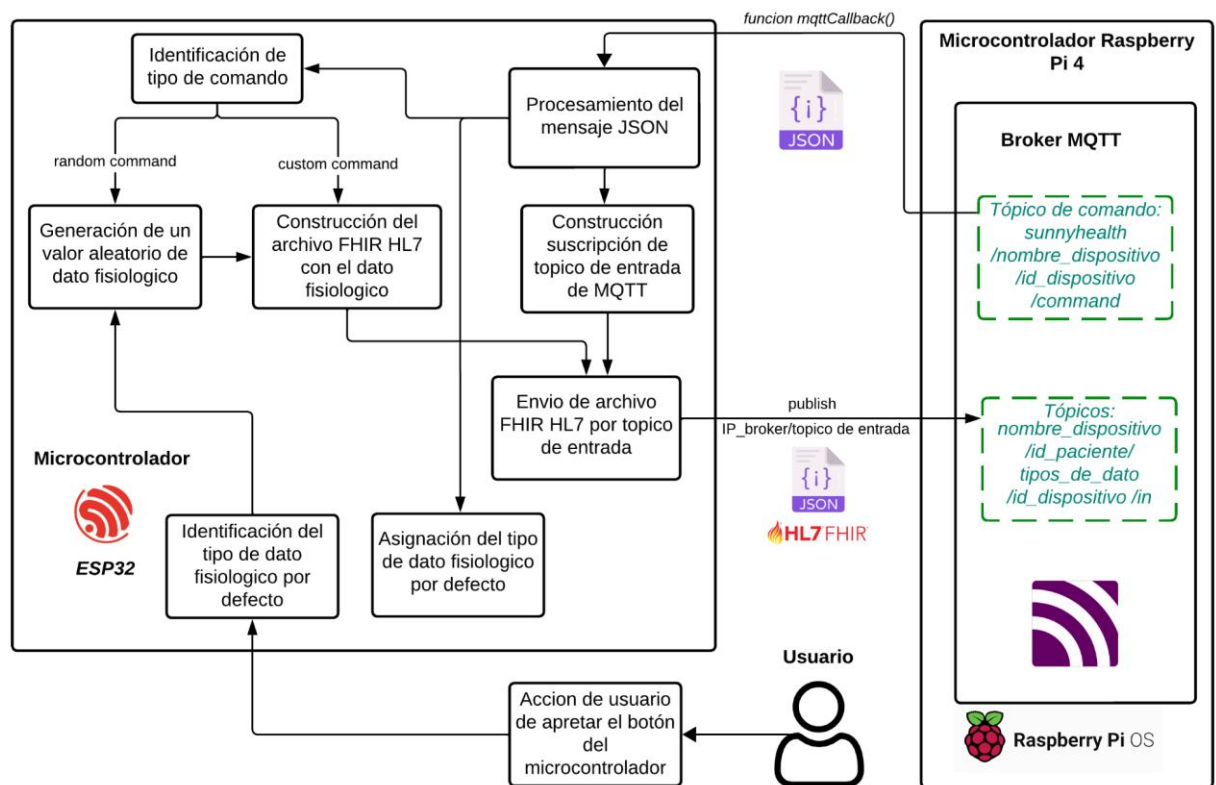
Fuente. Elaboración propia

Mediante esta secuencia de evidencias, queda validado el proceso de comunicación descendente (desde la nube hacia el hardware), permitiendo que las instrucciones emitidas desde la interfaz lleguen de forma precisa al Nodo Simulador.

5.1.3 Generación de datos fisiológicos desde el Nodo Simulador

Luego de la recepción y decodificación del archivo JSON con el comando correspondiente, el microcontrolador inicia el ciclo de generación y estructuración del dato fisiológico. A continuación, se presenta el diagrama de flujo que modela el comportamiento interno del simulador durante esta etapa:

Figura 42. Diagrama de flujo del proceso de generación y estructuración de datos fisiológicos en el Nodo Simulador.



Fuente: Elaboración propia

5.1.3.1 Descripción del flujo de generación de datos

El Nodo Simulador (ESP32) se mantiene a la escucha activa del tópico de comandos exclusivo, definido durante el ciclo de configuración. Al detectar el ingreso de un nuevo JSON, el firmware extrae los metadatos y construye dinámicamente el canal de

transmisión de destino. Este tópico de entrada MQTT adopta la siguiente forma: *nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in*.

Simultáneamente, el algoritmo identifica la modalidad del comando recibido. Si se trata de un comando aleatorio (sin un valor clínico adjunto), el dispositivo ejecuta una función de generación de números pseudoaleatorios, acotada dentro de los límites fisiológicos normales correspondientes al parámetro solicitado. Por el contrario, si se trata de un comando personalizado, el algoritmo extrae y asimila el valor exacto enviado por el usuario.

Con el valor clínico definido, el ESP32 procede a ensamblar la carga útil (payload) bajo la estructura del estándar de interoperabilidad clínica HL7 FHIR. Este archivo JSON resultante encapsula el valor fisiológico, la unidad de medida, los metadatos de configuración (paciente, dispositivo, médico) y una marca de tiempo real obtenida mediante sincronización NTP. Una vez estructurado, el mensaje es publicado en el broker MQTT a través del tópico de entrada previamente construido.

Un valor agregado en el diseño del firmware es su capacidad de almacenamiento de estado. Al procesar un comando, el ESP32 guarda el tipo de variable fisiológica solicitada en su memoria no volátil (EEPROM), estableciéndola como la "variable por defecto".

En base a esta funcionalidad, el dispositivo queda configurado para operar de forma autónoma. Al presionar físicamente el botón de envío integrado en el hardware del ESP32, el sistema recupera la variable por defecto de la memoria, genera un nuevo valor aleatorio, estructura el mensaje HL7 FHIR y lo transmite automáticamente. Esta funcionalidad emula el comportamiento de un equipo biomédico real y permite las transmisiones sucesivas sin requerir nuevas instrucciones desde la interfaz web. Adicionalmente, se implementó un mecanismo de restablecimiento seguro: si el usuario mantiene presionado el botón físico por más de tres segundos, la variable por defecto se borra de la memoria, reiniciando el estado operativo del nodo.

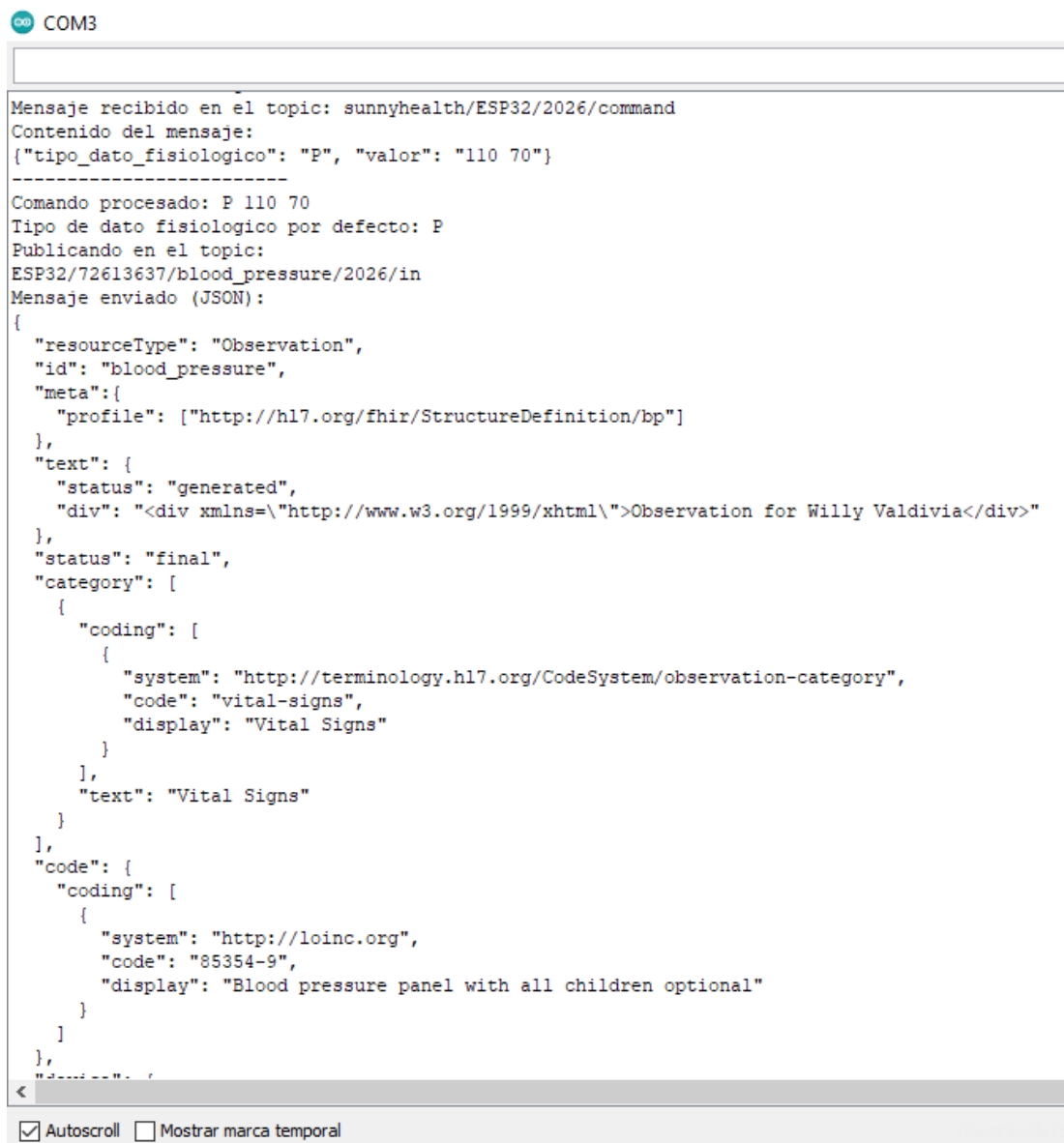
5.1.3.2 Evidencia de ejecución de generación de datos desde el Nodo simulador

Para evidenciar el comportamiento interno del microcontrolador ESP32 durante la asimilación y ejecución del comando personalizado de presión arterial (110/70 mmHg), se extrajeron los registros del monitor serial. Dada la extensión y complejidad de la

estructura del estándar HL7 FHIR, la evidencia gráfica se presenta dividida en tres segmentos secuenciales.

La Figura 43 ilustra el momento exacto en que el ESP32 procesa el comando recibido y almacena la variable correspondiente (Presión Arterial) en la memoria no volátil (EEPROM). Asimismo, se evidencia la construcción dinámica del tópico MQTT de entrada definido para este caso y el inicio del ensamblaje del documento JSON, destacando la correcta asignación del código estandarizado LOINC asociado a esta variable clínica.

Figura 43. Procesamiento del comando, asignación en memoria y apertura de estructura JSON en el monitor serial (Parte 1).



```
COM3

Mensaje recibido en el topic: sunnyhealth/ESP32/2026/command
Contenido del mensaje:
{"tipo_dato_fisiologico": "P", "valor": "110 70"}
-----
Comando procesado: P 110 70
Tipo de dato fisiologico por defecto: P
Publicando en el topic:
ESP32/72613637/blood_pressure/2026/in
Mensaje enviado (JSON):
{
  "resourceType": "Observation",
  "id": "blood_pressure",
  "meta": {
    "profile": ["http://hl7.org/fhir/StructureDefinition/bp"]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\\"http://www.w3.org/1999/xhtml\\">Observation for Willy Valdivia</div>"
  },
  "status": "final",
  "category": [
    {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs",
          "display": "Vital Signs"
        }
      ],
      "text": "Vital Signs"
    }
  ],
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "85354-9",
        "display": "Blood pressure panel with all children optional"
      }
    ]
  },
  "-----"
}
```

☒ Autoscroll ☐ Mostrar marca temporal

Fuente: Elaboración propia

Por otro lado, la Figura 44 constata la integración de los metadatos correspondientes a las entidades vinculadas (identificadores del paciente, médico tratante y dispositivo). Un aspecto que se observa en esta sección es la inyección de la marca de tiempo real, obtenida por sincronización NTP, dentro del atributo `effectiveDateTime`, lo cual permite registrar la trazabilidad cronológica del evento médico.

Figura 44. Inserción de metadatos clínicos y marca de tiempo en el monitor serial del ESP32 (Parte 2).



```

"display": "Blood pressure panel with all children optional"
  }
},
"device": {
  "reference": "Device/2026",
  "display": "ESP32"
},
"subject": {
  "reference": "Patient/72613637",
  "display": "Willy Valdivia"
},
"performer": [
  {
    "reference": "Performer/72613638",
    "display": "Cristopher Casani"
  }
],
"effectiveDateTime": "2026-02-28T11:49:29-05:00",
"component": [
  {
    "code": {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "8480-6",
          "display": "Systolic blood pressure"
        }
      ]
    },
    "valueQuantity": {
      "value": 110.00,
      "unit": "mmHg",
      "system": "http://unitsofmeasure.org",
      "code": "mm[Hg]"
    }
  },
  {
    "code": {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "8480-6",
          "display": "Systolic blood pressure"
        }
      ]
    }
  }
]

```

☒ Autoscroll ☐ Mostrar marca temporal

Fuente: Elaboración propia

Finalmente, la Figura 45 expone el núcleo de la carga útil (payload) bajo el estándar HL7 FHIR. En esta captura se demuestra cómo los valores clínicos ingresados desde la interfaz web (presión sistólica: 110 y presión diastólica: 70) son insertados con precisión en sus respectivos componentes anidados dentro de la estructura JSON, acompañados de sus unidades de medida estandarizadas (mmHg).

Figura 45. Ensamblaje de valores clínicos compuestos y cierre del mensaje HL7 FHIR
(Parte 3).

COM3

```

{
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "8480-6",
        "display": "Systolic blood pressure"
      }
    ]
  },
  "valueQuantity": {
    "value": 110.00,
    "unit": "mmHg",
    "system": "http://unitsofmeasure.org",
    "code": "mm[Hg]"
  }
},
{
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "8462-4",
        "display": "Diastolic blood pressure"
      }
    ]
  },
  "valueQuantity": {
    "value": 70.00,
    "unit": "mmHg",
    "system": "http://unitsofmeasure.org",
    "code": "mm[Hg]"
  }
}
]
}

```

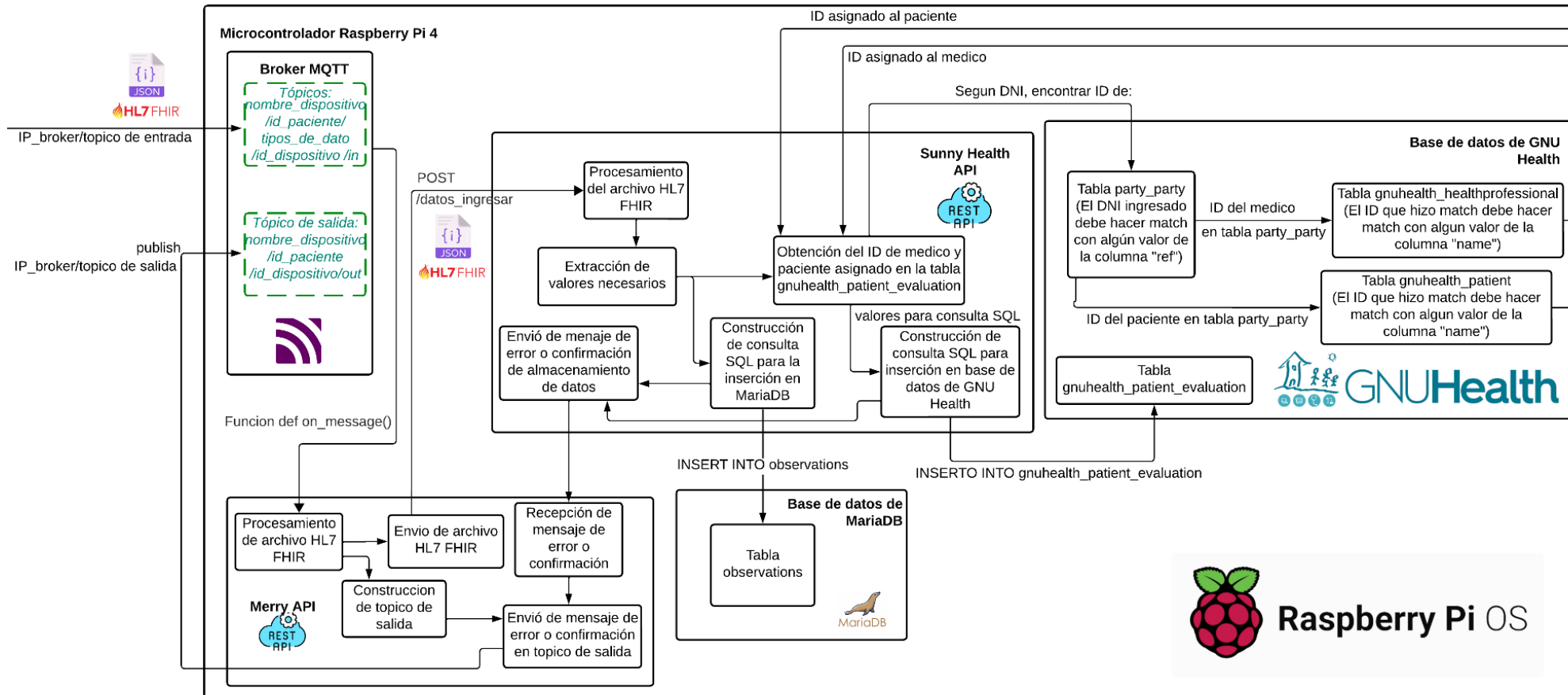
Fuente: Elaboración propia

Mediante esta secuencia de capturas, se valida la capacidad computacional del Nodo Simulador no solo para procesar comandos remotos y gestionar estados de memoria persistente a nivel de hardware, sino también para generar y transmitir conjuntos de datos altamente estructurados e interoperables.

5.1.4 Enrutamiento y procesamiento en el Middleware (API Sunny y API Merry)

Posterior a la publicación del mensaje JSON en formato HL7 FHIR a través del tópico dinámico de entrada, el sistema avanza al siguiente ciclo del flujo de información. En esta etapa, el middleware (compuesto por las APIs Merry y Sunny) asume la responsabilidad de recibir, deserializar, procesar y persistir la información clínica en los repositorios de datos (PostgreSQL para GNU Health y MariaDB como base de datos espejo). A continuación, se presenta el diagrama de flujo que modela la lógica de procesamiento transaccional en esta capa:

Figura 46. Diagrama de flujo del proceso de enrutamiento y procesamiento en el Middleware.



Fuente: Elaboración propia.

5.1.4.1 Descripción del flujo de enrutamiento y procesamiento en el Middleware

Como se estableció en el primer ciclo de transmisión, la API Merry opera como un servicio de escucha activa, suscrito a todos los tópicos dinámicos de entrada generados para el Nodo Simulador. Al ingresar un mensaje FHIR en el broker MQTT, Merry lo intercepta, extrae la carga útil (payload) y ejecuta una transición de protocolo: reenvía el mensaje completo hacia la API principal (Sunny) mediante una petición HTTP POST dirigida al endpoint */datos/ingresar*. Simultáneamente, Merry analiza la estructura jerárquica del tópico de origen (nombre_dispositivo/id_paciente/tipo_de_dato/id_dispositivo/in), reconstruye el tópico de salida correspondiente (nombre_dispositivo/id_paciente/id_dispositivo/out) y queda a la espera del diagnóstico transaccional por parte de Sunny.

Al recibir el JSON clínico, la API Sunny inicia un proceso de extracción (deserialización) de los metadatos clave: identificadores del paciente, médico, dispositivo y la marca de tiempo original (effectiveDateTime). Debido a las divergencias estructurales y de propósito entre GNU Health y MariaDB, la API ejecuta dos rutinas de persistencia independientes y adaptadas a cada motor de base de datos:

5.1.4.1.1 Almacenamiento en la base de datos espejo (MariaDB)

Para el almacenamiento en la tabla observations de la base de datos health_monitoring, el algoritmo extrae los valores planos y sus unidades de medida. Un aspecto a destacar del código desarrollado es su capacidad para identificar el tipo de dato; en el caso de la presión arterial, detecta que es una variable compuesta y extrae por separado la presión sistólica (value) y diastólica (value2). Asimismo, la API inyecta el mensaje HL7 FHIR íntegro en formato de texto dentro de la columna full_message como respaldo de auditoría.

Cabe resaltar que la inserción SQL omite explícitamente el parámetro effectiveDateTime. Esto obedece a una decisión de diseño: esta base de datos soporta la interfaz de monitoreo en tiempo real, por lo que la tabla está configurada para autogenerar un timestamp del sistema en el momento exacto de la inserción, privilegiando el tiempo de llegada sobre el tiempo de medición.

5.1.4.1.2 Almacenamiento en el sistema hospitalario (GNU Health / PostgreSQL)

Para insertar el registro en el entorno oficial, Sunny implementa una lógica más estricta. Primero, cruza los identificadores (DNI) contra la tabla maestra `party_party` para obtener los IDs relacionales internos de las tablas `gnuhealth_patient` y `gnuhealth_healthprofessional`.

Posteriormente, ejecuta una estandarización temporal crítica: captura el parámetro `effectiveDateTime` (formato ISO 8601) y, mediante la librería `pytz`, lo convierte desde la zona horaria local (America/Lima) al Tiempo Universal Coordinado (UTC), permitiendo la compatibilidad nativa con la arquitectura del ERP Tryton/GNU Health. Finalmente, mapea el valor clínico hacia la columna específica de la variable en la tabla `gnuhealth_patient_evaluation` (ej. Para el caso de prueba, *systolic* y *diastolic*) y ejecuta la inserción SQL.

Finalizadas ambas rutinas, Sunny ejecuta una evaluación transaccional global. Si ambas inserciones son correctas, genera un código HTTP 200; si una falla, pero la otra se concreta, emite un código HTTP 207 (Multi-Status); y si ambas fallan, retorna un error 400. Este diagnóstico detallado (incluyendo los errores específicos de cada motor de base de datos) es devuelto como respuesta HTTP a Merry.

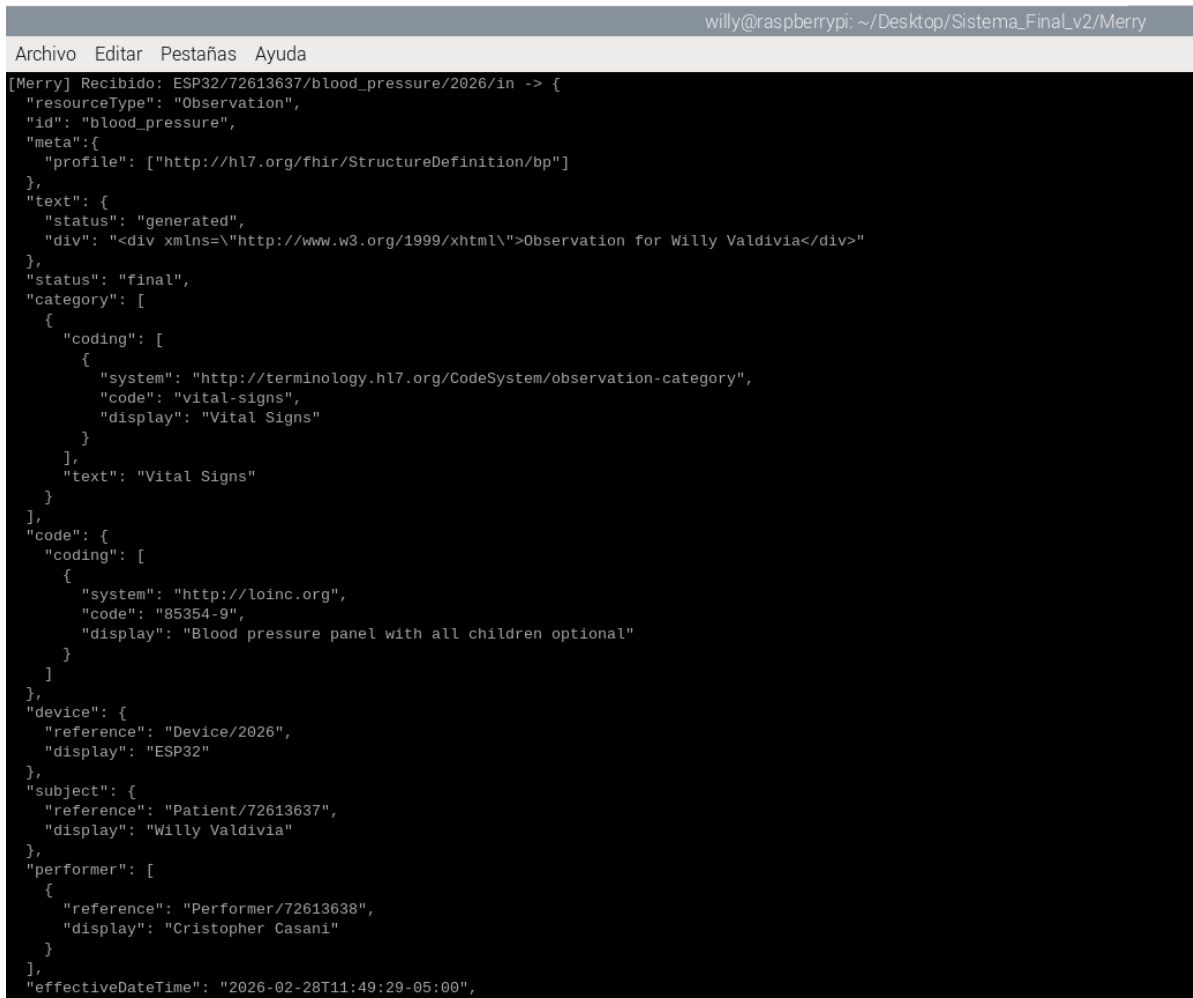
Por otro lado, Merry encapsula esta respuesta en un nuevo JSON, le adhiere una marca de tiempo local y lo publica en el broker MQTT a través del tópico de salida, cerrando el ciclo de comunicación y notificando al Nodo Simulador el estado final de su transmisión.

5.1.4.2 Evidencia de ejecución de enrutamiento y procesamiento de datos

Para constatar empíricamente el flujo de procesamiento transaccional en el middleware, se analizaron los registros de eventos de los servicios involucrados durante el envío del dato de presión arterial.

En primer lugar, la Figura 47 expone la terminal de la API Merry en el instante en que intercepta el mensaje publicado por el microcontrolador. En dicha captura se constata la recepción del documento HL7 FHIR a través del tópico dinámico de entrada asociado, evidenciando la capacidad de escucha activa de este servicio.

Figura 47. Registro de ejecución en la terminal de la API Merry tras la intercepción del mensaje en el tópico de entrada



```
willy@raspberrypi: ~/Desktop/Sistema_Final_v2/Merry
Archivo Editar Pestañas Ayuda
[Merry] Recibido: ESP32/72613637/blood_pressure/2026/in -> {
  "resourceType": "Observation",
  "id": "blood_pressure",
  "meta": {
    "profile": ["http://hl7.org/fhir/StructureDefinition/bp"]
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns='http://www.w3.org/1999/xhtml'>Observation for Willy Valdivia</div>"
  },
  "status": "final",
  "category": [
    {
      "coding": [
        {
          "system": "http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs",
          "display": "Vital Signs"
        }
      ],
      "text": "Vital Signs"
    }
  ],
  "code": {
    "coding": [
      {
        "system": "http://loinc.org",
        "code": "85354-9",
        "display": "Blood pressure panel with all children optional"
      }
    ]
  },
  "device": {
    "reference": "Device/2026",
    "display": "ESP32"
  },
  "subject": {
    "reference": "Patient/72613637",
    "display": "Willy Valdivia"
  },
  "performer": [
    {
      "reference": "Performer/72613638",
      "display": "Cristopher Casani"
    }
  ],
  "effectiveDateTime": "2026-02-28T11:49:29-05:00",
  "effectivePeriod": null
}
```

Fuente. Elaboración propia

Una vez que Merry reenvía la carga útil, la API principal asume el control del flujo. La Figura 48 muestra la terminal de Sunny, evidenciando la recepción exitosa de la petición HTTP POST en el endpoint /datos/ingresar. Este registro confirma que el backend procesó la información y ejecutó las rutinas de inserción en las bases de datos correspondientes (MariaDB y PostgreSQL).

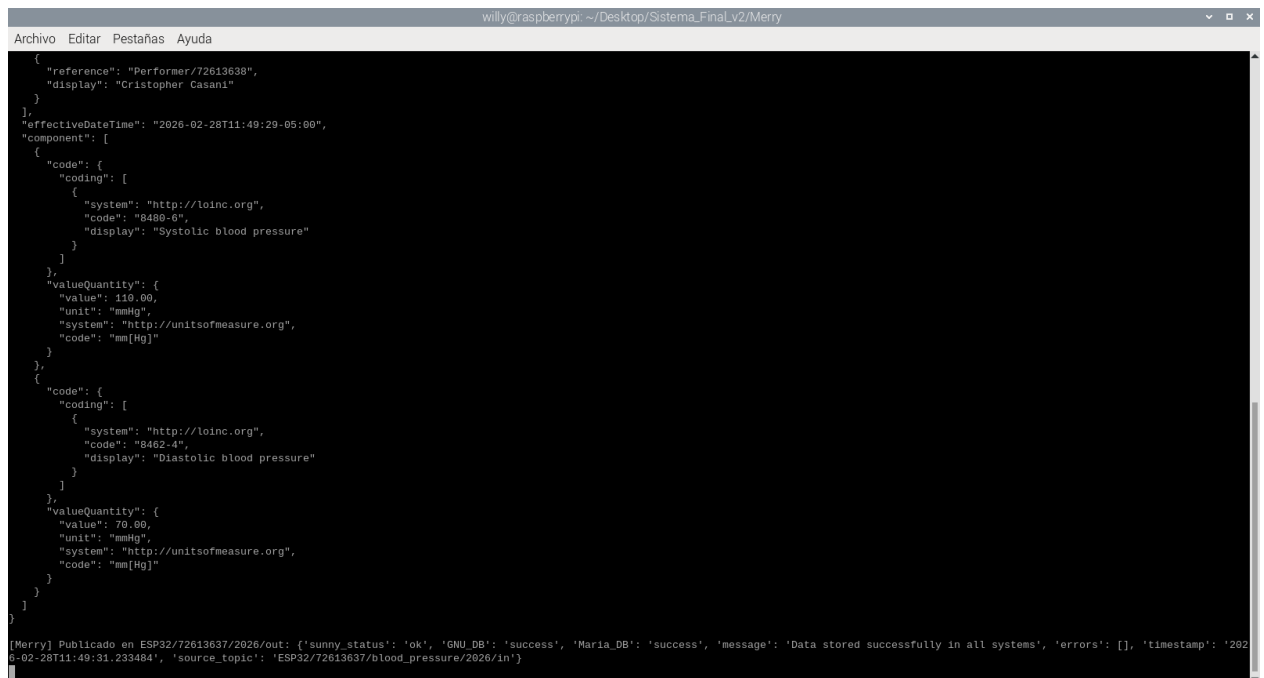
Figura 48. Registro de ejecución en la terminal de la API Sunny confirmando el procesamiento de los repositorios de datos.

```
willy@raspberrypi: ~/Desktop/Sistema_Final_v2/API
Archivo Editar Pestañas Ayuda
willy@raspberrypi:~$ cd /home/willy/Desktop/Sistema_Final_v2/API
willy@raspberrypi:~/Desktop/Sistema_Final_v2/API$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/env_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/delta_generator.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/elements/heading.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/metrics_util.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py', reloading
* Detected change in '/home/willy/.local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/exec_code.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Detected change in '/usr/lib/python3.11/threading.py', reloading
* Restarting with watchdog (inotify)
* Debugger is active!
* Debugger PIN: 291-618-858
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /users/willy HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:45:17] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:46:22] "POST /mqtt/setting HTTP/1.1" 200 -
127.0.0.1 - - [28/Feb/2026 11:49:30] "POST /mqtt/command HTTP/1.1" 200 -
[Sunny] Recibido desde Merry: ESP32/72613637/blood_pressure/2026/in
[Sunny] Datos insertados en GNU Health correctamente.
[Sunny] Datos insertados en MariaDB correctamente.
127.0.0.1 - - [28/Feb/2026 11:49:31] "POST /datos/ingresar HTTP/1.1" 200 -
```

Fuente. Elaboración propia.

Luego de la confirmación del almacenamiento de los datos por parte de Sunny, la API Merry ensambla el mensaje de respuesta (ACK). La Figura 49 demuestra el momento en que Merry publica este payload, el cual detalla el estado de guardado global y los posibles errores a través del tópico dinámico de salida, redirigiéndolo hacia el hardware de origen.

Figura 49. Publicación del mensaje de estado transaccional (ACK) desde la API Merry hacia el tópicos de salida.



```
Archivo  Editar  Pestañas  Ayuda
willy@raspberrypi: ~/Desktop/Sistema_FinalV2/Merry

{
  "reference": "Performer/72613638",
  "display": "Cristopher Casani"
}
},
"effectiveDateTime": "2026-02-28T11:49:29-05:00",
"component": [
  {
    "code": {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "8480-6",
          "display": "Systolic blood pressure"
        }
      ]
    },
    "valueQuantity": {
      "value": 110.00,
      "unit": "mmHg",
      "system": "http://unitsofmeasure.org",
      "code": "mm[Hg]"
    }
  },
  {
    "code": {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "8462-4",
          "display": "Diastolic blood pressure"
        }
      ]
    },
    "valueQuantity": {
      "value": 70.00,
      "unit": "mmHg",
      "system": "http://unitsofmeasure.org",
      "code": "mm[Hg]"
    }
  }
]
}
}

[Merry] Publicado en ESP32/72613637/2026/out: {'sunny_status': 'ok', 'GNU_DB': 'success', 'Maria_DB': 'success', 'message': 'Data stored successfully in all systems', 'errors': [], 'timestamp': '2026-02-28T11:49:31.233484', 'source_topic': 'ESP32/72613637/blood_pressure/2026/in'}
```

Fuente. Elaboración propia

Finalmente, el ciclo de comunicación bidireccional se completa a nivel de hardware. Para visualizar íntegramente la respuesta recibida por el Nodo Simulador (ESP32), la evidencia extraída del monitor serial se ha fragmentado en dos capturas consecutivas, debido a la extensión horizontal del formato JSON. La Figura 50 y la Figura 51 demuestran que el microcontrolador recibe y decodifica correctamente el mensaje emitido por el middleware. De este modo, el dispositivo queda notificado de forma automatizada sobre el éxito o fracaso de la inserción clínica en las bases de datos.

Figura 50. Recepción del mensaje de confirmación (ACK) en el monitor serial del ESP32 (Parte 1).



Fuente. Elaboración propia

Figura 51. Recepción del mensaje de confirmación (ACK) en el monitor serial del ESP32 (Parte 2).



Fuente. Elaboración propia

Con la presentación de estas evidencias, queda validada la operativa de la capa middleware diseñada. Se comprueba que la arquitectura orquestada entre las APIs Merry

y Sunny no solo logra la ingesta y traducción de protocolos (MQTT a HTTP), sino que ejecuta el desensamblaje de estándares de interoperabilidad complejos (HL7 FHIR) para su almacenamiento en motores de bases de datos heterogéneos. Asimismo, la recepción del mensaje de confirmación (ACK) en el monitor serial del microcontrolador demuestra que el emisor es capaz de conocer si su mensaje llegó de forma correcta.

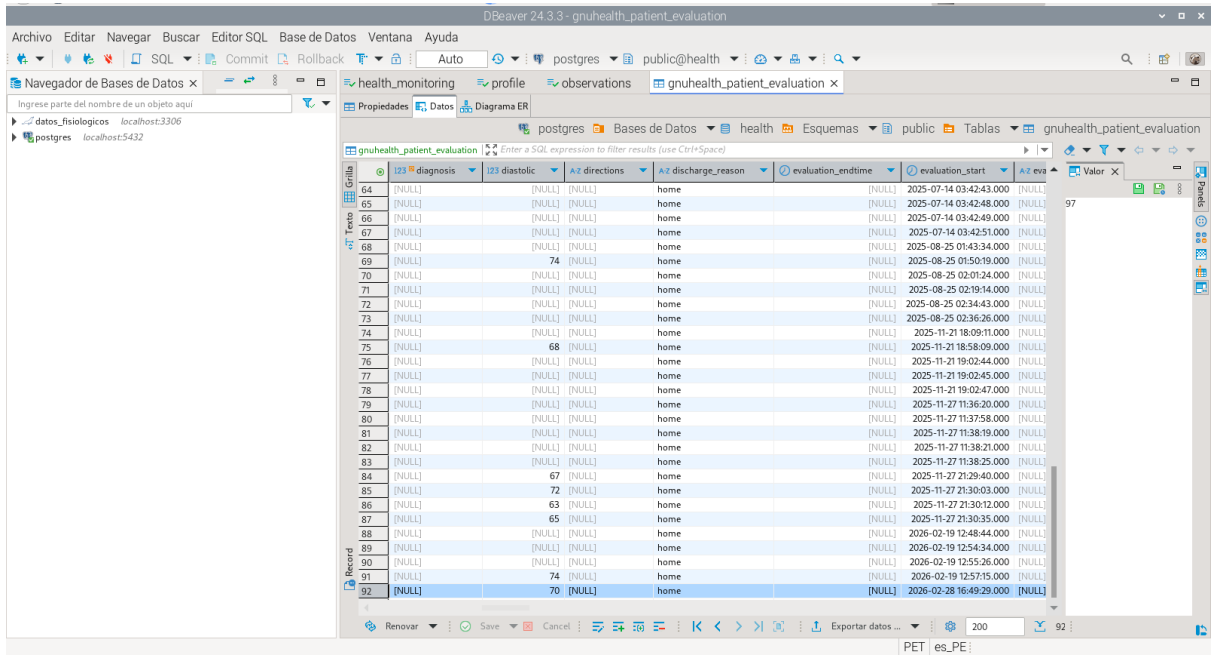
5.1.4.3 Evidencia de ejecución almacenamiento en los repositorios de datos

Con el propósito de validar la culminación exitosa del flujo transaccional y asegurar que los datos procesados por el middleware llegaron a sus repositorios finales, se procedió a auditar físicamente las bases de datos MariaDB y PostgreSQL. Esta auditoría se ejecutó mediante el gestor de bases de datos DBeaver, permitiendo verificar la consistencia estructural del mensaje almacenado. Complementariamente, se comprobó la disponibilidad de la información a través de las interfaces gráficas de usuario (frontends) asociadas a cada entorno, permitiendo así la trazabilidad de extremo a extremo. A continuación, se detallan las evidencias de este proceso.

5.1.4.3.1 Evidencia de almacenamiento en el sistema hospitalario (GNU Health / PostgreSQL)

Como se detalló en el diseño de la arquitectura, el entorno de GNU Health opera sobre una base de datos en PostgreSQL denominada *health*. La tabla central destinada a registrar las mediciones fisiológicas de los pacientes es `gnuhealth_patient_evaluation`. Las siguientes figuras exponen la consulta realizada a esta tabla a través de DBeaver, tras finalizar el procesamiento de las APIs (Merry y Sunny).

Figura 52. Captura del registro en la tabla gnuhealth_patient_evaluation mediante DBeaver (columnas diastolic y evaluation_start).

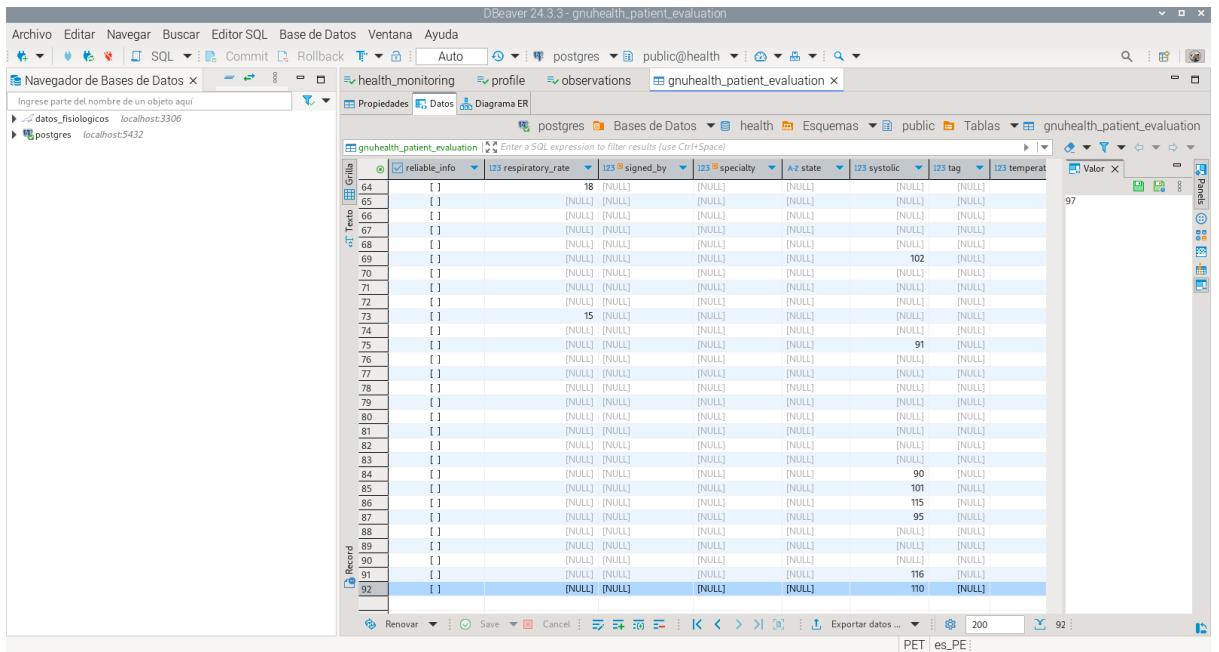


The screenshot shows the DBeaver interface with the 'gnuhealth_patient_evaluation' table selected. The columns 'diastolic' and 'evaluation_start' are highlighted in the header. The table contains 29 rows of data, with the first row (ID 64) showing a diastolic value of [NULL] and an evaluation_start value of 2025-07-14 03:42:43.000.

ID	diagnosis	diastolic	directions	discharge_reason	evaluation_endtime	evaluation_start	ev
64	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-07-14 03:42:43.000	[NULL]
65	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-07-14 03:42:48.000	[NULL]
66	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-07-14 03:42:49.000	[NULL]
67	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-07-14 03:42:51.000	[NULL]
68	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-08-25 01:43:34.000	[NULL]
69	[NULL]	74	[NULL]	home	[NULL]	2025-08-25 01:50:19.000	[NULL]
70	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-08-25 02:01:24.000	[NULL]
71	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-08-25 02:19:14.000	[NULL]
72	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-08-25 02:34:43.000	[NULL]
73	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-08-25 02:36:26.000	[NULL]
74	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-21 18:09:11.000	[NULL]
75	[NULL]	68	[NULL]	home	[NULL]	2025-11-21 18:58:09.000	[NULL]
76	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-21 19:02:44.000	[NULL]
77	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-21 19:02:45.000	[NULL]
78	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-21 19:02:47.000	[NULL]
79	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-27 11:36:20.000	[NULL]
80	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-27 11:37:58.000	[NULL]
81	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-27 11:38:19.000	[NULL]
82	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-27 11:38:21.000	[NULL]
83	[NULL]	[NULL]	[NULL]	home	[NULL]	2025-11-27 11:38:25.000	[NULL]
84	[NULL]	67	[NULL]	home	[NULL]	2025-11-27 11:29:40.000	[NULL]
85	[NULL]	72	[NULL]	home	[NULL]	2025-11-27 11:30:03.000	[NULL]
86	[NULL]	63	[NULL]	home	[NULL]	2025-11-27 11:30:12.000	[NULL]
87	[NULL]	65	[NULL]	home	[NULL]	2025-11-27 11:30:35.000	[NULL]
88	[NULL]	[NULL]	[NULL]	home	[NULL]	2026-02-19 12:48:44.000	[NULL]
89	[NULL]	[NULL]	[NULL]	home	[NULL]	2026-02-19 12:54:34.000	[NULL]
90	[NULL]	[NULL]	[NULL]	home	[NULL]	2026-02-19 12:55:26.000	[NULL]
91	[NULL]	74	[NULL]	home	[NULL]	2026-02-19 12:57:15.000	[NULL]
92	[NULL]	70	[NULL]	home	[NULL]	2026-02-28 16:49:29.000	[NULL]

Fuente. Elaboración propia

Figura 53. Captura del registro en la tabla gnuhealth_patient_evaluation mediante DBeaver (columna systolic).



The screenshot shows the DBeaver interface with the 'gnuhealth_patient_evaluation' table selected. The column 'systolic' is highlighted in the header. The table contains 29 rows of data, with the first row (ID 64) showing a systolic value of [NULL].

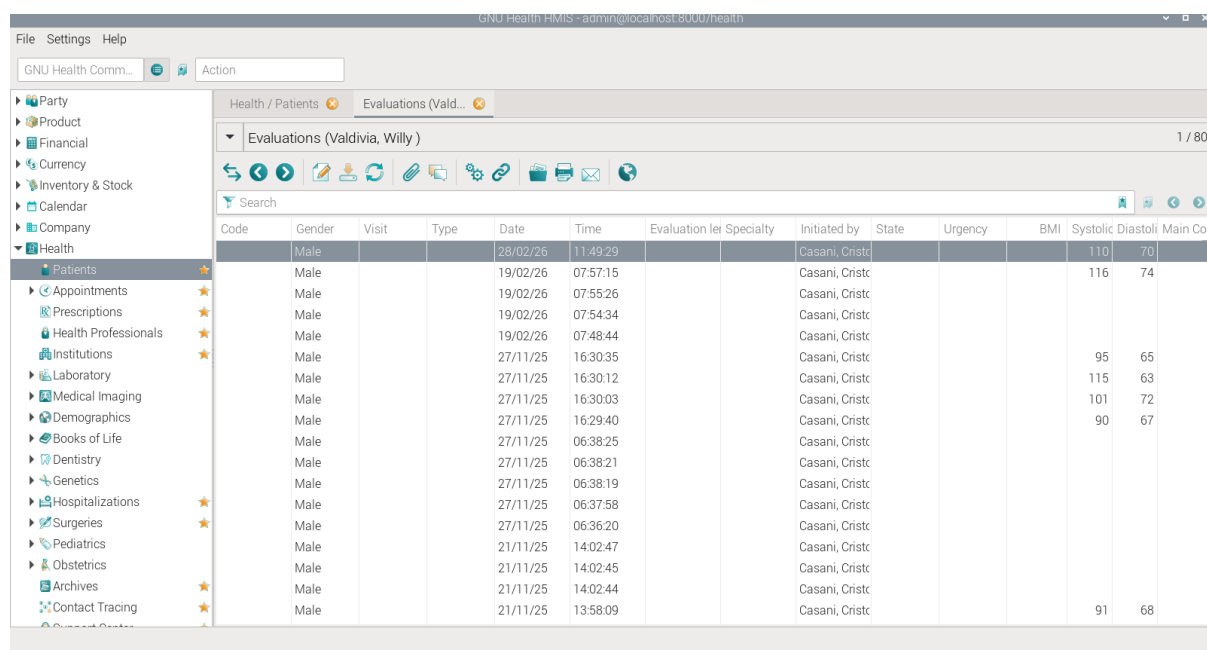
ID	reliable_info	respiratory_rate	signed_by	specialty	state	systolic	tag	temperat
64	[]	18	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
65	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
66	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
67	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
68	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
69	[]	[NULL]	[NULL]	[NULL]	[NULL]	102	[NULL]	[NULL]
70	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
71	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
72	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
73	[]	15	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
74	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
75	[]	[NULL]	[NULL]	[NULL]	[NULL]	91	[NULL]	[NULL]
76	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
77	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
78	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
79	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
80	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
81	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
82	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
83	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
84	[]	[NULL]	[NULL]	[NULL]	[NULL]	90	[NULL]	[NULL]
85	[]	[NULL]	[NULL]	[NULL]	[NULL]	101	[NULL]	[NULL]
86	[]	[NULL]	[NULL]	[NULL]	[NULL]	115	[NULL]	[NULL]
87	[]	[NULL]	[NULL]	[NULL]	[NULL]	95	[NULL]	[NULL]
88	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
89	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
90	[]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]	[NULL]
91	[]	[NULL]	[NULL]	[NULL]	[NULL]	116	[NULL]	[NULL]
92	[]	[NULL]	[NULL]	[NULL]	[NULL]	110	[NULL]	[NULL]

Fuente. Elaboración propia

Como se evidencia en las figuras, al tratarse de un dato clínico compuesto (presión arterial), el sistema mapeó y almacenó correctamente la medición generando dos ingresos en la misma fila: 110 mmHg para la presión sistólica (systolic) y 70 mmHg para la presión diastólica (diastolic), coincidiendo de forma exacta con los valores ingresados al inicio de la transmisión. Asimismo, se constata que la columna evaluation_start contiene una marca de tiempo que refleja con precisión el atributo effectiveDateTime extraído del archivo JSON HL7 FHIR original (previamente evidenciado en el monitor serial del ESP32).

Además del almacenamiento en el backend, resulta necesario comprobar la visualización de este dato para el personal clínico. Para ello, se accedió a la aplicación de escritorio GNU Health Client, el programa oficial del ecosistema destinado a la gestión de la HCE. La Figura 54 exhibe la pestaña "Evaluations" correspondiente al paciente configurado para la prueba, Willy Valdivia.

Figura 54. Panel de evaluaciones clínicas del paciente Willy Valdivia en la interfaz de GNU Health Client.



Code	Gender	Visit	Type	Date	Time	Evaluation lei	Specialty	Initiated by	State	Urgency	BMI	Systolic	Diastoli	Main Con
	Male			28/02/26	11:49:29			Casani, Cristr				110	70	
	Male			19/02/26	07:55:26			Casani, Cristr				116	74	
	Male			19/02/26	07:54:34			Casani, Cristr						
	Male			19/02/26	07:48:44			Casani, Cristr						
	Male			27/11/25	16:30:35			Casani, Cristr				95	65	
	Male			27/11/25	16:30:12			Casani, Cristr				115	63	
	Male			27/11/25	16:30:03			Casani, Cristr				101	72	
	Male			27/11/25	16:29:40			Casani, Cristr				90	67	
	Male			27/11/25	06:38:25			Casani, Cristr						
	Male			27/11/25	06:38:21			Casani, Cristr						
	Male			27/11/25	06:38:19			Casani, Cristr						
	Male			27/11/25	06:37:58			Casani, Cristr						
	Male			27/11/25	06:36:20			Casani, Cristr						
	Male			21/11/25	14:02:47			Casani, Cristr						
	Male			21/11/25	14:02:45			Casani, Cristr						
	Male			21/11/25	14:02:44			Casani, Cristr						
	Male			21/11/25	13:58:09			Casani, Cristr				91	68	

Fuente. Elaboración propia

En esta vista general, se comprueba la creación de un nuevo registro clínico, cuya fecha y hora coinciden con la marca temporal de la transmisión. El sistema refleja de manera amigable los valores de presión sistólica y diastólica ingresados. Además, en cumplimiento con las normas de trazabilidad médica, el profesional de la salud asignado

en la etapa de configuración (Dr. Cristopher Casani) figura debidamente registrado como el médico responsable de efectuar la evaluación.

Al hacer doble clic sobre esta fila, el sistema despliega la ficha completa de la medición, tal como se observa en la Figura 55, corroborando la consolidación detallada de todas las variables fisiológicas en la interfaz dada por GNU Health Client.

Figura 55. Vista detallada del registro de presión arterial en GNU Health Client.

The screenshot shows the GNU Health Client interface. The top menu bar includes 'File', 'Settings', and 'Help'. Below it is a search bar and an 'Action' button. The left sidebar contains a tree view of categories: Party, Product, Financial, Currency, Inventory & Stock, Calendar, Company, and Health. Under 'Health', there are sub-items like Patients, Appointments, Prescriptions, Health Professionals, Institutions, Laboratory, Medical Imaging, Demographics, Books of Life, Dentistry, Genetics, Hospitalizations, Surgeries, Pediatrics, Obstetrics, Archives, and Contact Tracing. The main window title is 'GNU Health HMIS - admin@localhost:8000/health'. The main content area is titled 'Health / Patients' and 'Evaluations (Valdivia, Willy)'. It shows patient details: Patient: Valdivia, Willy, Male, 24y 6m 25d, Code: [blank]. Visit: [blank], Initiated by: Casani, Cristopher. The 'Clinical' tab is active, showing 'Vital Signs' with fields for Temperature, Heart Rate, Systolic Pressure (110), Diastolic Pressure (70), Respiratory Rate, and Oxygen Saturation. Below this is 'Anthropometry' with fields for Weight, Height, BMI, Waist, Hip, and WHR. There is also a 'Nutrition' section with 'Malnutrition' and 'Dehydration' checkboxes. At the bottom, the 'Signs and Symptoms' section includes 'Start' (28/02/26, 11:49:29), 'End', 'State', 'Signed by', 'Discharge Reason' (Home / Selfcare), and a 'Discharge' button.

Fuente. Elaboración propia

5.1.4.3.2 Evidencia de almacenamiento en base de datos MariaDB

Dado que la arquitectura del sistema contempla el uso de una base de datos relacional paralela (MariaDB) diseñada específicamente para dar soporte a las consultas del frontend, se procedió a auditar la persistencia de los datos en este entorno. Para ello, se consultó la tabla observations perteneciente a la base de datos health_monitoring, la cual consolida todos los registros procesados por el middleware. Las siguientes figuras exponen el nuevo registro insertado al concluir la transmisión del dato simulado de presión arterial:

Figura 56. Captura del registro en la tabla observations mediante DBeaver (metadatos de usuario y paciente).

id	physiological_type	patient_name	patient_id	doctor_name	doctor_id	device_name
30	respiratory_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
31	respiratory_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
32	oxygen_saturation	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
33	oxygen_saturation	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
34	oxygen_saturation	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
35	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
36	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
37	oxygen_saturation	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
38	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
39	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
40	respiratory_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
41	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
42	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
43	heart_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
44	heart_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
45	heart_rate	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
46	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
47	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
48	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
49	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
50	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
51	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
52	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
53	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
54	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
55	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
56	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
57	temperature	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
58	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32
59	blood_pressure	Willy Valdivia	72613637	Cristopher Casani	72613638	ESP32

Fuente. Elaboración propia

Figura 57. Captura del registro en la tabla observations mediante DBeaver (metadatos de dispositivo, valores de presión arterial y marca de tiempo autogenerada)).

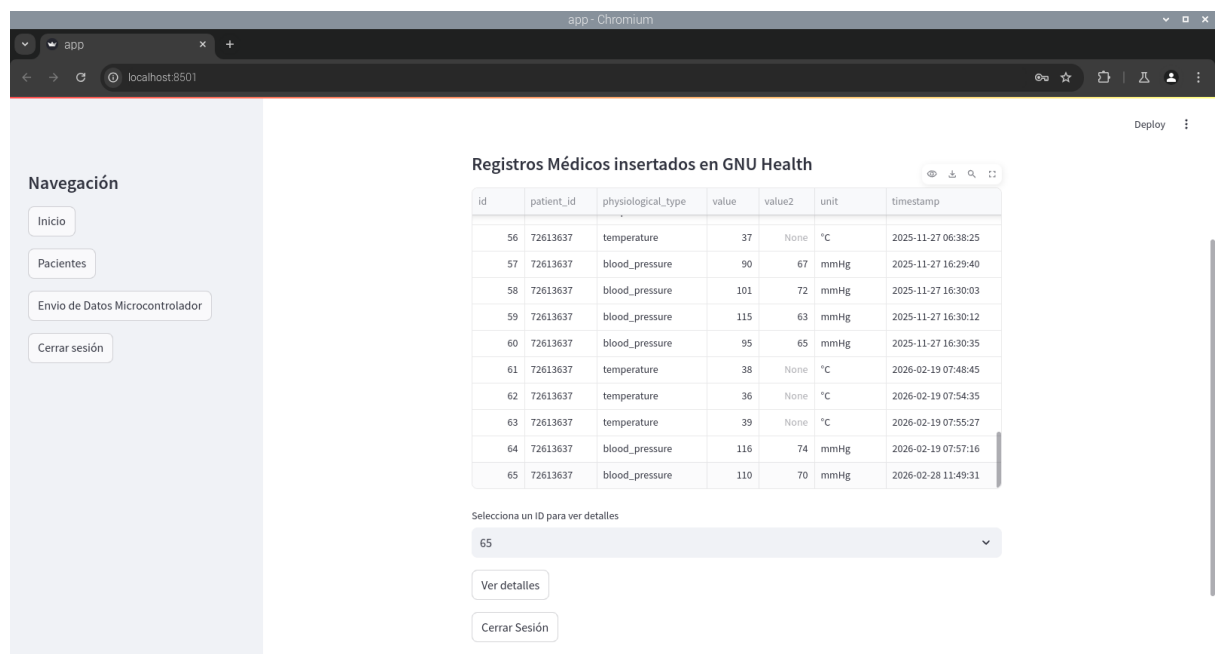
id	device_name	device_id	value	unit	timestamp	full_message
30	ESP32	2025	17	[NULL]	2025-07-13 22:42:41.000	["resourceType": "Observati
31	ESP32	2025	18	[NULL]	2025-07-13 22:42:48.000	["resourceType": "Observati
32	ESP32	2025	100	%02	2025-07-13 22:42:48.000	["resourceType": "Observati
33	ESP32	2025	100	%02	2025-07-13 22:42:49.000	["resourceType": "Observati
34	ESP32	2025	99	%02	2025-07-13 22:42:51.000	["resourceType": "Observati
35	ESP32	2025	38	[NULL]	2025-08-24 20:43:34.000	["resourceType": "Observati
36	ESP32	2025	102	mmHg	2025-08-24 20:50:19.000	["resourceType": "Observati
37	ESP32	2025	98	%02	2025-08-24 21:01:24.000	["resourceType": "Observati
38	ESP32	2025	36	[NULL]	2025-08-24 21:19:14.000	["resourceType": "Observati
39	ESP32	2025	37	[NULL]	2025-08-24 21:34:43.000	["resourceType": "Observati
40	ESP32	2025	15	[NULL]	2025-08-24 21:36:26.000	["resourceType": "Observati
41	ESP32	2025	36	[NULL]	2025-11-21 13:09:12.000	["resourceType": "Observati
42	ESP32	2025	91	mmHg	2025-11-21 13:58:09.000	["resourceType": "Observati
43	ESP32	2025	66	beats/min	2025-11-21 14:02:44.000	["resourceType": "Observati
44	ESP32	2025	104	beats/min	2025-11-21 14:02:45.000	["resourceType": "Observati
45	ESP32	2025	95	beats/min	2025-11-21 14:02:47.000	["resourceType": "Observati
46	ESP32	2025	38	[NULL]	2025-11-27 06:36:20.000	["resourceType": "Observati
47	ESP32	2025	36	[NULL]	2025-11-27 06:37:58.000	["resourceType": "Observati
48	ESP32	2025	36	[NULL]	2025-11-27 06:38:16.000	["resourceType": "Observati
49	ESP32	2025	37	[NULL]	2025-11-27 06:38:22.000	["resourceType": "Observati
50	ESP32	2025	37	[NULL]	2025-11-27 06:38:25.000	["resourceType": "Observati
51	ESP32	2025	90	mmHg	2025-11-27 16:29:40.000	["resourceType": "Observati
52	ESP32	2025	101	mmHg	2025-11-27 16:30:03.000	["resourceType": "Observati
53	ESP32	2025	115	mmHg	2025-11-27 16:30:12.000	["resourceType": "Observati
54	ESP32	2025	95	mmHg	2025-11-27 16:30:35.000	["resourceType": "Observati
55	ESP32	2026	38	[NULL]	2026-02-19 07:48:45.000	["resourceType": "Observati
56	ESP32	2026	36	[NULL]	2026-02-19 07:54:35.000	["resourceType": "Observati
57	ESP32	2026	39	[NULL]	2026-02-19 07:55:27.000	["resourceType": "Observati
58	ESP32	2026	116	mmHg	2026-02-19 07:57:16.000	["resourceType": "Observati
59	ESP32	2026	110	mmHg	2026-02-28 11:49:31.000	["resourceType": "Observati

Fuente. Elaboración propia

A través de estas capturas, se corrobora que la API Sunny extrajo e insertó correctamente los metadatos configurados (identificadores del paciente, médico tratante y dispositivo). Asimismo, se evidencia que los valores de la variable compuesta (110 y 70) fueron aislados y guardados en columnas independientes, conservando su respectiva unidad de medida (mmHg).

Complementando la validación, se verificó la disponibilidad de estos datos en la interfaz web desarrollada en Streamlit. Esta plataforma renderiza la información realizando peticiones de lectura a la base de datos MariaDB a través del middleware. En primer lugar, la Figura 58 exhibe el panel principal de la plataforma tras la autenticación del usuario.

Figura 58. Panel principal de monitoreo en Streamlit mostrando el historial de registros recientes.

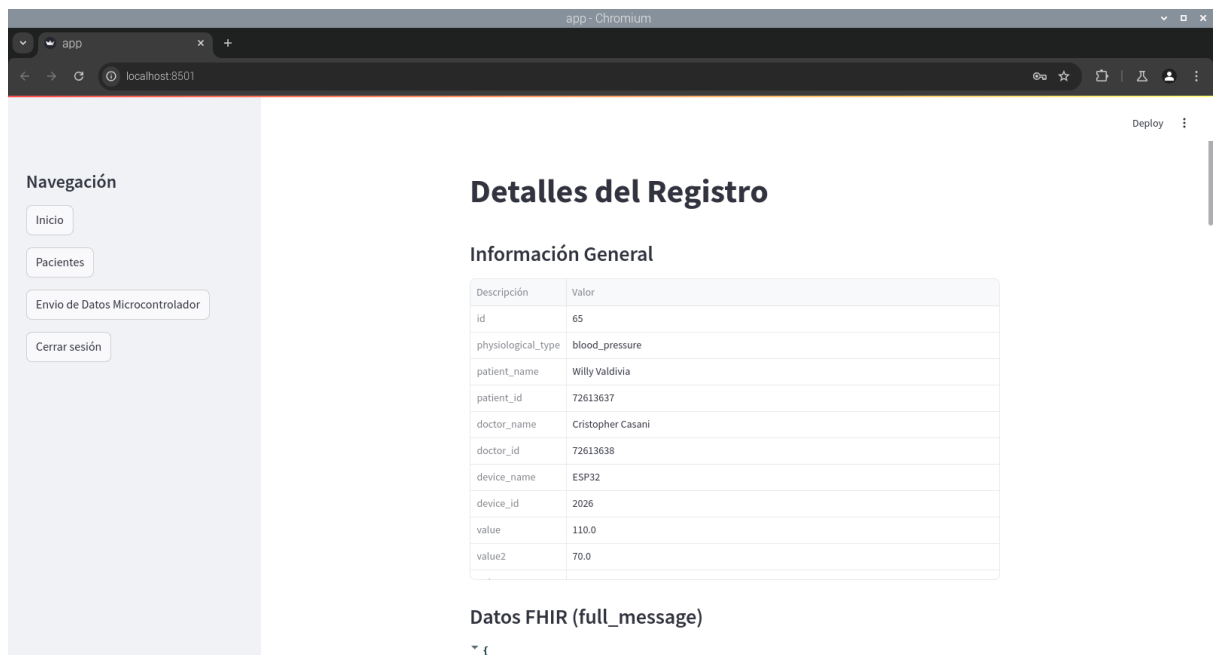


id	patient_id	physiological_type	value	value2	unit	timestamp
56	72613637	temperature	37	None	°C	2025-11-27 06:38:25
57	72613637	blood_pressure	90	67	mmHg	2025-11-27 16:29:40
58	72613637	blood_pressure	101	72	mmHg	2025-11-27 16:30:03
59	72613637	blood_pressure	115	63	mmHg	2025-11-27 16:30:12
60	72613637	blood_pressure	95	65	mmHg	2025-11-27 16:30:35
61	72613637	temperature	38	None	°C	2026-02-19 07:48:45
62	72613637	temperature	36	None	°C	2026-02-19 07:54:35
63	72613637	temperature	39	None	°C	2026-02-19 07:55:27
64	72613637	blood_pressure	116	74	mmHg	2026-02-19 07:57:16
65	72613637	blood_pressure	110	70	mmHg	2026-02-28 11:49:31

Fuente. Elaboración propia

En este panel general se observa que el último registro ingresado corresponde efectivamente al caso de prueba ejecutado. Al interactuar con la interfaz y acceder a los detalles específicos de dicha transmisión, el sistema despliega una vista exhaustiva del evento. En la Figura 59, los metadatos principales se presentan de manera tabular para facilitar la lectura clínica rápida.

Figura 59. Vista detallada de los metadatos del registro clínico en la interfaz web.



Fuente. Elaboración propia

Adicionalmente, el sistema expone el documento HL7 FHIR de forma íntegra como respaldo de auditoría técnica. Dada la extensión vertical y la estructura anidada característica de este estándar de interoperabilidad, la evidencia de su renderizado en la página Details se presenta segmentada en las siguientes capturas consecutivas:

Figura 60. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 1).

device_name	ESP32
device_id	2026
value	110.0
value2	70.0
unit	mmHg
timestamp	2026-02-28 11:49:31

Datos FHIR (full_message)

```
{
  "resourceType": "Observation",
  "id": "blood_pressure",
  "meta": {
    "profile": [
      0: "http://hl7.org/fhir/StructureDefinition/bp"
    ]
  },
  "text": {
    "status": "generated",
    "div": "
      <div xmlns='http://www.w3.org/1999/xhtml'>Observation for Willy Valdivia</div>
    "
  },
  "status": "final"
}
```

Fuente. Elaboración propia

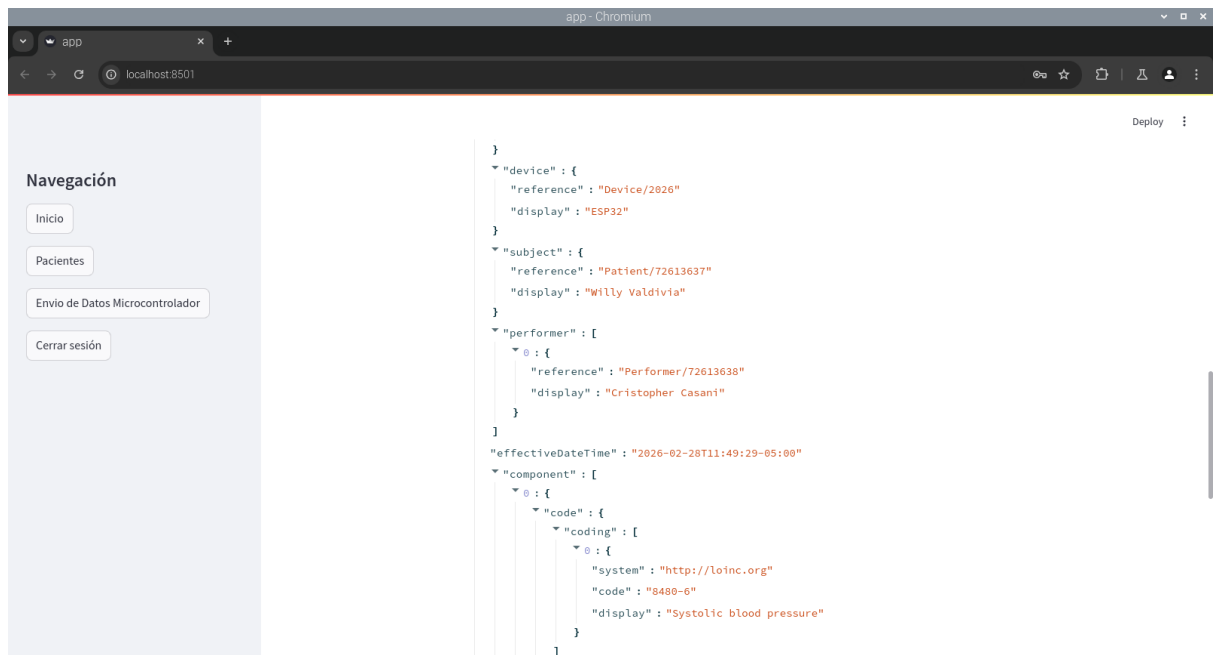
Figura 61. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 2).

```

    "status": "final"
  },
  "category": [
    0: {
      "coding": [
        0: {
          "system": "http://terminology.hl7.org/CodeSystem/observation-category",
          "code": "vital-signs",
          "display": "Vital Signs"
        }
      ],
      "text": "Vital Signs"
    }
  ],
  "code": {
    "coding": [
      0: {
        "system": "http://loinc.org",
        "code": "85354-9",
        "display": "Blood pressure panel with all children optional"
      }
    ]
  },
  "device": {
    "reference": "Device/2026",
    "display": "ESP32"
  }
}
```

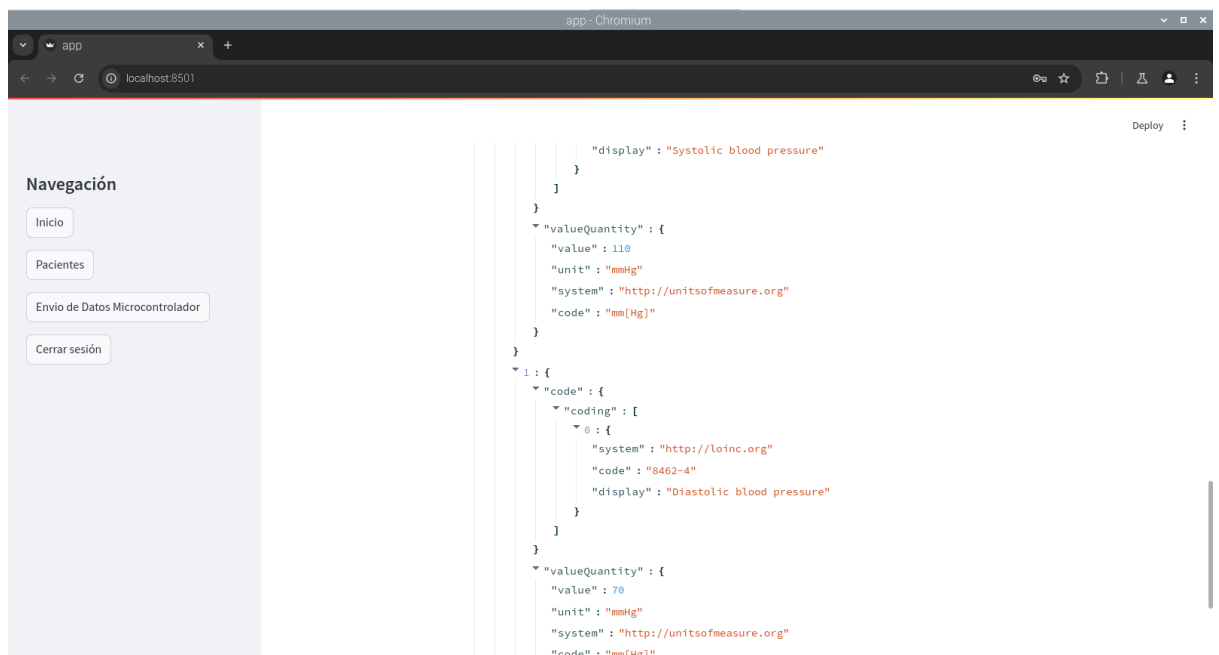
Fuente. Elaboración propia

Figura 62. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 3).



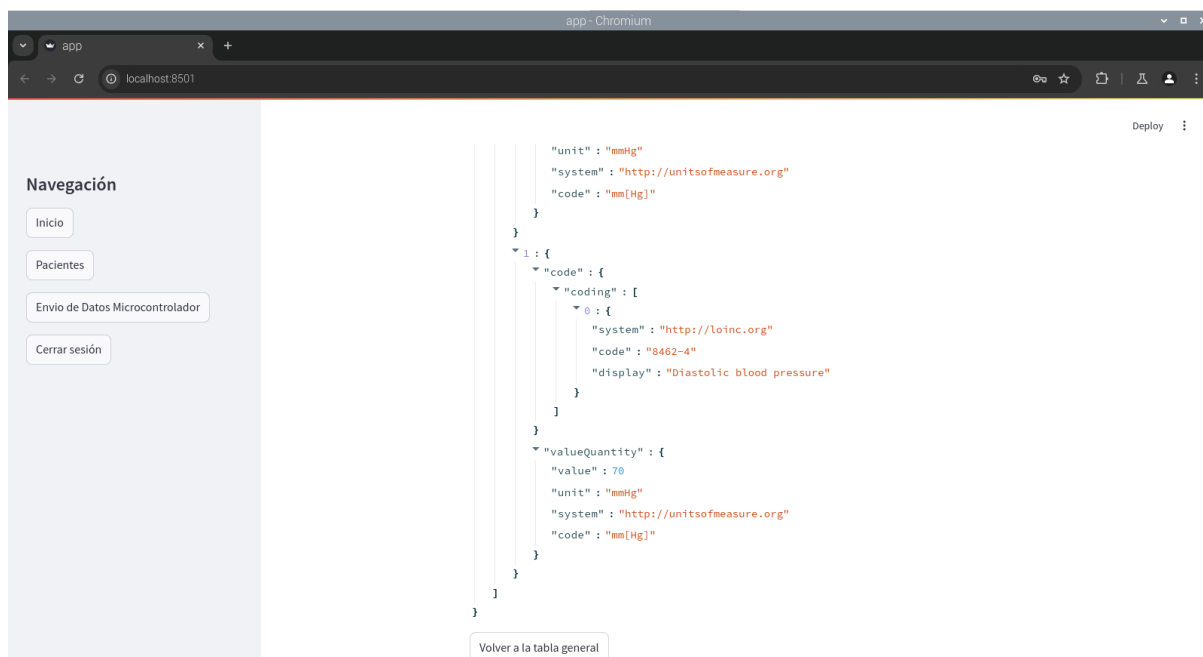
Fuente. Elaboración propia

Figura 63. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 4).



Fuente. Elaboración propia

Figura 64. Vista detallada del mensaje FHIR en la interfaz web a través de la página Details (Parte 5).



Fuente. Elaboración propia

Cabe destacar una particularidad arquitectónica reflejada en esta interfaz: la marca de tiempo (timestamp) mostrada en los detalles no coincide exactamente con el atributo `effectiveDateTime` encapsulado en el mensaje FHIR. Esta divergencia es intencional; mientras que GNU Health registra el momento clínico exacto de la medición, la base de datos MariaDB (y por extensión, la interfaz en Streamlit) registra el instante preciso de llegada y procesamiento del mensaje en el servidor central. De este modo, la interfaz web cumple una doble función: visualización de parámetros médicos y herramienta de trazabilidad técnica (auditoría de red) para los datos recibidos en el sistema.

5.2 Evaluación de consistencia y segregación de datos mediante Matriz Combinatoria (N=120)

En concordancia con el protocolo experimental descrito en la Sección 4.6.4.2, la segunda fase de validación consistió en evaluar la consistencia funcional del sistema, la segregación de la información biomédica y el enrutamiento de los datos mediante la ejecución de la matriz combinatoria definida para el estudio. Para ello, se establecieron dos perfiles de configuración remota del microcontrolador, alternando pacientes, médicos tratantes y dispositivos de origen, todos ellos previamente registrados en la base de datos

maestra de GNU Health. Las credenciales y parámetros de red inyectados al Nodo Simulador para cada escenario fueron los siguientes:

- **Configuración 1 (Perfil A):**

- ❖ Paciente: Milagros Alvites (ID / DNI: 12243648)
- ❖ Médico Tratante: Cristopher Casani (ID / DNI: 72613638)
- ❖ Dispositivo lógico configurado: ESP32 (ID: 1002)

- **Configuración 2 (Perfil B):**

- ❖ Paciente: Jose Cebrian (ID / DNI: 58213479)
- ❖ Médico Tratante: Richard Lonconi (ID / DNI: 04244464)
- ❖ Dispositivo lógico configurado: Arduino Uno (ID: 2018). La ejecución física se realizó sobre el nodo simulador ESP32.

Luego de configurar estos parámetros en el firmware del dispositivo, se procedió con la ejecución del protocolo de 120 transmisiones estructuradas. De acuerdo con el diseño experimental descrito previamente, los 24 escenarios combinatorios (2 perfiles x 6 variables x 2 modalidades) se ejecutaron bajo un esquema de 5 repeticiones consecutivas cada uno. Esto resultó en 10 mediciones por cada uno de los seis tipos de datos fisiológicos habilitados (5 envíos mediante generación aleatoria en hardware y 5 envíos mediante inyección personalizada desde la interfaz web), alcanzando 60 mediciones por perfil de paciente y un total acumulado de 120 transmisiones registradas durante la evaluación.

A continuación, se presentan las evidencias gráficas que documentan los resultados obtenidos durante esta evaluación. En primer lugar, se exponen las páginas de monitoreo (Data View) de la interfaz web en Streamlit, donde se constata la correcta segregación, recepción y visualización de los datos para cada configuración. Posteriormente, se exhiben las vistas de las HCE dentro del entorno GNU Health Client, evidenciando que la información clínica de ambos pacientes fue persistida de manera íntegra, sin colisiones de datos y respetando el modelo relacional del sistema hospitalario.

5.2.1 Renderizado y evolución clínica en la Interfaz Web

Como evidencia de la recepción y segregación de los datos transmitidos, se presentan las vistas obtenidas desde la página Data View de Streamlit. El flujo de validación consistió

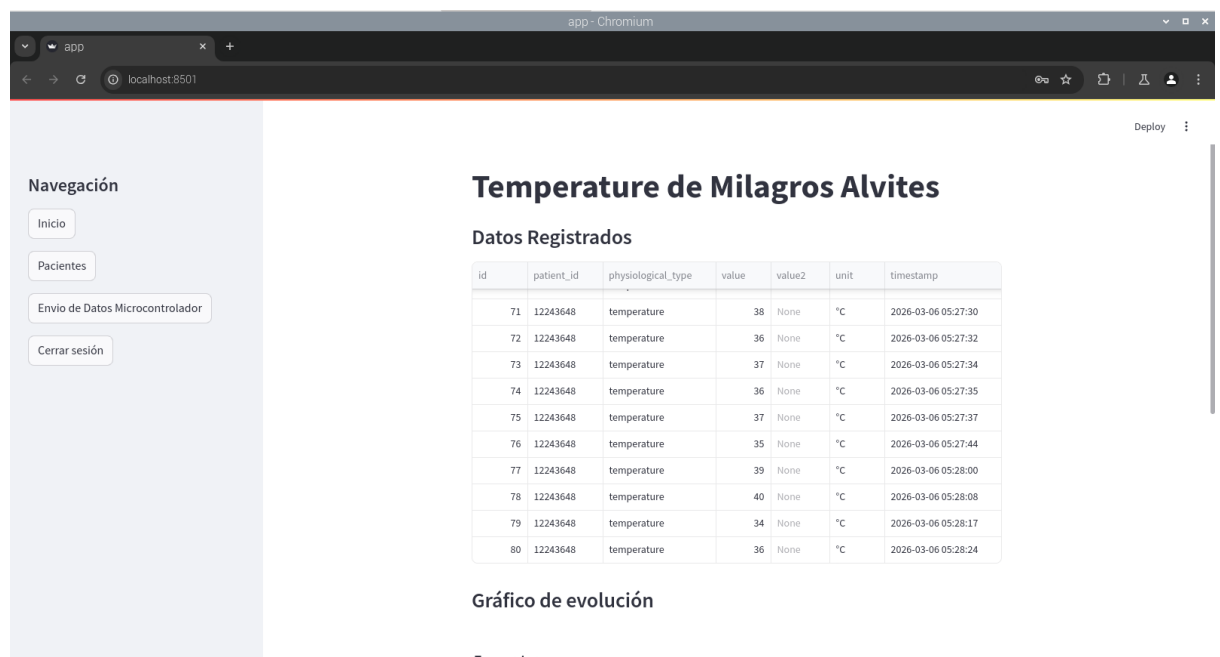
en acceder al directorio general de la plataforma y consultar los historiales específicos de los dos pacientes configurados al inicio del protocolo.

Al contar con seis variables fisiológicas evaluadas de forma independiente para cada perfil, se ha documentado el comportamiento de la interfaz ante cada una de ellas. Dada la extensión de los paneles y la densidad de la información renderizada, la evidencia de cada parámetro fisiológico se ha estructurado en dos capturas consecutivas:

- **Parte 1 (Historial Tabular y Valores Atípicos):** Presenta el registro detallado de los datos almacenados en la base de datos espejo. Cabe destacar que, del total de 10 registros por parámetro, 5 corresponden deliberadamente a valores atípicos (fuera del rango fisiológico normal). La inclusión deliberada de valores atípicos permitió verificar el funcionamiento de la modalidad de envío personalizado.
- **Parte 2 (Curvas de Tendencia):** Expone el gráfico de evolución temporal generado dinámicamente por la interfaz. Estos resultados evidencian que la plataforma es capaz de interpretar los historiales tabulares y traducirlos a herramientas visuales para representar la evolución de los parámetros.

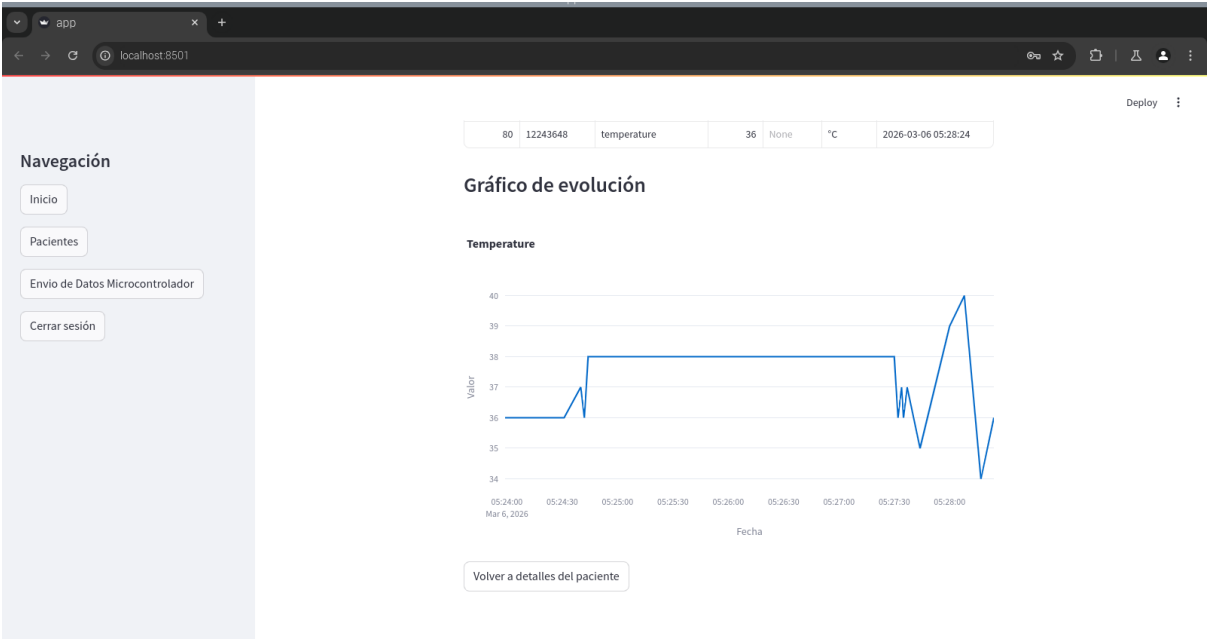
A continuación, se presentan las evidencias correspondientes al Perfil A (Configuración 1 - Paciente: Milagros Alvites):

Figura 65. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 1: Tabular).



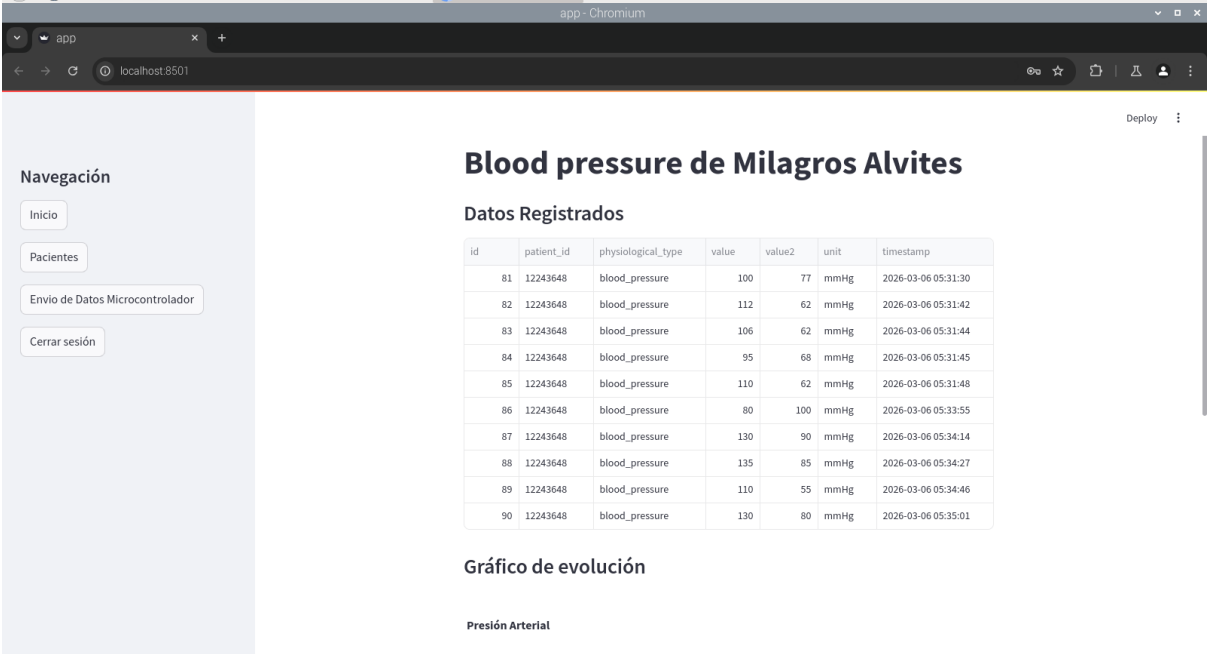
Fuente. Elaboración propia

Figura 66. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 2: Gráfica)



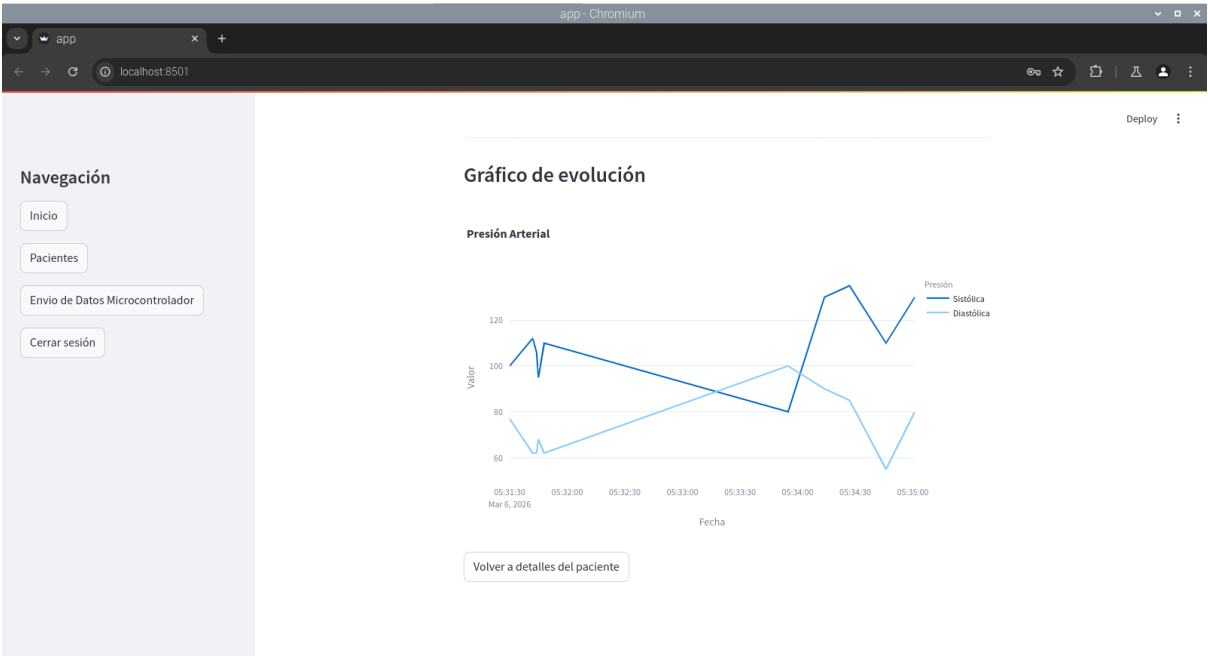
Fuente. Elaboración propia

Figura 67. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 1: Tabular).



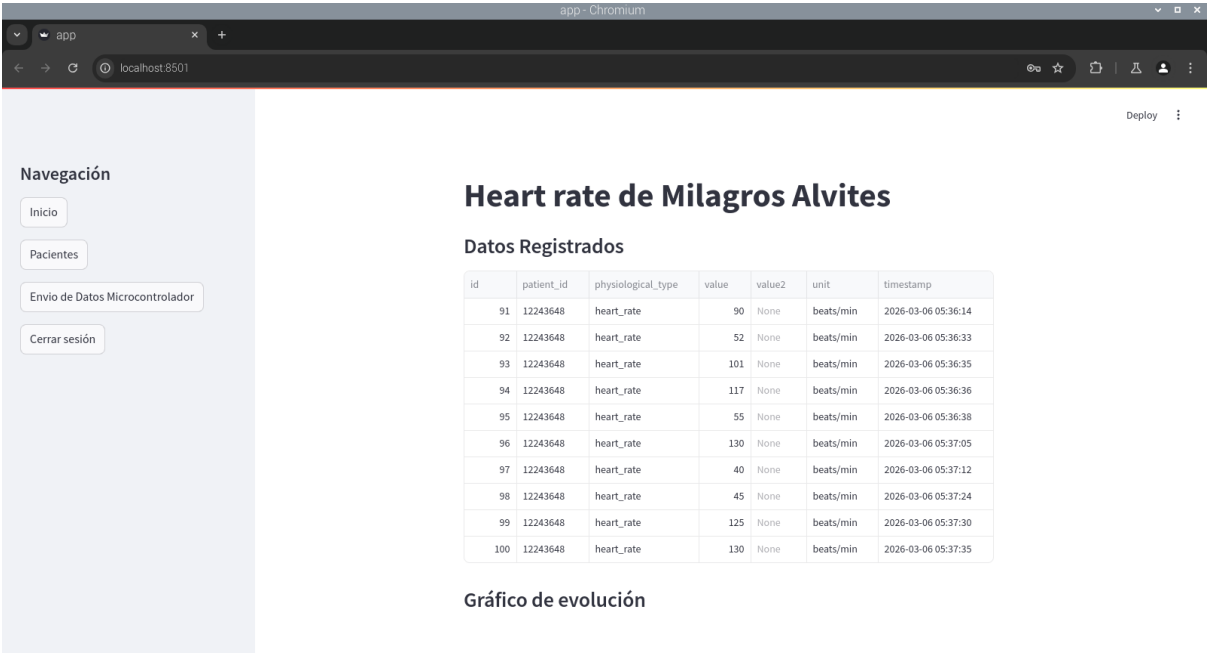
Fuente. Elaboración propia

Figura 68. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 2: Gráfica).



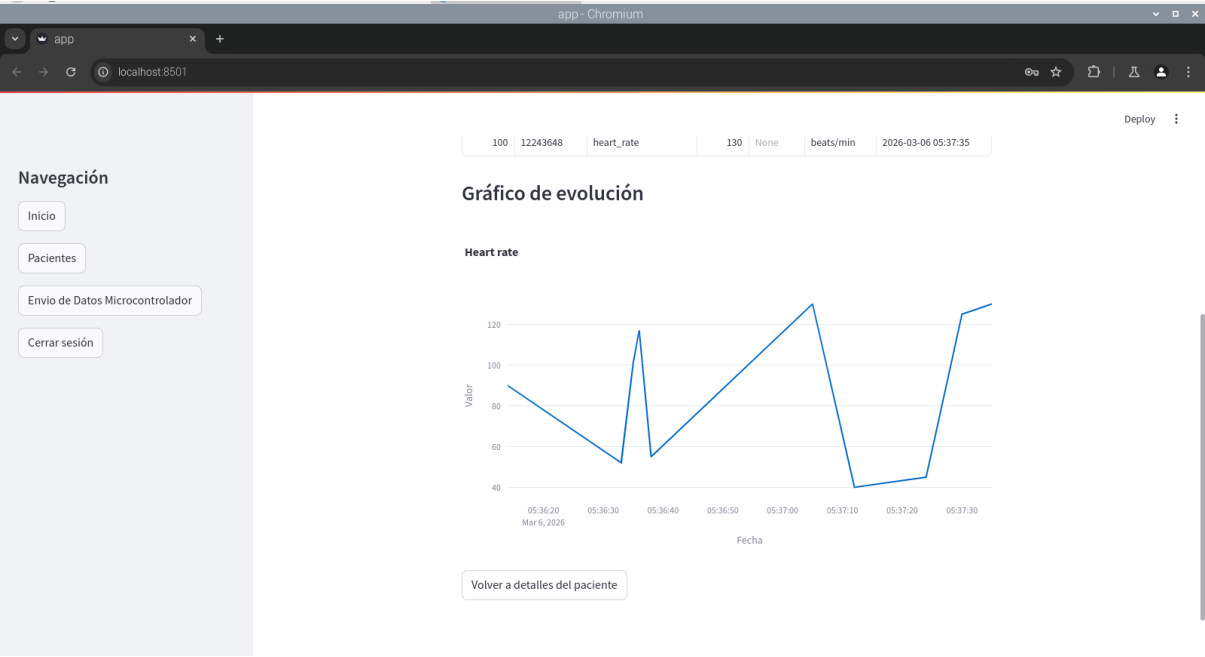
Fuente. Elaboración propia

Figura 69. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 1: Tabular).



Fuente. Elaboración propia

Figura 70. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 2: Gráfica).



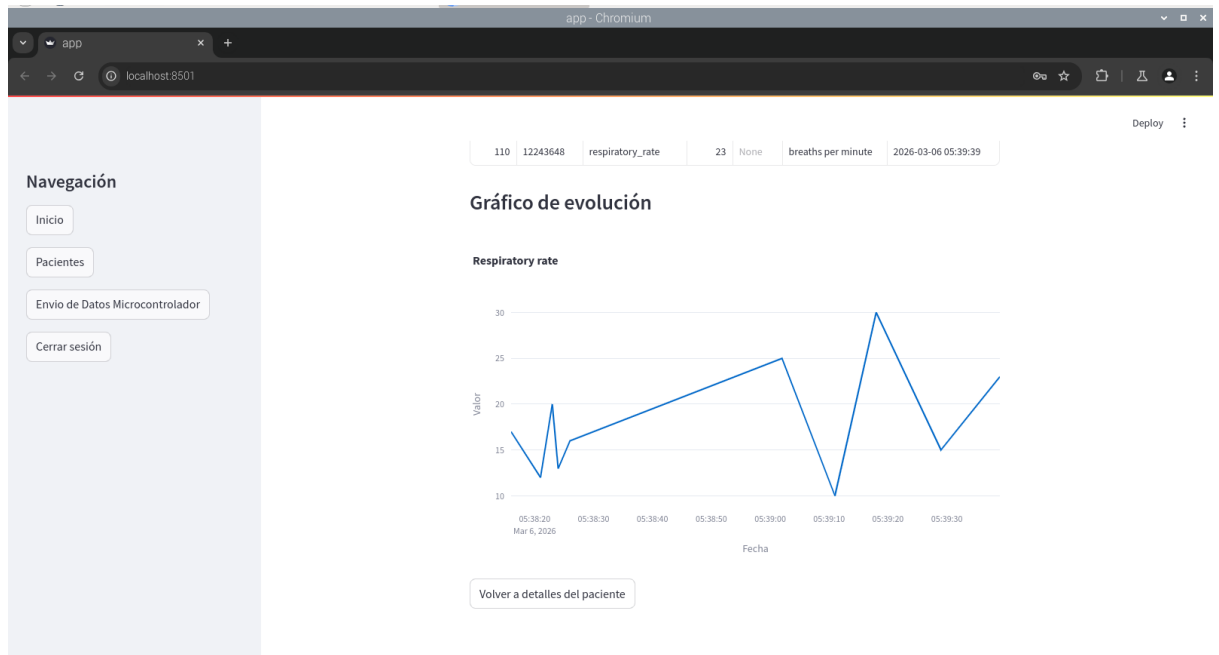
Fuente. Elaboración propia

Figura 71. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria (Parte 1: Tabular)



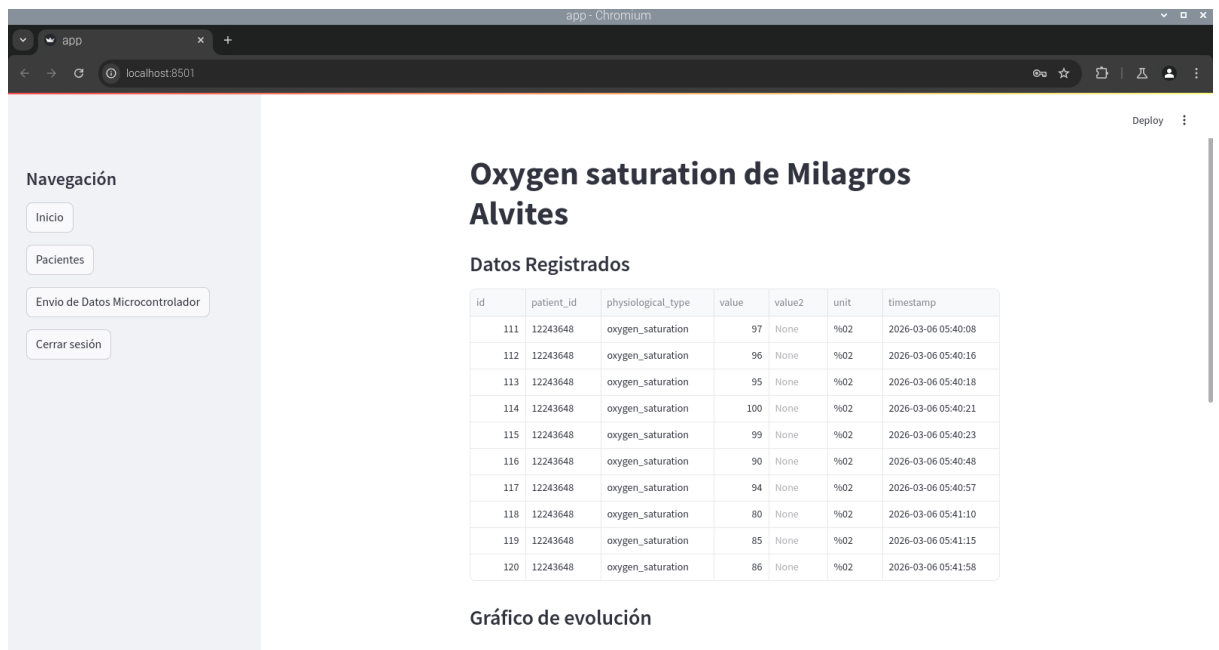
Fuente. Elaboración propia

Figura 72. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria
(Parte 2: Gráfica)



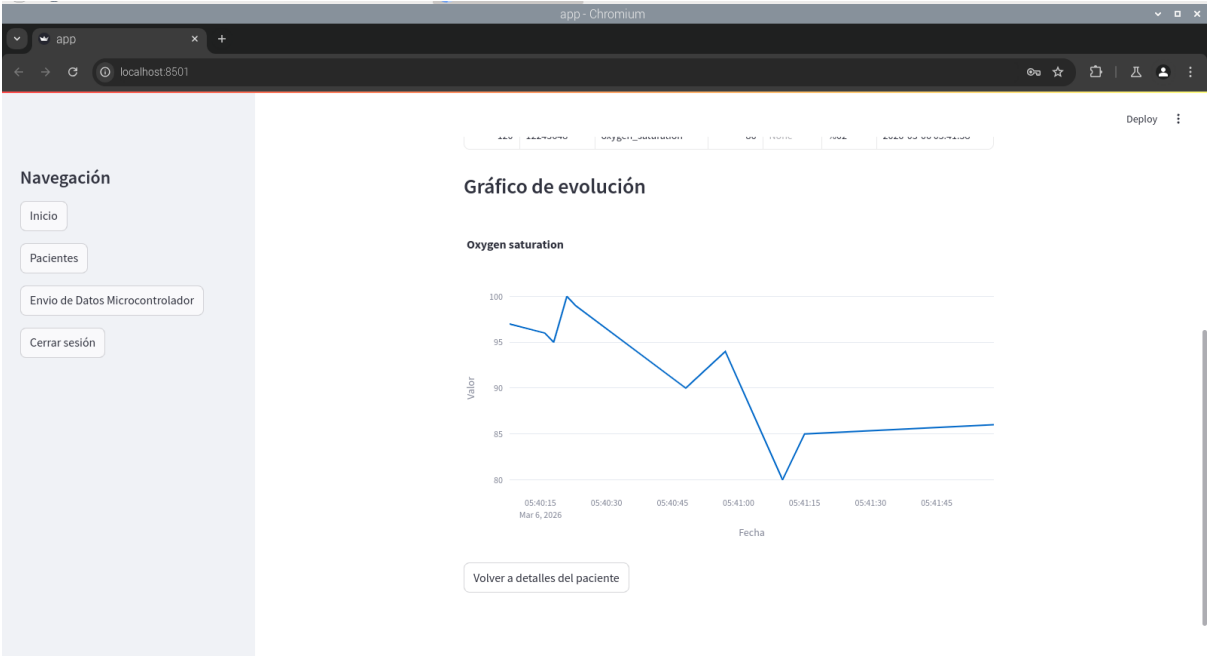
Fuente. Elaboración propia

Figura 73. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno
(Parte 1: Tabular).



Fuente. Elaboración propia

Figura 74. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 2: Gráfica).



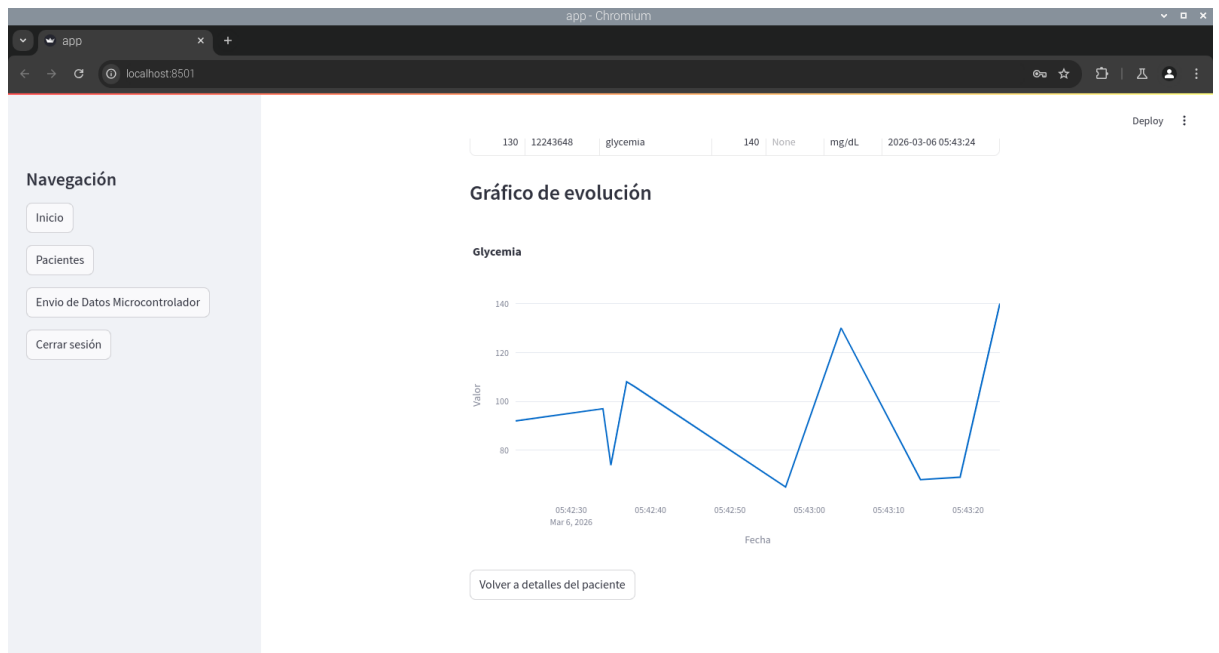
Fuente. Elaboración propia

Figura 75. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 1: Tabular)



Fuente. Elaboración propia

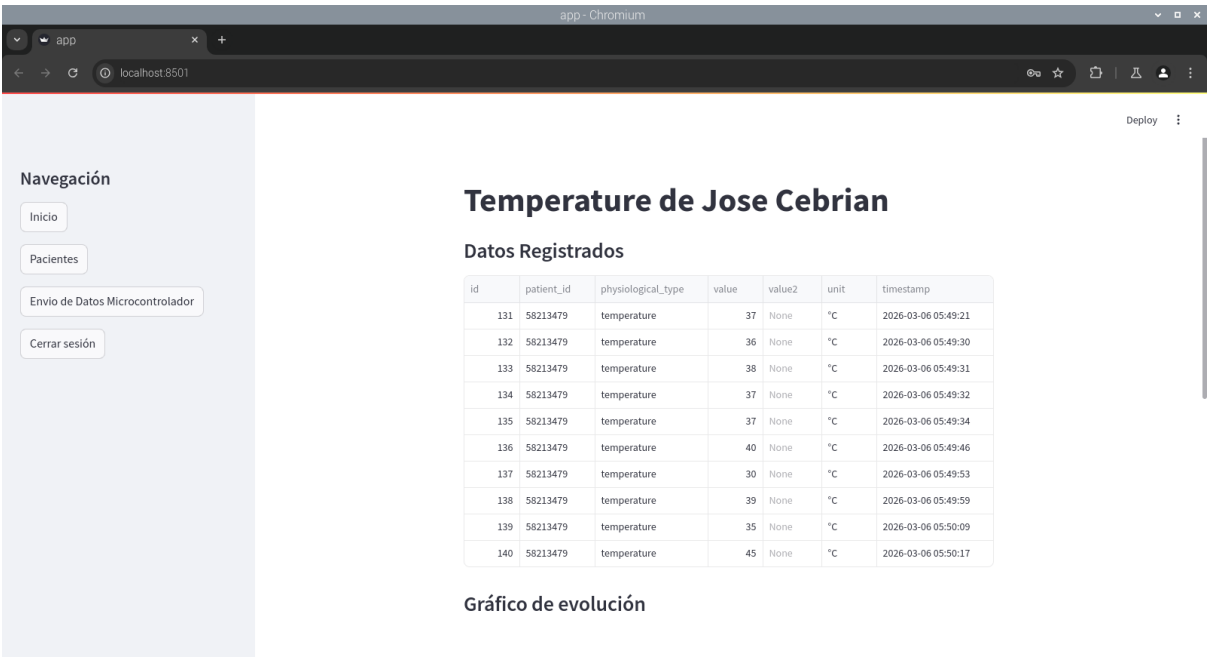
Figura 76. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 2: Gráfica)



Fuente. Elaboración propia

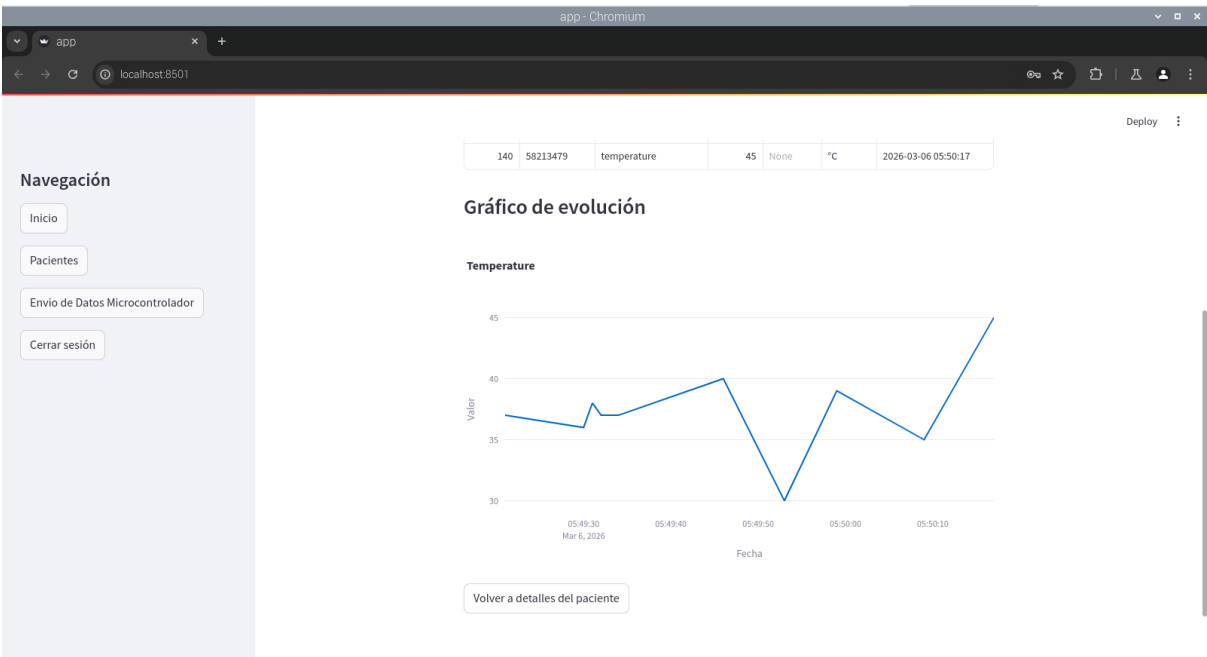
De la misma forma, se exponen las evidencias correspondientes al Perfil B (Configuración 2 - Paciente: Jose Cebrian), evidenciando que el middleware realiza la correcta segregación de la información y la ausencia de colisiones de datos entre ambos pacientes

Figura 77. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 1: Tabular).



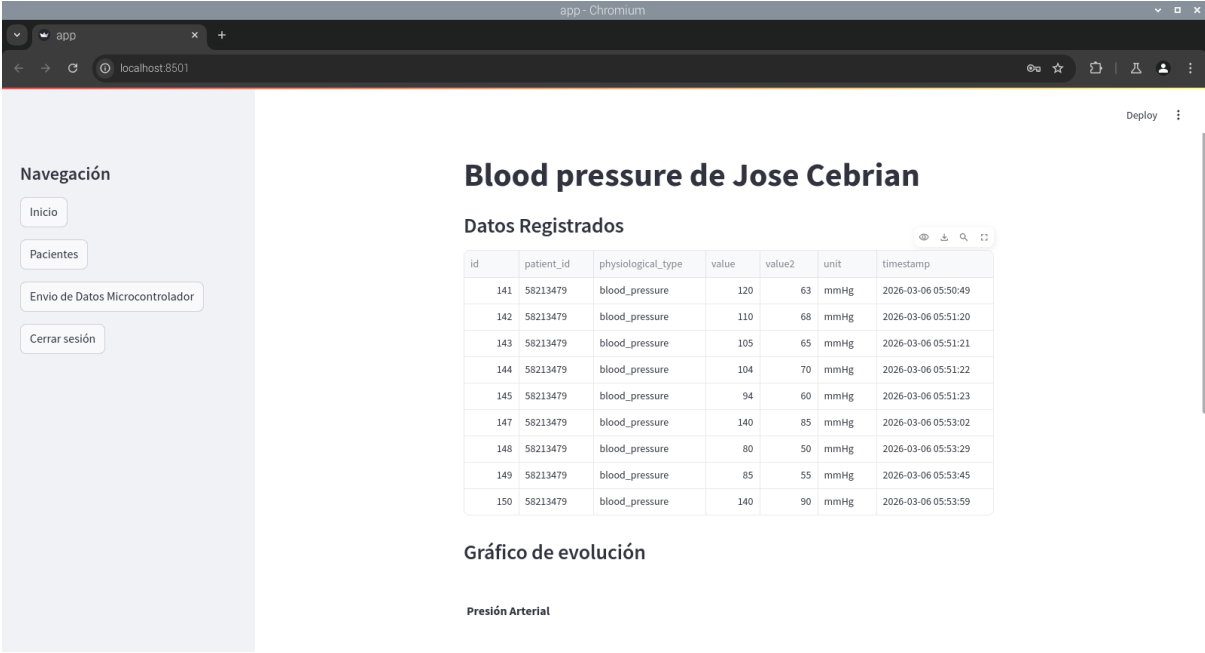
Fuente. Elaboración propia

Figura 78. Vista de la pantalla Data-view correspondiente a la temperatura (Parte 2: Gráfica)



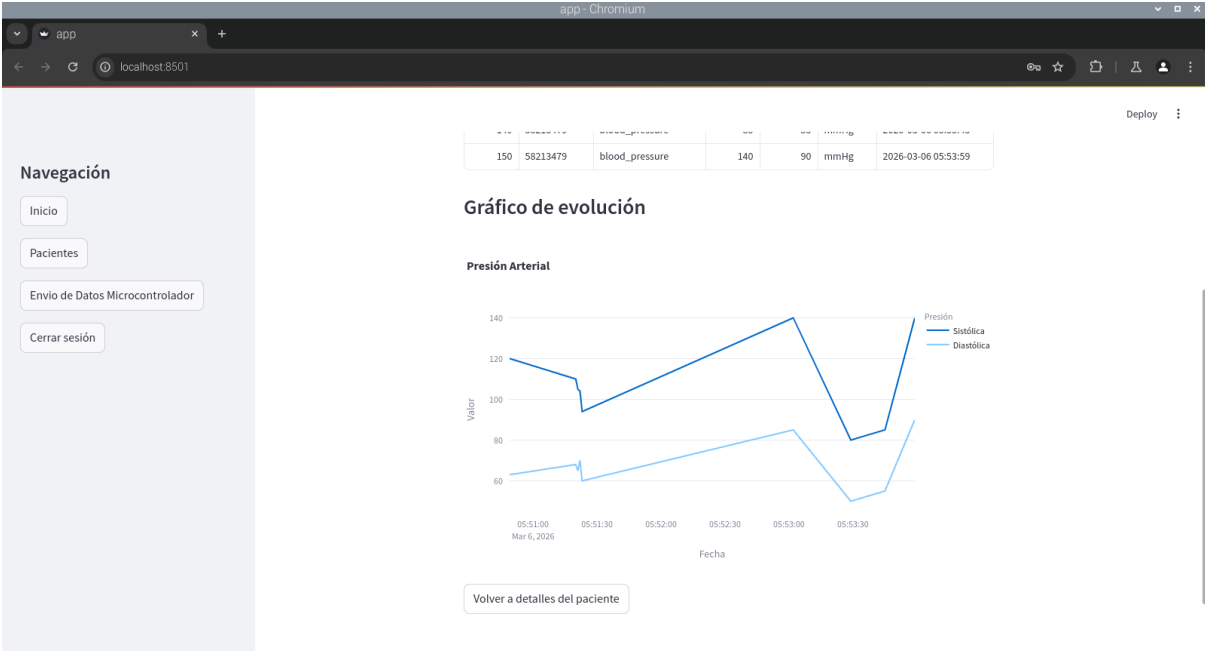
Fuente. Elaboración propia

Figura 79. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 1: Tabular).



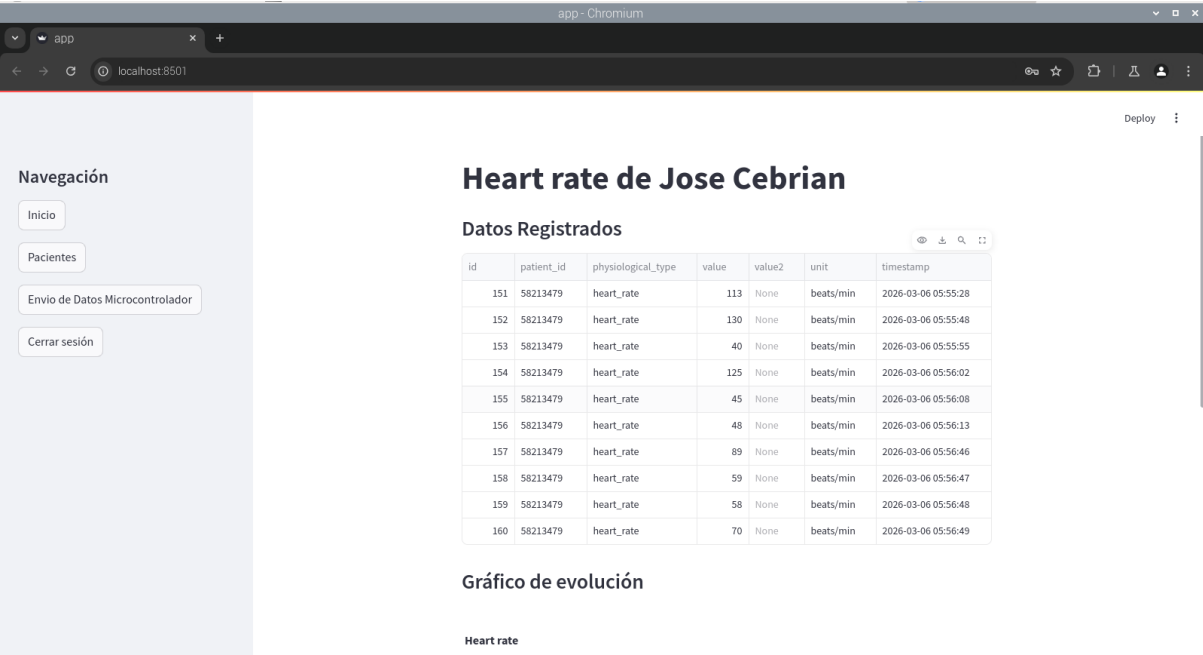
Fuente. Elaboración propia

Figura 80. Vista de la pantalla Data-view correspondiente a la presión arterial (Parte 2: Gráfica).



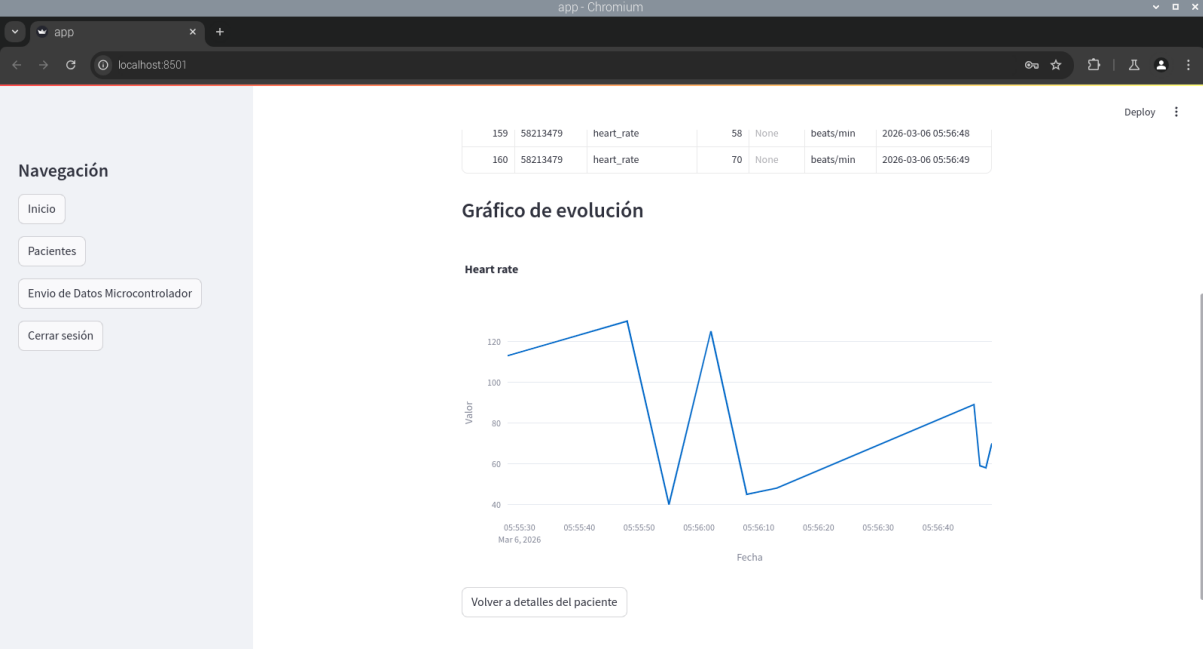
Fuente. Elaboración propia

Figura 81. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 1: Tabular).



Fuente. Elaboración propia

Figura 82. Vista de la pantalla Data-view correspondiente a la frecuencia cardíaca (Parte 2: Gráfica).



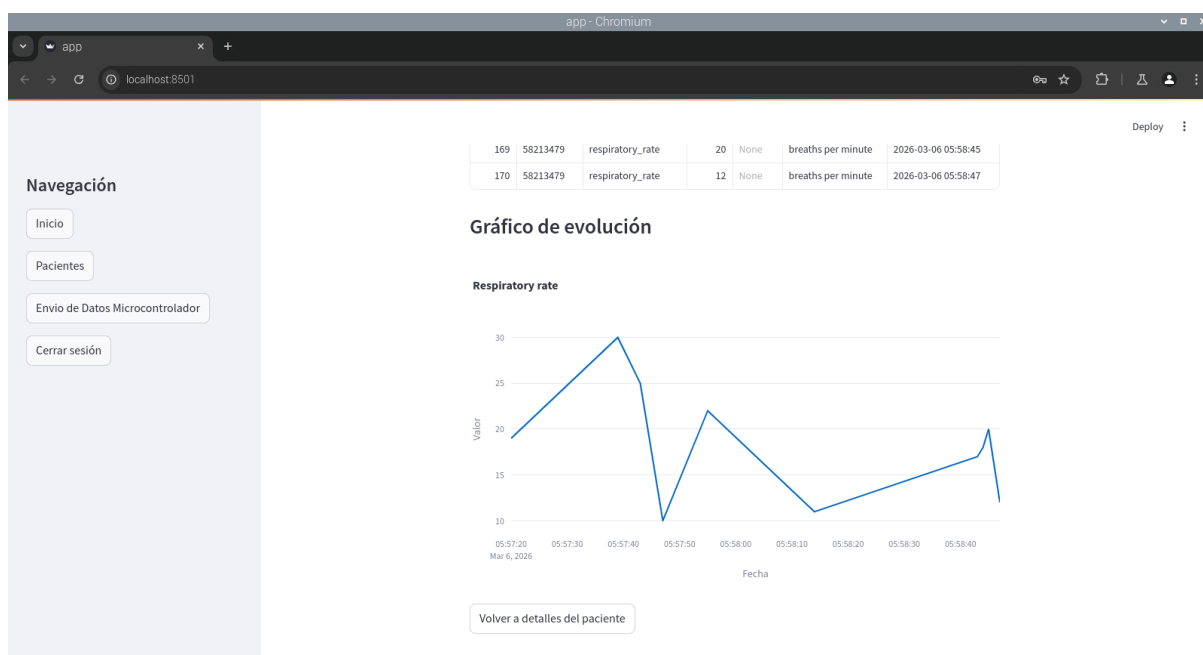
Fuente. Elaboración propia

Figura 83. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria
(Parte 1: Tabular)



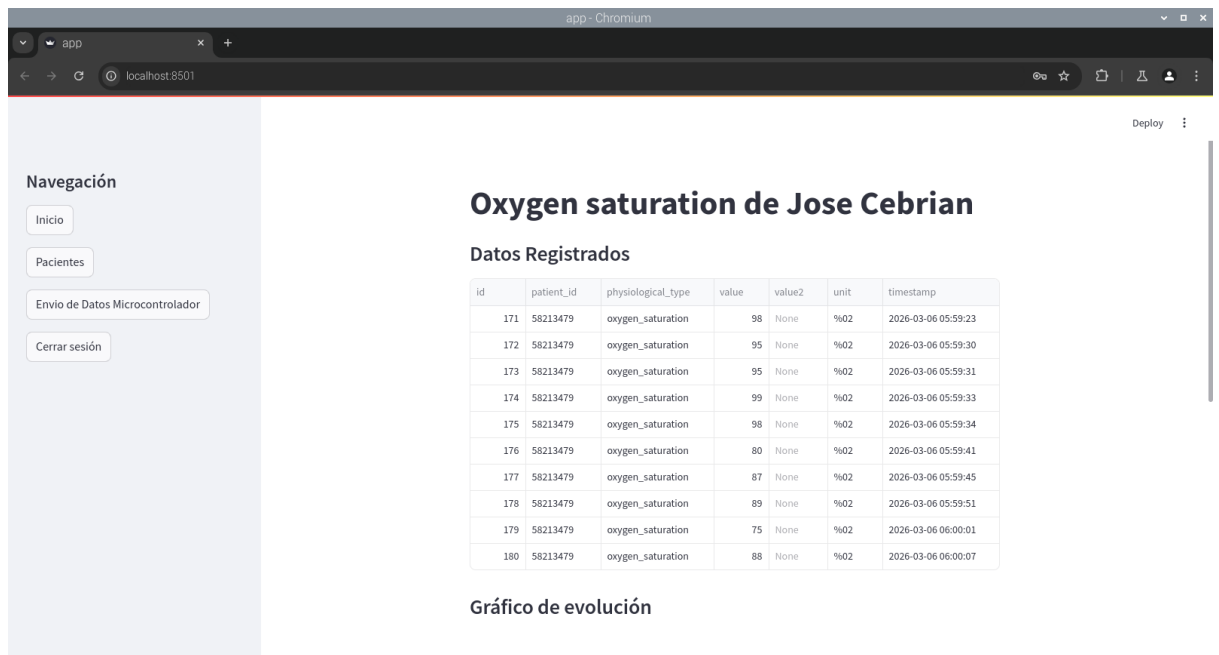
Fuente. Elaboración propia

Figura 84. Vista de la pantalla Data-view correspondiente a la frecuencia respiratoria
(Parte 2: Gráfica)



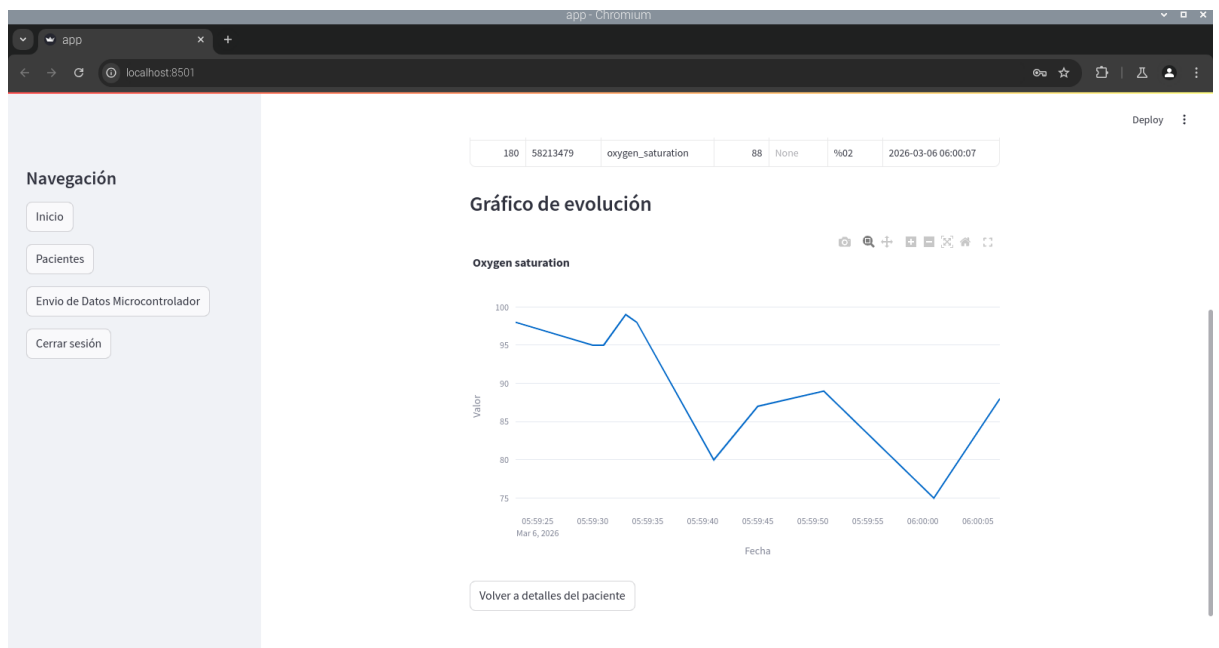
Fuente. Elaboración propia

Figura 85. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 1: Tabular).



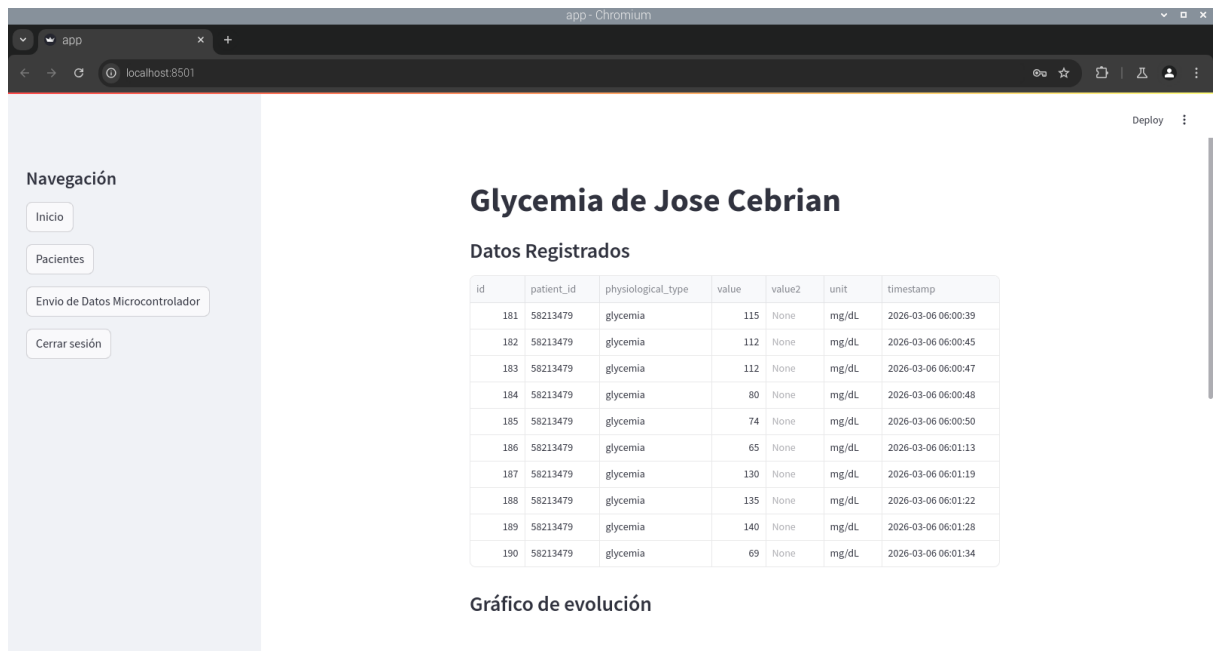
Fuente. Elaboración propia

Figura 86. Vista de la pantalla Data-view correspondiente a la saturación de oxígeno (Parte 2: Gráfica).



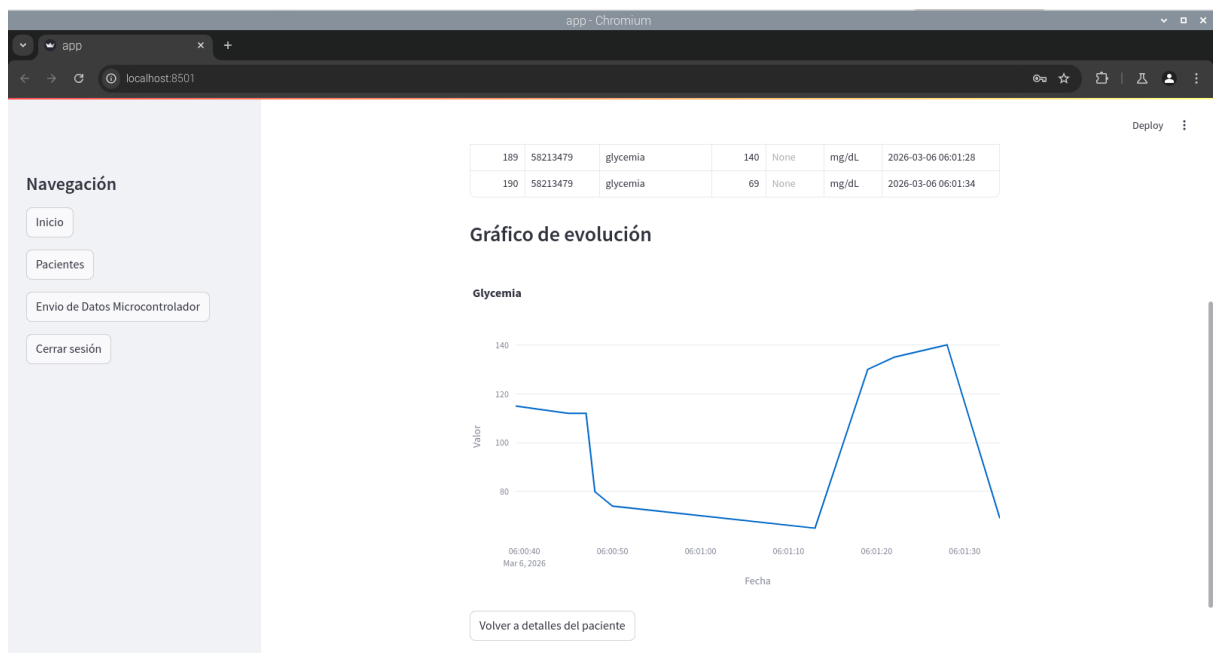
Fuente. Elaboración propia

Figura 87. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 1: Tabular)



Fuente. Elaboración propia

Figura 88. Vista de la pantalla Data-view correspondiente a la glicemia (Parte 2: Gráfica)



Fuente. Elaboración propia

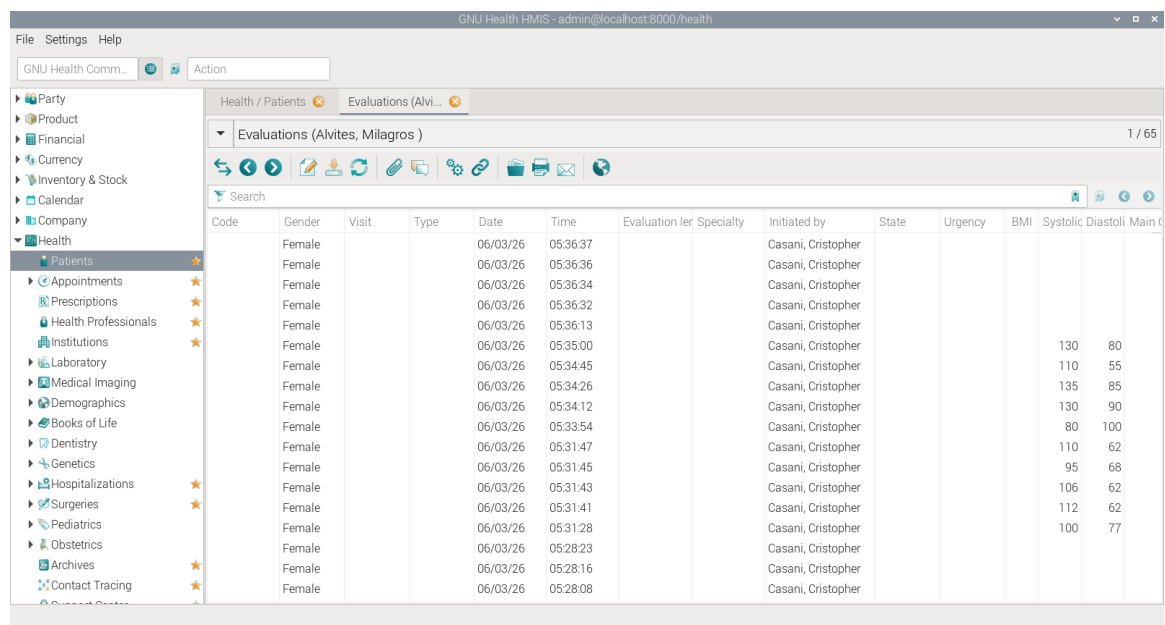
5.2.2 Trazabilidad en el sistema hospitalario (GNU Health)

Para concluir con la evaluación de consistencia y segregación de datos, se auditó el almacenamiento y correcta asignación de la información en el entorno GNU Health. A través de la aplicación de escritorio GNU Health Client, se accedió a la sección de "Evaluaciones" de cada uno de los pacientes configurados.

Dada la densidad del protocolo experimental (120 registros en total), la presentación de la evidencia visual se ha enfocado estratégicamente en un segmento de la trama de inserción que expone las mediciones de la presión arterial. Los registros adyacentes observados en las capturas corresponden a las transmisiones de las demás variables fisiológicas, cuyo detalle clínico completo es accesible al ingresar al registro individual de cada evaluación.

Cabe destacar que los valores clínicos documentados en esta vista nativa del sistema hospitalario coinciden de forma exacta con los presentados previamente en los paneles de la interfaz web en Streamlit. Además, se observa cómo en cada caso está registrado el médico configurado para cada paciente. A continuación, se presenta el registro de datos correspondientes a la presión arterial del Perfil A (Configuración 1 - Paciente: Milagros Alvites)

Figura 89. Vista de la sección de evaluaciones de la paciente Milagros Alvites en GNU Health Client.



Code	Gender	Visit	Type	Date	Time	Evaluation level	Specialty	Initiated by	State	Urgency	BMI	Systolic	Diastolic	Main
	Female			06/03/26	05:36:37			Casani, Cristopher						
	Female			06/03/26	05:36:36			Casani, Cristopher						
	Female			06/03/26	05:36:34			Casani, Cristopher						
	Female			06/03/26	05:36:32			Casani, Cristopher						
	Female			06/03/26	05:36:13			Casani, Cristopher						
	Female			06/03/26	05:35:00			Casani, Cristopher				130	80	
	Female			06/03/26	05:34:45			Casani, Cristopher				110	55	
	Female			06/03/26	05:34:26			Casani, Cristopher				135	85	
	Female			06/03/26	05:34:12			Casani, Cristopher				130	90	
	Female			06/03/26	05:33:54			Casani, Cristopher				80	100	
	Female			06/03/26	05:31:47			Casani, Cristopher				110	62	
	Female			06/03/26	05:31:45			Casani, Cristopher				95	68	
	Female			06/03/26	05:31:43			Casani, Cristopher				106	62	
	Female			06/03/26	05:31:41			Casani, Cristopher				112	62	
	Female			06/03/26	05:31:28			Casani, Cristopher				100	77	
	Female			06/03/26	05:28:23			Casani, Cristopher						
	Female			06/03/26	05:28:16			Casani, Cristopher						
	Female			06/03/26	05:28:08			Casani, Cristopher						

Fuente: Elaboración propia.

De la misma forma, se expone la validación para el Perfil B (Configuración 2 - Paciente: Jose Cebrian), confirmando la correcta segregación de datos por paciente:

Figura 90. Vista de la sección de evaluaciones del paciente Jose Cebrian en GNU Health Client.

Code	Gender	Visit	Type	Date	Time	Evaluation let Specie	Initiated by	State	Urgency	BMI	Systolic	Diastolic	Main Cond
	Male			06/03/26	05:55:47		Lonconi, Richard						
	Male			06/03/26	05:55:27		Lonconi, Richard						
	Male			06/03/26	05:53:58		Lonconi, Richard				140	90	
	Male			06/03/26	05:53:44		Lonconi, Richard				85	55	
	Male			06/03/26	05:53:28		Lonconi, Richard				80	50	
	Male			06/03/26	05:53:00		Lonconi, Richard				140	85	
	Male			06/03/26	05:51:22		Lonconi, Richard				94	60	
	Male			06/03/26	05:51:21		Lonconi, Richard				104	70	
	Male			06/03/26	05:51:20		Lonconi, Richard				105	65	
	Male			06/03/26	05:51:19		Lonconi, Richard				110	68	
	Male			06/03/26	05:50:48		Lonconi, Richard				120	63	
	Male			06/03/26	05:50:16		Lonconi, Richard						
	Male			06/03/26	05:50:08		Lonconi, Richard						
	Male			06/03/26	05:49:57		Lonconi, Richard						
	Male			06/03/26	05:49:52		Lonconi, Richard						
	Male			06/03/26	05:49:45		Lonconi, Richard						
	Male			06/03/26	05:49:33		Lonconi, Richard						
	Male			06/03/26	05:49:32		Lonconi, Richard						

Fuente: Elaboración propia.

VI. Discusión

La presente investigación se desarrolló con el propósito de abordar una problemática identificada en el ámbito académico: la falta de integración entre prototipos biomédicos desarrollados por estudiantes de ingeniería biomédica y HCE de acceso libre. Al analizar los hallazgos expuestos en el Capítulo V, se evidencia que el sistema diseñado, teniendo como núcleo una API REST y los protocolos MQTT y HTTP, permitió gestionar el ciclo completo del dato fisiológico durante las condiciones experimentales evaluadas, desde su captura en el sistema embebido hasta su almacenamiento en plataformas hospitalarias como GNU Health.

En contraste con los enfoques tradicionales donde el hardware se limita a emitir valores crudos, los resultados de las pruebas de consistencia evidenciaron un avance importante: el nodo simulador puede ser configurado de forma remota para estructurar la información nativamente bajo el estándar HL7 FHIR (formato JSON). Por otro lado, el hecho de que el sistema haya logrado enrutar y persistir la información correctamente ante un protocolo sistemático de 120 transmisiones distribuidas en 24 escenarios combinatorios en un entorno controlado, sin registrar pérdidas de información ni asignaciones cruzadas de datos clínicos, evidencia que la arquitectura propuesta mantuvo un flujo de interoperabilidad funcional y consistente bajo las condiciones de evaluación planteadas. Estos resultados permiten establecer una base tecnológica orientada a facilitar la integración futura de prototipos biomédicos estudiantiles con una HCE de acceso libre. De este modo, la arquitectura desarrollada constituye un modelo de integración orientado al ámbito académico e implementado mediante tecnologías abiertas y hardware de bajo costo, cuya relevancia cobra especial importancia al contrastar la solución propuesta con los enfoques arquitectónicos documentados en el estado del arte. Antes de realizar dicho contraste, resulta pertinente analizar el significado de los resultados experimentales obtenidos durante la validación funcional del sistema.

6.1 Análisis de la validación funcional de la arquitectura propuesta

Al analizar las pruebas realizadas, tanto la prueba unitaria como la prueba de segregación y consistencia de datos, los resultados obtenidos evidencian que la arquitectura planteada presentó un comportamiento funcional consistente en entornos controlados en todas las fases de transmisión del dato fisiológico sin observar pérdidas de información ni asignaciones cruzadas durante las pruebas realizadas. La ejecución satisfactoria de las

120 transmisiones distribuidas en los 24 escenarios definidos mediante la metodología Input Space Partitioning (ISP) y el criterio All Combinations Coverage (ACoC) (58), descritos en el Marco Teórico y aplicados en el diseño experimental, evidencia que el sistema mantuvo un comportamiento funcional consistente bajo las condiciones experimentales establecidas. Asimismo, la ausencia de asignaciones cruzadas entre los dos perfiles clínicos evaluados demuestra que el mecanismo de direccionamiento implementado mediante la API preservó correctamente la asociación entre paciente, médico, dispositivo y recurso clínico durante toda la validación experimental.

La arquitectura dual compuesta por Merry y Sunny permitió distribuir las funciones de recepción y procesamiento dentro del flujo de interoperabilidad, mientras que la estructuración del recurso HL7 FHIR se realizó directamente en el nodo simulador ESP32. En consecuencia, los componentes intermedios no requirieron realizar la transformación semántica del dato. No obstante, dado que no se efectuaron pruebas comparativas de rendimiento frente a otras arquitecturas, los resultados obtenidos no permiten determinar si esta distribución de responsabilidades reduce la carga computacional o la aparición de posibles cuellos de botella.

Además, la interoperabilidad alcanzada en este trabajo no se limita al intercambio sintáctico de mensajes JSON, sino que incorpora interoperabilidad semántica mediante la estructuración nativa de la información conforme al recurso Observation del estándar HL7 FHIR. Esto permite que los datos fisiológicos sean estructurados desde el nodo simulador y posteriormente procesados por la API para su almacenamiento en GNU Health, sin requerir una transformación adicional al formato FHIR en las capas intermedias

Por otro lado, la ejecución de cinco transmisiones consecutivas para cada uno de los 24 escenarios evaluados permitió observar la repetibilidad del comportamiento funcional del sistema durante transmisiones sucesivas. Los resultados obtenidos mostraron consistencia en el procesamiento y almacenamiento de la información bajo las condiciones controladas del estudio. Estas repeticiones fueron utilizadas para evaluar la repetibilidad de cada escenario y no constituyen una determinación estadística de suficiencia muestral.

No obstante, los resultados obtenidos deben interpretarse dentro del alcance del diseño experimental. Las pruebas fueron realizadas en un entorno controlado de laboratorio utilizando dos perfiles clínicos y una infraestructura de red local, por lo que no representan escenarios de operación con múltiples dispositivos concurrentes, degradación

de la red o cargas elevadas de procesamiento. En consecuencia, futuras investigaciones podrán complementar esta validación mediante pruebas de rendimiento, escalabilidad y tolerancia a fallos en entornos más próximos a la práctica clínica.

6.2 Comparación con el Estado del Arte

6.2.1 Comparación con Sistemas Basados en Blockchain (Categoría A)

Al comparar la solución planteada en la presente tesis con los estudios de la Categoría A (sistemas basados en blockchain), se evidencian diferencias fundamentales en el enfoque de interoperabilidad. En primer lugar, la mayoría de las investigaciones en esta categoría emplean blockchain principalmente como un mecanismo de gestión de acceso y seguridad; es decir, la red descentralizada se utiliza para intercambiar credenciales o permisos, delegando el intercambio de la información clínica a bases de datos externas. Esta estrategia busca evitar la sobrecarga de la red y mitigar la latencia. En particular, el estudio desarrollado por Bae et al. (23) propone una plataforma blockchain que gestiona el consentimiento dinámico del paciente, permitiendo compartir los permisos necesarios para acceder a datos clínicos a través de APIs institucionales. En contraste, nuestro sistema no emplea blockchain, sino que prioriza la transmisión directa y en tiempo real de los datos fisiológicos mediante el protocolo MQTT de baja latencia. Asimismo, a diferencia del enfoque de Bae et al. (23), nuestra arquitectura aprovecha la versatilidad del microcontrolador para estructurar nativamente los datos bajo el estándar HL7 FHIR desde el origen (previa configuración remota), sin requerir un middleware de transformación semántica adicional.

En segundo lugar, se observa una diferencia notoria en el nivel de integración de la información. Si bien Bae et al. (23) logran compartir información entre instituciones utilizando el estándar HL7 FHIR, los datos transmitidos no se inyectan directamente en el sistema transaccional de la institución receptora. En este sentido, la arquitectura desarrollada en esta tesis da un paso adicional al lograr el almacenamiento automatizado de los recursos compartidos dentro del sistema de información GNU Health, permitiendo que el receptor tenga la información disponible de forma nativa para su uso clínico, y no solo para su visualización.

No obstante, es imperativo reconocer una limitación respecto a la trazabilidad en comparación con la literatura. Mientras que el estudio de Bae et al. (23) garantiza un intercambio validado mediante blockchain y el consentimiento dinámico del paciente,

nuestro proyecto aborda la trazabilidad a través de una interfaz de auditoría visual (Streamlit). Dado que la seguridad de acceso a dicha herramienta es básica y la información clínica intercambiada es altamente sensible, se identifica como línea de trabajo futuro la implementación de capas de seguridad más robustas, tanto en el control de acceso al dashboard como en el cifrado de la comunicación MQTT.

Por último, desde el punto de vista de la validación empírica, Bae et al. (23) desarrollaron pruebas de usabilidad que arrojaron resultados positivos en términos de facilidad de uso, utilidad percibida y satisfacción general. Al ser nuestra propuesta un sistema concebido específicamente para resolver un vacío tecnológico en estudiantes, resulta evidente la necesidad de validar la aceptación de la herramienta por parte de este grupo demográfico. Por consiguiente, se sugiere para futuras investigaciones el diseño y ejecución de estudios de usabilidad que evalúen empíricamente la experiencia de los estudiantes de ingeniería al interactuar con el sistema propuesto.

6.2.2 Comparación con Middlewares de Estandarización IoT (Categoría B)

Con relación a los sistemas de integración y estandarización de datos (Categoría B), la revisión de la literatura evidencia una tendencia hacia el uso de middlewares centralizados para la transformación semántica. En particular, el estudio "BIOTHR: Electronic Health Record Servicing Scheme in IoT-Blockchain Ecosystem" desarrollado por Ray et al. (28), plantea una arquitectura donde los datos crudos emitidos por los nodos IoT son normalizados en un gateway. Posteriormente, un middleware transforma estos datos al estándar HL7 FHIR y los almacena en un servidor FHIR dedicado. Al contrastar este modelo con la arquitectura propuesta en la presente tesis, se identifican diferencias principalmente en la distribución de responsabilidades entre los componentes y en el destino final de la información.

En primer lugar, Ray et al. (28) delegan la estandarización del dato a las capas intermedias de la red. En contraste, en la arquitectura desarrollada en la presente investigación, el propio nodo emisor (ESP32) es el encargado de estructurar el formato HL7 FHIR antes de la transmisión. De este modo, la transformación semántica no es realizada por el middleware. Asimismo, las funciones de recepción y procesamiento se distribuyen entre los microservicios Merry y Sunny, respectivamente. Esta organización representa una distribución distinta de responsabilidades respecto a arquitecturas donde la normalización y transformación se concentran en una capa intermedia. Sin embargo, debido a que en la presente investigación no se realizaron pruebas comparativas de rendimiento entre ambas

arquitecturas, no es posible establecer si esta distribución representa una reducción de carga computacional o una disminución de posibles cuellos de botella.

En segundo lugar, existe una diferencia en el destino final del dato. Es importante señalar que Ray et al. (28) almacenan la información en un servidor FHIR interoperable, lo cual le otorga a su arquitectura una versatilidad a futuro, permitiendo que múltiples sistemas de información clínica se adhieran a dicho servidor para consumir los datos. La propuesta de esta tesis, por su parte, no implementa un servidor FHIR intermedio, sino que utiliza los recursos FHIR como estructura de mensajería para posteriormente registrar la información en la base de datos de GNU Health. Esta decisión permitió validar el registro efectivo de la información clínica dentro de una HCE funcional en un entorno experimental.

Finalmente, ambas propuestas requieren evaluaciones adicionales para determinar su comportamiento en escenarios de mayor exigencia. En futuras investigaciones sería necesario realizar pruebas de estrés y rendimiento, así como evaluaciones específicas de seguridad, con el fin de caracterizar de manera más completa la robustez de este tipo de arquitecturas de interoperabilidad.

6.3 Relación de la propuesta con el marco normativo peruano

Dado que el sistema desarrollado fue concebido como una herramienta para fomentar la interoperabilidad en el ámbito académico dentro del contexto nacional peruano, resulta pertinente relacionar el diseño de la arquitectura propuesta con el marco normativo vigente en materia de historias clínicas electrónicas. En este sentido, la Ley N.º 30024, promulgada por el Estado peruano, crea el Registro Nacional de Historias Clínicas Electrónicas (RENHICE), concebido como una infraestructura tecnológica administrada por el Ministerio de Salud (MINSA) para facilitar el acceso e intercambio de información clínica entre establecimientos de salud autorizados (63). En relación con este objetivo, el sistema desarrollado en la presente tesis implementa un mecanismo de interoperabilidad basado en APIs REST y el estándar HL7 FHIR, permitiendo el intercambio de información clínica entre sistemas heterogéneos, representados en este trabajo por el nodo simulador ESP32 y la HCE GNU Health.

Por otro lado, uno de los objetivos del RENHICE es promover la estandarización de la información clínica para favorecer la interoperabilidad entre los sistemas de información en salud (63). En concordancia con este principio, la arquitectura propuesta emplea el

estándar internacional HL7 FHIR para estructurar los recursos clínicos generados por el nodo simulador ESP32 antes de su transmisión. Esta decisión permite incorporar interoperabilidad semántica desde el origen de los datos y proporciona una base para una futura integración con otros sistemas que adopten el mismo estándar.

No obstante, la presente investigación no implementa una conexión directa con el RENHICE, sino que desarrolla una herramienta con fines académicos alineada con los principios generales de interoperabilidad promovidos por dicho marco normativo. En consecuencia, el sistema no incorpora mecanismos institucionales contemplados en la Ley N.º 30024, tales como la integración con la infraestructura nacional del RENHICE, mecanismos de autenticación, firma digital o intercambio interinstitucional de información clínica (63). Estos aspectos deberán considerarse en futuras investigaciones orientadas a adaptar la arquitectura propuesta para su eventual implementación en escenarios asistenciales reales dentro del ecosistema de salud digital peruano.

6.4 Limitaciones del estudio

A pesar de los resultados obtenidos, la solución desarrollada presenta ciertas limitaciones inherentes a su fase de prototipado:

1. Las pruebas de transmisión de extremo a extremo se realizaron en un entorno de red controlado, utilizando exclusivamente el microcontrolador ESP32 como nodo simulador. Para determinar la conducta en producción, se requerirán pruebas de estrés con múltiples microcontroladores simultáneos y la medición formal de latencia y throughput de red.
2. La validación funcional de la arquitectura se realizó utilizando datos fisiológicos sintéticos generados por el microcontrolador ESP32 (nodo simulador). En consecuencia, el estudio no incluyó la adquisición de señales fisiológicas provenientes de sensores biomédicos reales, debido a que el objetivo de la investigación fue validar el flujo de interoperabilidad de la arquitectura y no el desempeño de los dispositivos de adquisición. Por consiguiente, futuras investigaciones podrán integrar módulos de adquisición de señales fisiológicas reales para evaluar el comportamiento del sistema frente a la variabilidad inherente de las señales biológicas y validar su funcionamiento en escenarios más cercanos a la práctica clínica.

3. La validación funcional de la arquitectura se realizó exclusivamente mediante el nodo simulador ESP32 desarrollado como parte de la presente investigación y no incluyó la integración de prototipos biomédicos desarrollados independientemente por otros estudiantes. En consecuencia, los resultados obtenidos permiten validar el funcionamiento del flujo de interoperabilidad implementado, pero no permiten generalizar su compatibilidad o facilidad de integración con diferentes prototipos estudiantiles. En este sentido, sería conveniente que futuras investigaciones evalúen la arquitectura mediante la integración de prototipos externos desarrollados por estudiantes, considerando diferentes configuraciones de hardware, sensores y estructuras de datos.
4. La inyección remota de parámetros hacia los dispositivos IoT (como el ID del paciente, del médico y del dispositivo) debe realizarse de manera estrictamente secuencial. El sistema aún no soporta la configuración paralela o concurrente de múltiples nodos de forma simultánea, ya que esto podría derivar en cruces de información o asignaciones cruzadas no deseadas durante la fase de inicio.
5. La protección de datos es prioritaria en entornos clínicos. Actualmente, el sistema cuenta con autenticación básica y hashing de contraseñas para la interfaz web. Sin embargo, carece de mecanismos de encriptación de extremo a extremo (ej. TLS/SSL para las tramas MQTT), lo cual es una vulnerabilidad ante la posible interceptación de datos sensibles.
6. La API principal expone métodos REST esenciales (GET, POST, PATCH) para la gestión del flujo actual. Sin embargo, no se han desarrollado métodos de eliminación (DELETE) ni de corrección avanzada, los cuales serían necesarios en caso de requerir la anulación de registros clínicos erróneos bajo protocolos de auditoría hospitalaria.
7. El sistema actual solo admite la transmisión de seis tipos de variables fisiológicas puntuales, cuya ampliación de más tipos sería clave en proyectos futuros. Además, dado que la herramienta está dirigida a estudiantes de ingeniería biomédica, queda pendiente realizar evaluaciones cuantitativas de usabilidad (UX/UI) y satisfacción del usuario final para validar su adopción académica.
8. El sistema utiliza el estándar HL7 FHIR exclusivamente como modelo de estructuración de mensajería (payload) para permitir la interoperabilidad semántica desde el nodo simulador. Sin embargo, el sistema no implementa un

servidor FHIR RESTful nativo. Por lo tanto, las referencias internas en los archivos JSON generados (como Patient/{id} o Device/{id}) actúan como identificadores lógicos para el enrutamiento interno de la API hacia GNU Health, y no como recursos URI resolubles de manera independiente.

9. La estrategia de integración empleada realiza el registro de la información mediante inserciones directas en la base de datos PostgreSQL de GNU Health. Esta decisión permitió validar el flujo completo de interoperabilidad dentro del entorno experimental de la presente investigación. No obstante, su funcionamiento depende de la estructura interna de la base de datos del HMIS. En consecuencia, futuras actualizaciones del esquema de GNU Health podrían requerir ajustes en la lógica de integración desarrollada. Como línea de trabajo futuro, se recomienda evaluar mecanismos oficiales de interoperabilidad o servicios de aplicación que reduzcan el nivel de acoplamiento con la estructura interna del sistema y favorezcan una mayor mantenibilidad y escalabilidad de la solución

VII. Conclusiones

A partir del desarrollo, implementación y validación del presente proyecto de investigación, se establecen las siguientes conclusiones:

- ❖ Se logró desarrollar e implementar una API y un sistema integral orientado a la interoperabilidad estandarizada entre prototipos biomédicos estudiantiles e HCE de acceso libre, específicamente GNU Health. La validación experimental de la propuesta se realizó mediante un único nodo simulador ESP32 desarrollado en la presente investigación. La solución desarrollada constituye una herramienta para incorporar capacidades de interoperabilidad durante el desarrollo de prototipos, utilizando los protocolos de comunicación MQTT y HTTP, así como el recurso Observation del estándar HL7 FHIR.
- ❖ Se construyó de forma exitosa un firmware en C++ para un sistema embebido (ESP32) que opera como nodo de adquisición IoT. Este nodo simulador demostró la capacidad de generar datos fisiológicos (tanto en modalidad aleatoria como personalizada), estructurarlos nativamente como recursos Observation en formato JSON siguiendo el estándar HL7 FHIR (permitiendo la interoperabilidad semántica) y transmitirlos dinámicamente vía MQTT. Para ello, se implementó un procesamiento de estructuras JSON, la sincronización de marcas de tiempo (timestamps) mediante servidores NTP y la lógica de recepción para la configuración remota de metadatos clínicos.
- ❖ Se diseñó e implementó una arquitectura de backend dual y desacoplada compuesta por dos microservicios: Merry y Sunny Health API. Esta decisión arquitectónica permitió distribuir las funciones de recepción y procesamiento entre ambos microservicios; de modo que Merry se mantuvo en escucha constante como middleware MQTT, mientras que Sunny actuó bajo demanda (vía peticiones REST POST) para procesar el mensaje FHIR. Asimismo, la lógica implementada automatizó la ingesta de la información hacia dos destinos: MariaDB, utilizada como base de datos espejo, y la base de datos PostgreSQL de GNU Health, respetando la estructura relacional requerida para el almacenamiento de la información clínica.
- ❖ Se diseñó y desplegó una plataforma web interactiva en Streamlit sustentada por una base de datos espejo (MariaDB). Esta herramienta permitió el control remoto

bidireccional del nodo simulador, así como la visualización y trazabilidad de los datos clínicos recibidos. Al derivar las consultas de visualización hacia esta base de datos espejo, se evitaron accesos continuos a la base de datos PostgreSQL de GNU Health durante el uso de la interfaz.

- ❖ Las pruebas de validación funcional de extremo a extremo evidenciaron un comportamiento consistente del sistema durante la transmisión, procesamiento y almacenamiento de la información clínica. La prueba de trazabilidad unitaria verificó que la carga útil generada por el nodo simulador se preservó íntegramente hasta su almacenamiento en GNU Health y su visualización en Streamlit. Asimismo, la ejecución de 120 transmisiones correspondientes a 24 escenarios experimentales definidos mediante la metodología ISP y el criterio ACoC permitió verificar la correcta segregación de la información conforme a los parámetros configurados en el nodo simulador, sin observar pérdidas de información ni asignaciones cruzadas de datos clínicos bajo las condiciones controladas del estudio.

VIII. Referencias Bibliográficas

1. Clínica Universidad de Navarra. <https://www.cun.es> [Internet]. [citado 31 de enero de 2026]. Qué es la historia clínica. Diccionario médico. Clínica U. Navarra. Disponible en: <https://www.cun.es/diccionario-medico/terminos/historia-clinica>
2. Cervera-Barba E, Denizon-Arranz S, Mateos-Rodríguez A, Neria-Serrano F. Concordancia y calidad de las historias clínicas en los alumnos de Medicina: ¿reflejan la realidad de la consulta? Educ Médica. 1 de mayo de 2024;25(3):100907. doi:10.1016/j.edumed.2024.100907
3. Fernandez Leon EM. Desarrollo de un sistema Web de historias clínicas electrónicas para mejorar la gestión en el proceso de consultas ambulatorias del Hospital Belén de Lambayeque [Internet]. [Chiclayo]: Universidad Católica Santo Toribio de Mogrovejo; 2022. Disponible en: <http://hdl.handle.net/20.500.12423/4532>
4. Bernabe Cordova LMV, Piscocoya Bances JR. Sistema móvil para la gestión de historias clínicas en un policlínico de Lima Norte en el año 2024 [Internet]. [Lima, Perú]: Universidad Tecnológica del Perú; 2025. Disponible en: <https://hdl.handle.net/20.500.12867/11995>
5. Calderon Chavez ME, Guerrero Neyra N. Implementación de la tecnología cloud computing para las historias clínicas en el Instituto Médico de la Mujer 2023 [Internet]. [Lima, Perú]: Universidad Peruana de Ciencias Aplicadas (UPC); 2023. Disponible en: <http://hdl.handle.net/10757/670771>
6. ISO. La historia clínica electrónica explicada [Internet]. [citado 31 de enero de 2026]. La historia clínica electrónica explicada. Disponible en: <https://www.iso.org/es/asistencia-sanitaria/historia-clinica-electronica>
7. Coico Lama AH, Diaz Chingay LL. Beneficios de la implementación de un registro electrónico de pacientes de cuidados intensivos: una revisión de alcance [Internet]. [Lima, Perú]: Universidad Peruana Unión; 2021 [citado 21 de noviembre de 2025]. Disponible en: <https://hdl.handle.net/20.500.12840/4661>
8. Añel Rodríguez RM, García Alfaro I, Bravo Toledo R, Carballeira Rodríguez JD. Historia clínica y receta electrónica: riesgos y beneficios detectados desde su implantación. Diseño, despliegue y usos seguros. Vol. 53. 1 de diciembre de 2021;53:102220. doi:10.1016/j.aprim.2021.102220
9. MINSA. gob.pe [Internet]. 2025 [citado 31 de enero de 2026]. Decreto Supremo N.º 005-2025-SA. Disponible en: <https://www.gob.pe/institucion/minsa/normas-legales/6820724-005-2025-sa>
10. MINSA. gob.pe [Internet]. 2025 [citado 31 de enero de 2026]. Resolución Ministerial N.º 553-2025-MINSA. Disponible en: <https://www.gob.pe/institucion/minsa/normas-legales/7098166-553-2025-minsa>
11. Huapaya-Huertas O, Palomino-Rojas J, Calle-Teixeira C, Alvarez-Huiman G, Montesinos-Segura R, Taype-Rondan A, et al. Experiencia del Complejo Hospitalario

San Pablo (Perú) en la implementación de un sistema de historias clínicas electrónicas. *An Fac Med.* octubre de 2021;82(4):349-54. doi:10.15381/anales.v82i4.22211

12. Martínez V, Rodrich R, Soto M. Propuesta de salud al bicentenario: La historia clínica electrónica [Internet]. julio de 2024 [citado 21 de noviembre de 2025]. Disponible en: <https://repositorio.pucp.edu.pe/index/handle/123456789/200345>

13. MINSA. gob.pe [Internet]. 2025 [citado 21 de noviembre de 2025]. ¡Hito histórico! Minsa dio primer gran paso hacia la interoperabilidad entre sistemas de información de historias clínicas electrónicas. Disponible en: <https://www.gob.pe/institucion/minsa/noticias/1190113-hito-historico-minsa-dio-primer-gran-paso-hacia-la-interoperabilidad-entre-sistemas-de-informacion-de-historias-clinicas-electronicas>

14. Gómez Burgos FM. Tecnologías Emergentes en Salud: Implementación de tecnologías emergentes como IoT y big data en sistemas de información de salud para mejorar la atención al paciente y la gestión de recursos en Perú. *Rev Vínculos* [Internet]. 13 de noviembre de 2025 [citado 4 de febrero de 2026];21(2). Disponible en: <https://revistas.udistrital.edu.co/index.php/vinculos/article/view/23873>

15. Mejia torres SJ, Jiménez AM, Ramírez EM, Ovalle Santamaría CS, Acevedo Peñaranda JA. Internet de las cosas (IoT) y sistemas interconectados para mejorar la seguridad del paciente y la automatización de alertas en tiempo real, con integración de IA aplicada a la monitorización continua y al análisis de imágenes médicas [Internet]. 2025. Disponible en: <https://repository.unad.edu.co/handle/10596/77375>

16. Ahmed SF, Sharmin S, Kuldeep SA, Lameesa A, Alam MdSB, Liu G, et al. Transformative impacts of the internet of medical things on modern healthcare. *Results Eng.* 1 de marzo de 2025;25:103787. doi:10.1016/j.rineng.2024.103787

17. Li C, Wang J, Wang S, Zhang Y. A review of IoT applications in healthcare. *Neurocomputing.* 2024;565:127017. doi:<https://doi.org/10.1016/j.neucom.2023.127017>

18. Hernandez Forero P, Cardorna Lozano BA, Rivera Ortiz P. Efectos de los datos erróneos en la historia clínica en la prestación de los servicios de salud [Internet]. Universidad Ean; 2024. Disponible en: <http://hdl.handle.net/10882/14291>

19. Daza Rojas JJ. Modelo de computación en la nube para el telemonitoreo de signos vitales en tiempo real [Internet]. 2023. Disponible en: <http://hdl.handle.net/11349/39706>

20. Falcon L. <https://docs.gnuhealth.org/thalamus/> [Internet]. [citado 21 de noviembre de 2025]. Thalamus - The GNU Health Message and Authentication Server. Disponible en: <https://docs.gnuhealth.org/thalamus/>

21. Sonkamble RG, Phansalkar SP, Potdar VM, Bongale AM. Survey of Interoperability in Electronic Health Records Management and Proposed Blockchain Based Framework: MyBlockEHR. *IEEE Access.* 2021;9:158367-401. doi:10.1109/ACCESS.2021.3129284

22. Zhuang Y, Sheets LR, Chen YW, Shae ZY, Tsai JJP, Shyu CR. A Patient-Centric Health Information Exchange Framework Using Blockchain Technology. *IEEE J Biomed Health Inform.* 2020;24(8):2169-76. doi:10.1109/JBHI.2020.2993072
23. Bae YS, Park Y, Lee SM, Seo HH, Lee H, Ko T, et al. Development of Blockchain-Based Health Information Exchange Platform Using HL7 FHIR Standards: Usability Test. *IEEE Access.* 2022;10:79264-71. doi:10.1109/ACCESS.2022.3194159
24. Akkaoui R, Hei X, Cheng W. EdgeMediChain: A Hybrid Edge Blockchain-Based Framework for Health Data Exchange. *IEEE Access.* 2020;8:113467-86. doi:10.1109/ACCESS.2020.3003575
25. Pang Z, Yao Y, Li Q, Zhang X, Zhang J. Electronic Health Records Sharing Model Based on Blockchain With Checkable State PBFT Consensus Algorithm. *IEEE Access.* 2022;10:87803-15. doi:10.1109/ACCESS.2022.3186682
26. Mukhiya SK, Lamo Y. An HL7 FHIR and GraphQL approach for interoperability between heterogeneous Electronic Health Record systems. *Health Informatics J.* 1 de julio de 2021;27(3):14604582211043920. doi:10.1177/14604582211043920
27. Jaleel A, Mahmood T, Hassan MA, Bano G, Khurshid SK. Towards Medical Data Interoperability Through Collaboration of Healthcare Devices. *IEEE Access.* 2020;8:132302-19. doi:10.1109/ACCESS.2020.3009783
28. Ray PP, Chowhan B, Kumar N, Almogren A. BioTHR: Electronic Health Record Servicing Scheme in IoT-Blockchain Ecosystem. *IEEE Internet Things J.* 2021;8(13):10857-72. doi:10.1109/JIOT.2021.3050703
29. Malathi K, Nair S, Madhumitha N, Sreelakshmi S, Sathya U, Priya S. Medical Data Integration and Interoperability through Remote Monitoring of Healthcare Devices. Vol. 15. 30 de julio de 2024;15:60-72. doi:10.58346/JOWUA.2024.I2.005
30. PAHO. eHealth in Latin America and the Caribbean: interoperability standards review [Internet]. 2016 [citado 14 de febrero de 2026]. Disponible en: <https://iris.paho.org/handle/10665.2/28189>
31. Mason C. Wolters Kluwer [Internet]. 2025 [citado 14 de febrero de 2026]. Understand the four levels of interoperability in healthcare. Disponible en: <https://www.wolterskluwer.com/en/expert-insights/understand-the-four-levels-of-interoperability-in-healthcare>
32. Bodenreider O, Cornet R, Vreeman DJ. Recent Developments in Clinical Terminologies — SNOMED CT, LOINC, and RxNorm. *Yearb Med Inform.* agosto de 2018;27(1):129-39. doi:10.1055/s-0038-1667077 PubMed PMID: 30157516; PubMed Central PMCID: PMC6115234.
33. HL7 Perú. HL7 FHIR [Internet]. [citado 14 de febrero de 2026]. HL7 FHIR. Disponible en: <http://hl7peru.org/hl7-fhir>

34. Gazzarata R, Almeida J, Lindsköld L, Cangili G, Gaeta E, Fico G, et al. HL7 Fast Healthcare Interoperability Resources (HL7 FHIR) in digital healthcare ecosystems for chronic disease management: Scoping review. *Int J Med Inf.* 1 de septiembre de 2024;189:105507. doi:10.1016/j.ijmedinf.2024.105507
35. NIBIB. RADx® MARS: HL7v2 vs. FHIR | Instituto Nacional de Imágenes Biomédicas y Bioingeniería [Internet]. [citado 14 de febrero de 2026]. Disponible en: <https://www.nibib.nih.gov/programs/radx-tech-program/mars/hl7v2-fhir>
36. WillKinsman. What is FHIR? [Internet]. 2024 [citado 1 de marzo de 2026]. What is FHIR? - Tenasol. Disponible en: <https://www.tenasol.com/blog/explanation-of-hl7-fhir>
37. HL7 FHIR. Resource Observation - Content [Internet]. 2025 [citado 14 de febrero de 2026]. Observación - FHIR v6.0.0-ballot3. Disponible en: <https://build.fhir.org/observation.html>
38. AWS. Amazon Web Services, Inc. [Internet]. [citado 14 de febrero de 2026]. ¿Qué es el MQTT? - Explicación del protocolo MQTT - AWS. Disponible en: <https://aws.amazon.com/es/what-is/mqtt/>
39. Bayas Sampedro M, Parra Fernandez A, Rovira Jurado R, Montaña Blacio M, Gómez Morales O, Figueroa Olmedo J. Sistema de monitoreo de parámetros cardiovasculares basado en IoT y MQTT para alertas médicas. *Ingenius.* 1 de julio de 2025;(34):75-88. doi:10.17163/ings.n34.2025.06
40. Alshammari HH. The internet of things healthcare monitoring system based on MQTT protocol. *Alex Eng J.* 15 de abril de 2023;69:275-87. doi:10.1016/j.aej.2023.01.065
41. HiveMQ. MQTT Tutorial: An Easy Guide to Getting Started with MQTT [Internet]. 2026 [citado 1 de marzo de 2026]. Disponible en: <https://www.hivemq.com/blog/how-to-get-started-with-mqtt/>
42. GNU Solidario. Welcome to GNU Health! [Internet]. 2020 [citado 15 de febrero de 2026]. GNU Health | Freedom and Equity in Healthcare. Disponible en: <https://www.gnuhealth.org/about-us.html>
43. GNU Solidario. Features [Internet]. 2025 [citado 15 de febrero de 2026]. Features — GNU Health Hospital Information System documentation. Disponible en: <https://docs.gnuhealth.org/his/features.html>
44. GNU Solidario. First Steps [Internet]. 2025 [citado 15 de febrero de 2026]. First Steps — GNU Health Hospital Information System documentation. Disponible en: <https://docs.gnuhealth.org/his/userguide/basics/firststeps.html>
45. GNU Solidario. GNU Health Embedded [Internet]. 2025 [citado 15 de febrero de 2026]. GNU Health Embedded — GNU Health Hospital Information System documentation. Disponible en: <https://docs.gnuhealth.org/his/techguide/installation/embedded.html>

46. GNU Solidario. GNU Health/Installation [Internet]. 2024 [citado 22 de febrero de 2026]. GNU Health/Installation - Wikibooks, open books for an open world. Disponible en: https://en.wikibooks.org/wiki/GNU_Health/Installation
47. Garcia F. Arsys [Internet]. [citado 15 de febrero de 2026]. ¿Qué es Flask (Python) y cuáles son sus ventajas? | Blog de Arsys. Disponible en: <https://www.arsys.es/blog/que-es-flask-python-y-cuales-son-sus-ventajas>
48. Bienvenido a Flask — Flask Documentation (3.1.x) [Flask's documentation] [Internet]. 2024 [citado 15 de febrero de 2026]. Disponible en: <https://flask.palletsprojects.com/es/stable/>
49. Streamlit documentation [Internet]. 2026 [citado 15 de febrero de 2026]. Streamlit Docs. Disponible en: <https://docs.streamlit.io/>
50. Development concepts [Internet]. 2026 [citado 15 de febrero de 2026]. Development concepts - Streamlit Docs. Disponible en: <https://docs.streamlit.io/>
51. DataCamp. Tutorial de Python: Streamlit [Internet]. 2026 [citado 15 de febrero de 2026]. Tutorial de Python: Streamlit. Disponible en: <https://www.datacamp.com/tutorial/streamlit>
52. Goodwin Mi. ¿Qué es una API? | IBM [Internet]. 2024 [citado 15 de febrero de 2026]. Disponible en: <https://www.ibm.com/mx-es/think/topics/api>
53. IBM. ¿Qué es una API REST? [Internet]. 2025 [citado 15 de febrero de 2026]. ¿Qué es una API REST (API RESTful)? | IBM. Disponible en: <https://www.ibm.com/es-es/think/topics/rest-apis>
54. Díaz M. DEV Community [Internet]. 2024 [citado 1 de marzo de 2026]. Entendiendo las APIs RESTful y el Protocolo HTTP. Disponible en: <https://dev.to/milondiazco/entendiendo-las-apis-restful-y-el-protocolo-http-44a6>
55. Monción Rodríguez CL, Losilla Lopez F, Alcaraz Espin J jose. Diseño e implementación de una plataforma y API para la enseñanza de la tecnología REST [Trabajo Fin de Master] [Internet]. [Escuela Técnica Superior de Ingeniería de Telecomunicación]; Universidad Politécnica de Cartagena; 2023 [citado 15 de febrero de 2026]. Disponible en: <http://hdl.handle.net/10317/12841>
56. Espressif Systems. ESP32 Series Datasheet Version 5.2 [Internet]. Shanghai; 2025. Disponible en: https://documentation.espressif.com/esp32_datasheet_en.pdf
57. Naylamp Mechatronics. Naylamp Mechatronics - Perú [Internet]. 2023 [citado 27 de abril de 2026]. ESP32 DEVKIT V1 - NodeMCU-32 30-pin ESP32 WiFi micro-USB. Disponible en: <https://naylampmechatronics.com/espressif-esp/384-esp32-devkit-v1-nodemcu-32-30-pin-esp32-wifi-micro-usb.html>
58. Ammann P, Offutt J, editores. Input Space Partitioning. En: Introduction to Software Testing [Internet]. Cambridge: Cambridge University Press; 2008. p. 150-69. Located at: Cambridge Core. Disponible en:

<https://www.cambridge.org/core/product/417E407C3DE99D76D5E8AE4C953ACA1D> doi:10.1017/CBO9780511809163.005

59. Dias D, Paulo Silva Cunha J. Wearable Health Devices-Vital Sign Monitoring, Systems and Technologies. *Sensors*. 25 de julio de 2018;18(8). doi:10.3390/s18082414 PubMed PMID: 30044415; PubMed Central PMCID: PMC6111409.
60. Downey CL, Tahir W, Randell R, Brown JM, Jayne DG. Strengths and limitations of early warning scores: A systematic review and narrative synthesis. *Int J Nurs Stud*. 1 de noviembre de 2017;76:106-19. doi:10.1016/j.ijnurstu.2017.09.003
61. Instituto Nacional de Salud. INS advierte que más de 2 millones 500 mil peruanos sufre de diabetes y recomienda evaluaciones preventivas al menos una vez al año [Internet]. [citado 21 de julio de 2026]. Disponible en: <https://www.gob.pe/institucion/ins/noticias/1190543-ins-advierde-que-mas-de-2-millones-500-mil-peruanos-sufre-de-diabetes-y-recomienda-evaluaciones-preventivas-al-menos-una-vez-al-ano>
62. D. R. Kuhn, D. R. Wallace, A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans Softw Eng*. junio de 2004;30(6):418-21. doi:10.1109/TSE.2004.24
63. MINSA. Ley que Crea el Registro de Nacional de Historias Clínicas Electrónicas [Internet]. 2013 [citado 20 de julio de 2026]. Ley N.º 30024. Disponible en: <https://www.gob.pe/institucion/minsa/normas-legales/240527-30024>

Anexos

Anexo 1 : Tabla comparativa del estado del arte.

Tipo de Tecnología	Nombre del artículo	Metodología	Resultados	Ventajas	Limitaciones
Arquitectura híbrida basada en blockchain con almacenamiento distribuido on-chain/off-chain para gestionar historias clínicas electrónicas (HCE) (21)	Survey of Interoperability in Electronic Health Records Management and Proposed Blockchain Based Framework: MyBlockEHR	Se diseñó una arquitectura basada en blockchain con un enfoque de almacenamiento híbrido: los datos pesados y heterogéneos se enviaron a una base de datos off-chain (MongoDB), mientras que metadatos y hashes verificables se mantuvieron en la cadena de bloques. Esta arquitectura también incorporó un modelo de control de acceso en el cual el paciente es el propietario de su HCE y otorga permisos a los médicos o instituciones mediante contratos inteligentes. Finalmente, el prototipo se evaluó en un entorno controlado utilizando una testnet de Ethereum, donde se probaron casos simples de lectura y escritura para analizar su rendimiento.	Los experimentos realizados en un entorno controlado mostraron que el enfoque híbrido de almacenamiento (combinando datos on-chain y off-chain) ofrece un rendimiento superior en comparación con almacenar toda la información exclusivamente en la cadena de bloques. En particular, la arquitectura híbrida presentó tiempos de respuesta más rápidos.	El almacenamiento híbrido propuesto por la arquitectura incrementa la escalabilidad al evitar que datos heterogéneos y de gran tamaño saturen la cadena de bloques, reduciendo significativamente el tiempo de respuesta, el costo de gas y la carga computacional del sistema.	La arquitectura solo fue probada como un prototipo en un entorno controlado, con cargas reducidas y escenarios simplificados (casos simples de lectura y escritura). No se evaluaron métricas adicionales como escalabilidad, consumo energético y resiliencia bajo carga variable, lo que , lo que restringe su aplicabilidad práctica. Por otro lado, no se hace uso de estándares internacionales como HL7 FHIR para lograr interoperabilidad semántica.

Un sistema de intercambio de información clínica centrada en el paciente mediante la tecnología blockchain (22)	A Patient-Centric Health Information Exchange Framework Using Blockchain Technology	Se utilizó una blockchain privada basada en Ethereum, cuya función principal fue la transmisión de la información necesaria para descifrar los registros clínicos y almacenados externamente por cada hospital. Con este propósito se implementaron dos módulos funcionales: Linkage, encargado de conectar las bases de datos clínicas de los hospitales con la blockchain; y Request, responsable de gestionar los permisos otorgados por el paciente y las solicitudes de acceso realizadas por los médicos o instituciones autorizadas. Posteriormente, se generaron computacionalmente los registros clínicos, las cuentas de pacientes y las cuentas de médicos. También, se automatizaron los procesos que simulaban el intercambio de datos entre hospitales, la autorización de accesos y la verificación de la integridad de la información transmitida. Finalmente, se realizaron pruebas de estabilidad, rendimiento y resiliencia del sistema.	El sistema logró procesar exitosamente todas las transacciones realizadas durante los cuatro meses de simulación. El tiempo promedio requerido para autorizar accesos y transmitir la información necesaria entre hospitales fue de aproximadamente 20 segundos. Además, gracias al uso de valores hash asociados a cada registro, el sistema pudo detectar de manera inmediata cualquier modificación realizada sobre los datos intercambiados. Finalmente, la infraestructura mostró una alta robustez, manteniéndose operativa incluso cuando algunos nodos fueron desactivados deliberadamente durante las pruebas.	Entre las principales ventajas del sistema se encuentra su enfoque completamente centrado en el paciente, lo que otorga mayor flexibilidad y control sobre sus datos clínicos sin depender de un repositorio centralizado. La integridad de la información intercambiada está garantizada mediante el uso de valores hash. Asimismo, el intercambio de datos entre hospitales resulta ser rápido, ya que solo se transmiten los elementos necesarios para descifrar los registros clínicos cifrados, y no los datos completos. Finalmente, el sistema presenta altos niveles de seguridad en el control de acceso, asegurando que solo las entidades autorizadas por el paciente puedan visualizar su información clínica.	Entre las principales limitaciones se encuentra que el sistema aún no ha sido probado en entornos reales. Incluso en este contexto controlado, se observaron colas de transacciones debido a las restricciones de procesamiento propias de la blockchain privada empleada. Asimismo, para una implementación a gran escala sería necesario un consumo significativo de recursos por parte de cada entidad participante. Finalmente, el sistema requiere que todas las entidades de salud adopten un estándar común de interoperabilidad, como HL7 FHIR, para garantizar un intercambio de información consistente y plenamente interoperable.
--	--	--	--	---	--

Plataforma basada en blockchain para el intercambio de datos clínicos basado en los estándares HL7 FHIR (23)	Development of Blockchain-Based Health Information Exchange Platform Using HL7 FHIR Standards: Usability Test	Se desarrolló la plataforma Health Pocket, basada en blockchain, para permitir que el paciente controle el acceso a su información clínica mediante un sistema de consentimiento dinámico registrado en la red Panacea (Tendermint). Cuando el paciente autorizaba compartir datos, la plataforma obtenía la información desde las APIs de los hospitales y la convertía al estándar HL7 FHIR, validando su estructura con la herramienta HAPI-FHIR. Finalmente, se realizó una prueba de usabilidad con 116 participantes para evaluar la facilidad de uso y la satisfacción del servicio mediante encuestas Likert.	El sistema logró intercambiar información clínica entre instituciones de salud utilizando HL7 FHIR de manera exitosa. En las pruebas, los pacientes pudieron otorgar o revocar acceso a sus datos mediante el sistema de consentimiento dinámico. Se reportó además un caso de uso destacado, en el que un paciente fue adecuadamente referido desde atención primaria a un hospital terciario gracias a Health Pocket. Finalmente, la prueba de usabilidad concluyó sin abandonos y obtuvo puntuaciones superiores al promedio en facilidad de uso, utilidad percibida y satisfacción general.	El sistema permite el intercambio de información clínica entre distintas instituciones utilizando el estándar HL7 FHIR, lo que facilita la interoperabilidad con otros sistemas que adopten la misma norma. Asimismo, la plataforma soporta diversos tipos de recursos FHIR (información clínica y generada por wearables) lo que amplía su capacidad de transmisión. El uso de blockchain combinado con el consentimiento dinámico aporta integridad y trazabilidad al acceso de datos. Finalmente, las pruebas de usabilidad mostraron altos niveles de satisfacción entre los usuarios.	El estudio se encuentra aún en fase piloto, pues solo participó un número limitado de instituciones, por lo que su desempeño a gran escala no ha sido evaluado. Además, la plataforma únicamente permite visualizar la información clínica compartida, sin integrarla en el sistema de historia clínica del centro receptor. Asimismo, los perfiles FHIR utilizados requieren una mayor expansión para cubrir un mayor rango de datos. Finalmente, aunque la satisfacción general del usuario fue alta, los indicadores de facilidad de uso y utilidad percibida alcanzaron sólo niveles aceptables.
--	---	---	---	--	--

Un sistema de intercambio de datos clínicos y datos provenientes de wearables, basado en una arquitectura híbrida que combina edge computing y blockchain (24).	EdgeMediChain: A Hybrid Edge Blockchain-Based Framework for Health Data Exchange	Se desarrolló un sistema híbrido que combina edge computing y blockchain para el intercambio de datos de salud. Los nodos en el borde se encargan de procesar, validar y autorizar los datos antes de su transmisión, generando metadatos de la transacción que permiten posteriormente acceder a la información real. Estos metadatos se almacenan en la blockchain Ethereum mediante smart contracts, mientras que los datos completos se resguardan externamente en IPFS. Para validar la propuesta, se implementó un prototipo compuesto por cinco máquinas virtuales, una red Ethereum privada y contratos inteligentes desarrollados en Solidity, a partir de los cuales se realizaron pruebas de rendimiento.	A partir del sistema propuesto, se observó una mejora notable en el tiempo de ejecución de las transacciones en comparación con una red blockchain tradicional. El sistema también demostró ser capaz de manejar mayores volúmenes de datos sin degradar el rendimiento y requirió menos recursos computacionales gracias al uso de la tecnología edge computing. Asimismo, el sistema combinado permitió mantener una latencia estable y un crecimiento lineal del registro de transacciones, lo que hace que la red sea más estable con una gran cantidad de datos.	La arquitectura híbrida propuesta permite mantener un funcionamiento estable incluso ante grandes volúmenes de datos, lo que mejora su escalabilidad. La combinación de edge computing con blockchain ofrece un rendimiento superior al de una red blockchain tradicional, al distribuir el procesamiento y reducir la carga sobre la cadena principal. Además, el almacenamiento de los datos completos fuera de la blockchain facilita respuestas más rápidas y la atención de múltiples solicitudes en paralelo, optimizando la eficiencia general del sistema.	El sistema propuesto fue evaluado únicamente en términos de rendimiento dentro de una red privada de Ethereum, por lo que no se puede garantizar su comportamiento en una red pública y, además, los costos podrían aumentar significativamente en ese entorno. Asimismo, no se empleó ningún estándar de interoperabilidad, como HL7 FHIR, lo que podría dificultar su integración con futuras plataformas de salud. Finalmente, los experimentos se realizaron con datos simulados, por lo que aún no se ha validado el desempeño del sistema con información clínica real.
---	--	--	---	--	---

Modelo de intercambio de historias clínicas electrónicas basadas en blockchain con algoritmo de consenso Checkable-State PBFT (CS-PBFT) (25).	Electronic Health Records Sharing Model Based on Blockchain With Checkable State PBFT Consensus Algorithm	Se desarrolló un modelo de intercambio de historias clínicas electrónicas (EMR) basado en blockchain, en el cual los datos permanecen descentralizados y cada entidad de salud conserva su propio EMR. El intercambio se habilita cuando el mecanismo de consenso Checkable-State PBFT (CS-PBFT) válida una transacción. En caso se alcance la mayoría requerida entre los nodos activos, la transacción se considera válida. Una vez alcanzado el consenso, la blockchain registra los metadatos correspondientes permitiendo que la entidad solicitante acceda al EMR de interés a través del sistema del proveedor del dato. Finalmente, se ejecutaron diversas pruebas de rendimiento para evaluar el correcto funcionamiento del modelo.	Según las pruebas realizadas, el modelo logró un tiempo de consenso menor que el PBFT tradicional y redujo el tráfico de mensajes entre nodos al optimizar la cantidad de comunicaciones necesarias. Esto permitió una mayor velocidad de procesamiento general. Además, el uso de blockchain garantizó la integridad de la información registrada, y se verificó que el sistema mantiene un desempeño adecuado incluso al incrementarse el número de nodos en la red.	El modelo mostró buenos tiempos de respuesta y procesamiento en las pruebas realizadas. El algoritmo CS-PBFT demostró un rendimiento superior al PBFT tradicional, especialmente en eficiencia de consenso y manejo del tráfico entre nodos. La capacidad de identificar y excluir nodos inválidos mejoró la tolerancia a fallos del sistema. Además, el uso de blockchain garantizó la integridad de los metadatos registrados y, al no transferir directamente la información clínica dentro de la cadena, se protegió la seguridad y privacidad de los datos reales.	Aunque el modelo logró establecer un mecanismo de intercambio mediante blockchain, no fue evaluado con datos clínicos reales, sino principalmente con simulaciones matemáticas y de red. Asimismo, no se empleó un estándar internacional de interoperabilidad como HL7 FHIR, lo que podría dificultar su integración con otros sistemas de información en salud. Finalmente, pese a los buenos resultados en tiempos de respuesta, el modelo fue validado únicamente en redes privadas pequeñas, por lo que su desempeño en entornos más amplios o complejos aún es incierto.
---	---	---	--	---	--

Arquitectura de interoperabilidad que integra OpenMRS con un servidor HL7 FHIR mediante una capa intermedia basada en GraphQL (26).	An HL7 FHIR and GraphQL approach for interoperability between heterogeneous Electronic Health Record systems	Se propone una arquitectura que incorpora una capa intermedia basada en GraphQL, la cual actúa como interfaz de acceso flexible a los recursos clínicos gestionados por un servidor HL7 FHIR. En esta arquitectura, los datos clínicos almacenados en OpenMRS pueden ser solicitados a través de servicios intermediarios, donde son transformados al estándar HL7 FHIR. Los recursos resultantes son posteriormente almacenados en la base de datos del servidor FHIR y expuestos a las aplicaciones cliente que los solicitan. La validación de la arquitectura se realiza mediante la ejecución de once operaciones CRUD sobre recursos HL7 FHIR. Asimismo, el estudio complementa la validación técnica con una evaluación no empírica de la calidad del software basada en la norma ISO/IEC 25000.	Los resultados muestran que la arquitectura propuesta resolvió de manera adecuada las once operaciones CRUD evaluadas, permitiendo la correcta lectura, creación, actualización y eliminación de recursos clínicos bajo el estándar HL7 FHIR. El uso de GraphQL como capa de acceso contribuyó a mitigar limitaciones propias de los servicios REST tradicionales, tales como el over-fetching y el under-fetching de datos. Asimismo, se demostró la viabilidad de la arquitectura para soportar escenarios de interoperabilidad progresiva entre historias clínicas electrónicas y plataformas basadas en estándares modernos. En cuanto a la validación cualitativa, se evidenció resultados positivos en métricas relacionadas con interoperabilidad, escalabilidad y reutilización del software.	Entre las principales ventajas de la arquitectura propuesta se encuentra su capacidad para facilitar la interoperabilidad entre sistemas heterogéneos de información en salud. El uso de GraphQL como capa de acceso mejora la eficiencia y precisión de las consultas, reduciendo problemas asociados a los servicios REST tradicionales. Asimismo, la incorporación de mecanismos de autenticación y autorización permite restringir el acceso a los recursos clínicos únicamente a usuarios y aplicaciones debidamente autorizados. Por otro lado, la adopción de HL7 FHIR como estándar favorece la interoperabilidad semántica entre los sistemas comunicados.	Entre las principales limitaciones del estudio se encuentra que la arquitectura propuesta no aborda de manera integral aspectos relacionados con la seguridad y la privacidad de la información clínica, más allá de la implementación de mecanismos de autenticación y autorización. Asimismo, el prototipo fue evaluado únicamente en un entorno controlado utilizando datos clínicos simulados. Finalmente, no se realizaron pruebas de usabilidad con usuarios finales, lo cual podría influir en la adopción de la tecnología en contextos clínicos reales.
--	---	--	--	--	---

Arquitectura de interoperabilidad basada en contenedores y middleware para integrar dispositivos médicos heterogéneos (27).	Towards Medical Data Interoperability Through Collaboration of Healthcare Devices	El estudio desarrolló una arquitectura de interoperabilidad que integra datos médicos procedentes de distintos dispositivos. Para ello, se utilizaron contenedores de software encargados de recibir la información generada por cada dispositivo y transmitirla al middleware central. Este middleware buscaba estandarizar los datos recibidos mediante la semántica propuesta por IEEE 11073 y coordina el flujo proveniente de múltiples contenedores. Una vez procesados, los datos eran enviados a un servidor central encargado de consolidarlos. Por último, el funcionamiento del sistema se evaluó mediante pruebas de estabilidad e interoperabilidad funcional.	El prototipo permitió validar la interoperabilidad funcional del sistema. Se comprobó que la arquitectura podía integrar datos médicos procedentes de dispositivos heterogéneos sin requerir configuraciones manuales, integrándolos correctamente en el servidor central. El middleware gestionó de manera adecuada la llegada simultánea de información desde múltiples contenedores, manteniendo el flujo estable. Finalmente, las pruebas mostraron que los tiempos de respuesta fueron apropiados para las condiciones evaluadas.	El prototipo propuesto constituye una alternativa sólida para integrar datos clínicos provenientes de dispositivos heterogéneos. La incorporación de la semántica definida por el estándar IEEE 11073 permite representar los parámetros fisiológicos con significado clínico consistente. Asimismo, el middleware desarrollado mostró robustez al gestionar múltiples transmisiones simultáneas a través de diferentes medios de comunicación, como Wi-Fi o Bluetooth.	El sistema se encuentra aún en una etapa temprana de desarrollo, pues solo ha sido evaluado en un entorno controlado y no en un contexto clínico real. Además, la necesidad de contar con un contenedor específico para cada tipo de dispositivo puede dificultar su implementación en escenarios con una alta diversidad de dispositivos. Asimismo, el estudio no aborda mecanismos avanzados de seguridad ni privacidad, a pesar de que el sistema transmite datos clínicos sensibles.
---	---	---	--	---	--

Arquitectura de interoperabilidad orientada a la integración de datos fisiológicos generados por dispositivos IoT, utilizando HL7 FHIR como modelo canónico (28).	BloTHR: Electronic Health Record Servicing Scheme in IoT-Blockchain Ecosystem	Se desarrolló una arquitectura de interoperabilidad cuyo objetivo fue emplear el estándar HL7 FHIR como modelo canónico. Para ello, se utilizaron dispositivos IoT encargados de generar datos fisiológicos, los cuales se transmitían a un gateway el cual se encargaba de normalizar los datos recibidos. Posteriormente, los datos normalizados eran enviados a un middleware encargado de transformar los datos a recursos HL7 FHIR. Por último, los recursos FHIR generados fueron almacenados en un servidor FHIR el cual actúa como repositorio interoperable. Para la validación del funcionamiento del flujo diseñado, se implementó un prototipo funcional y se realizaron pruebas de rendimiento.	Los resultados evidenciaron que la arquitectura propuesta funciona de manera adecuada a lo largo de todo el flujo, desde la generación de datos fisiológicos en los dispositivos IoT hasta el almacenamiento de los recursos HL7 FHIR en el servidor FHIR. Asimismo, se demostró que el sistema es capaz de soportar múltiples dispositivos IoT, manteniendo una latencia dentro de rangos aceptables para escenarios de monitoreo no crítico. Finalmente, se validó la viabilidad técnica de integrar HL7 FHIR como modelo canónico dentro de una arquitectura de interoperabilidad orientada a dispositivos IoT.	Entre las principales ventajas de la arquitectura propuesta se encuentra el uso de HL7 FHIR, un estándar ampliamente reconocido para la representación y el intercambio de datos clínicos, lo cual facilita la interoperabilidad con sistemas de información en salud modernos. Asimismo, la arquitectura no depende de características de los dispositivos IoT, lo que le confiere versatilidad para integrar distintos tipos de dispositivos médicos. Finalmente, al tratarse de un diseño modular, cada uno de sus componentes puede escalar de manera independiente, sin afectar el funcionamiento del resto del sistema.	El estudio se basa en la implementación de un prototipo funcional, por lo que las pruebas de rendimiento fueron realizadas en un entorno controlado. En consecuencia, aún son necesarias evaluaciones en escenarios reales. Asimismo, el trabajo no profundiza en aspectos como la seguridad y la privacidad de la información clínica, los cuales resultan fundamentales en sistemas de información en salud. Finalmente, no se desarrollaron estudios que evalúen el impacto de la herramienta en entornos reales de atención médica.
--	--	---	---	--	--

Arquitectura de interoperabilidad técnica para datos de monitoreo médico generados por dispositivos IoT heterogéneos, basada en mecanismos de conversión en capas edge y fog (29).	Medical Data Integration and Interoperability through Remote Monitoring of Healthcare Devices	Se diseñó una arquitectura de interoperabilidad de datos de monitoreo médico generados por dispositivos IoT heterogéneos. Para ello, se propuso un sistema capaz de responder a la solicitud de un dato específico, realizando la conversión dinámica de dicho dato al formato requerido por el dispositivo solicitante, independientemente del formato original. La conversión de los datos se lleva a cabo mediante una capa de colaboración distribuida en el borde (edge/fog), en la cual se seleccionan dispositivos con mayor capacidad computacional para actuar como nodos de conversión. Asimismo, se definieron interfaces para la autenticación y gestión del acceso al sistema. Finalmente, el funcionamiento de la arquitectura fue validado mediante simulaciones Monte Carlo utilizando la herramienta iFogSim.	A partir de las simulaciones realizadas, se comprobó que el sistema reduce la dependencia de infraestructuras en la nube para el procesamiento de datos médicos, al disminuir el tráfico de datos enviado hacia esta. Asimismo, se observaron mejores tiempos de respuesta en escenarios con múltiples dispositivos involucrados. Por otro lado, cuando se disponía de varios nodos de colaboración en las capas edge y fog, se obtuvieron menores latencias y un menor consumo de recursos de red.	El sistema propuesto demostró ser capaz de integrar datos médicos heterogéneos sin requerir que todos los dispositivos utilicen un único estándar de transmisión. Asimismo, la solución presenta un enfoque escalable, ya que permite la incorporación de múltiples dispositivos manteniendo un desempeño adecuado. Por otro lado, al descentralizar los procesos de conversión hacia las capas edge y fog, se reduce el uso de recursos en la nube al evitar la centralización del procesamiento de los datos médicos.	El sistema propuesto se encuentra aún en una fase de desarrollo, ya que su desempeño fue evaluado únicamente mediante simulaciones y no en un entorno clínico real. Asimismo, la interoperabilidad alcanzada se limita al nivel técnico, debido a que no se emplean estándares internacionales como HL7 FHIR. Por otro lado, el enfoque planteado depende de que los dispositivos participantes cuenten con capacidades computacionales adicionales a la generación de datos, lo cual no siempre es viable. Finalmente, en términos de seguridad y privacidad de la información, el estudio no profundiza más allá de mecanismos básicos de autenticación.
--	---	--	---	---	--

Anexo 2: Códigos desarrollados para el sistema integrado.

A. Código desarrollado para el nodo simulador (ESP32)

A.1. Código Principal

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <EEPROM.h>
#include "setting.h"

// Datos a Ingresar
String device; // Nombre del dispositivo que manda el dato fisiológico
String paciente; // Nombre del paciente asociado al dato fisiológico
String id_paciente; // ID del paciente (DNI)
String id_device; // ID del dispositivo (código de 4 letras asociado al dispositivo)
String type; // tipo de dato fisiológico a mandar
String med; // Nombre del médico quien midió el dato fisiológico
String id_med; // ID del médico (DNI)

String mqtt_topic_in= device + "/" + id_paciente + "/" + id_device + "/in"; //
Tópico de MQTT por donde ESP32 manda los datos.
String mqtt_topic_out = device + "/" + id_paciente + "/" + id_device + "/out"; //
Tópico de MQTT por donde se escuchara la respuesta del sistema
String mqtt_topic_setting = "sunnyhealth/setting";
String mqtt_topic_command = "sunnyhealth/"+device+"/"+id_device+"/command";
const int mqtt_port = 1883; // Puerto predeterminado de
EMQX
bool configurado = false; // Bandera que indica si el dispositivo ya fue
configurado en setting

// Rango de valores para cada categoría
const int T_min = 36; // Temperatura corporal mínima (C°)
const int T_max = 38; // Temperatura corporal máxima (C°)

const int Ps_min = 90; // Presión arterial sistólica mínima (mmHg)
const int Ps_max = 120; // Presión arterial sistólica máxima (mmHg)

const int Pd_min = 60; // Presión arterial diastólica mínima (mmHg)
const int Pd_max = 80; // Presión arterial diastólica máxima (mmHg)

const int F_min = 50; // Frecuencia cardiaca mínima (bea/min)
const int F_max = 120; // Frecuencia cardiaca máxima (beat/min)

const int R_min = 12; // Frecuencia respiratoria mínima (resp/min)
const int R_max = 20; // Frecuencia respiratoria máxima (resp/min)

const int O_min = 95; // Saturación de oxígeno mínima (resp/min)
const int O_max = 100; // Saturación de oxígeno máxima (resp/min)

const int G_min = 70; // Glucosa en sangre mínima (mg/dL)
const int G_max = 125; // Glucosa en sangre máxima (mg/dL)

// Tipo de dato fisiológico configurado (guarda el ultimo tipo de dato que se
envió desde StreamLit)
String tipoConfigurado = ""; // guarda el tipo elegido: "T", "F", "P", etc

// Se guardan variables para el borrado del dato por defecto de la memoria FLASH
unsigned long tiempoPresionado = 0;
```

```

bool botonEnReset = false;

// Se define el pin del LED
#define LED_PIN 2

WiFiClient espClient;
PubSubClient mqtt_client(espClient);

// NTP Settings
WiFiUDP udp;
NTPClient timeClient(udp, "pool.ntp.org", -18000, 60000); // UTC-5 (Perú)

// Function Declarations
void connectToWiFi();
void connectToMQTT();
void mqttCallback(char *topic, byte *payload, unsigned int length);
String generarJsonObservacion(String tipo, float valor1, float valor2, String
paciente, String device, String med, String id_paciente, String id_device,
String id_med);
String obtenerHoraActualISO();

void setup() {
    Serial.begin(115200);
    connectToWiFi();

    // Configuración de la conexión al broker
    mqtt_client.setServer(mqtt_broker, mqtt_port);
    mqtt_client.setCallback(mqttCallback);
    connectToMQTT();
    timeClient.begin(); // Inicia la sincronización con el servidor NTP

    // Configuración del botón del microcontrolador
    pinMode(0, INPUT_PULLUP);

    // Configuración del led del microcontrolador
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, LOW);

    // Inicializar EEPROM memoria flash
    EEPROM.begin(10); // 10 bytes son suficientes

    // Leer el valor guardado del tipo de dato fisiológico por defecto al
    encender el ESP32

    tipoConfigurado = EEPROM.readString(0);

    if (tipoConfigurado != "") {
        Serial.println("Tipo de dato fisiológico por defecto guardado de memoria
FLASH: " + tipoConfigurado);
    } else {
        Serial.println("No hay tipo de dato fisiológico por defecto en memoria
FLASH.");
    }
}

void connectToWiFi() {
    WiFi.begin(ssid, password);
    Serial.print("Conectando a la red Wifi...");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConectado a la red Wifi");
}

```

```

}

void connectToMQTT() {
    while (!mqtt_client.connected()) {
        String client_id = "esp32-client-" + String(WiFi.macAddress());
        Serial.printf("Conectando al broker MQTT como: %s.....\n",
client_id.c_str());
        if (mqtt_client.connect(client_id.c_str())) { // Si no usas
usuario/contraseña
            Serial.println("Conectado al broker MQTT");
            // Nos suscribimos al t pico
            if (mqtt_client.subscribe(mqtt_topic_setting.c_str())) {
                Serial.println("Suscripci n exitosa al topic: ");
                Serial.println(mqtt_topic_setting);
            } else {
                Serial.println("Error al suscribirse al topic");
                Serial.println(mqtt_topic_setting);
            }
        } else {
            Serial.print("Fall  la conexi n. Intentando de nuevo en 5
segundos...");
            delay(5000);
        }
    }
}

void mqttCallback(char *topic, byte *payload, unsigned int length) {
    Serial.print("Mensaje recibido en el topic: ");
    Serial.println(topic);

    String message = "";
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }

    Serial.println("Contenido del mensaje:");
    Serial.println(message);
    Serial.println("-----");

    // Configuraci n inicial en caso no se haya configurado

    if (String(topic) == mqtt_topic_setting && !configurado) {
        StaticJsonDocument<256> doc;
        DeserializationError error = deserializeJson(doc, message);
        if (error) {
            Serial.print("Error al procesar el JSON: ");
            Serial.println(error.c_str());
            return;
        }

        paciente = doc["paciente"].as<String>();
        device = doc["device"].as<String>();
        med = doc["med"].as<String>();
        id_paciente = doc["id_paciente"].as<String>();
        id_device = doc["id_device"].as<String>();
        id_med = doc["id_med"].as<String>();

        // Construcci n de topico dinamico de salida de ESP32
        mqtt_topic_out = device + "/" + id_paciente + "/" + id_device + "/out";
        if (mqtt_client.subscribe(mqtt_topic_out.c_str())) {
            Serial.println("Suscripci n exitosa al topic: ");
            Serial.println(mqtt_topic_out);
        } else {
            Serial.println("Error al suscribirse al topic");
            Serial.println(mqtt_topic_out);
        }
    }
}

```

```

// Construcción de topico dinamico de comandos al ESP32
mqtt_topic_command = "sunnyhealth/" + device + "/" + id_device + "/command";
if (mqtt_client.subscribe(mqtt_topic_command.c_str())) {
    Serial.println("Suscripción exitosa al topic: ");
    Serial.println(mqtt_topic_command);
} else {
    Serial.println("Error al suscribirse al topic");
    Serial.println(mqtt_topic_command);
}

// Desuscripción de setting una vez configurado
mqtt_client.unsubscribe(mqtt_topic_setting.c_str());
Serial.println("Desuscrito de setting");

configurado=true;
}

// Comando (dato fisiológico)
if (String(topic) == mqtt_topic_command) {
    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, message);
    if (error) {
        Serial.print("Error al procesar el JSON: ");
        Serial.println(error.c_str());
        return;
    }

    String tipoStr = doc["tipo_dato_fisiologico"];
    String valor = doc["valor"] | ""; // Por si acaso no viene

    tipoStr.trim();
    valor.trim();

    // Combinar ambos si hay valor
    String comandoFinal = tipoStr;
    if (valor.length() > 0) {
        comandoFinal += " " + valor;
    }

    Serial.println("Comando procesado: " + comandoFinal);
    processInput(comandoFinal);
}
}

void processInput(String comando) {
    int valor1 = 0;
    int valor2 = 0;
    bool tieneValores = false;

    // Separar comando por espacios
    comando.trim();
    int espaciol = comando.indexOf(' ');
    char tipo = toupper(comando.charAt(0));

    // Se guarda el ultimo tipo de dato enviado desde StreamLit
    tipoConfigurado = String(tipo);
    Serial.println("Tipo de dato fisiologico por defecto: " + tipoConfigurado);

    // Se guarda el tipo de dato por defecto en memoria flash
    EEPROM.writeString(0, tipoConfigurado);
    EEPROM.commit();

    // En caso se haya enviado un valor personalizado entra al condicional
    if (espaciol > 0) {

```

```

String resto = comando.substring(espacio1 + 1);
resto.trim();

int espacio2 = resto.indexOf(' ');
// En caso se haya enviado dos valores personalizados entra al
condicional(ej. presión)
if (espacio2 > 0) {
    valor1 = resto.substring(0, espacio2).toInt();
    valor2 = resto.substring(espacio2 + 1).toInt();
} else {
    valor1 = resto.toInt();
    valor2 = 0; // se asigna 0 a la variable "valor2" en caso no se trate de
la presion
}

tieneValores = true;
}

// Validar tipo
switch (tipo) { // En cada caso se haya determinado que se tiene valores
personalizados se guardan estos (en caso no es uno random dentro de los limites)
    case 'T':
        type = "temperature";
        valor1 = tieneValores ? valor1 : random(T_min, T_max + 1);
        break;
    case 'F':
        type = "heart_rate";
        valor1 = tieneValores ? valor1 : random(F_min, F_max + 1);
        break;
    case 'R':
        type = "respiratory_rate";
        valor1 = tieneValores ? valor1 : random(R_min, R_max + 1);
        break;
    case 'O':
        type = "oxygen_saturation";
        valor1 = tieneValores ? valor1 : random(O_min, O_max + 1);
        break;
    case 'G':
        type = "glycemia";
        valor1 = tieneValores ? valor1 : random(G_min, G_max + 1);
        break;
    case 'P':
        type = "blood_pressure";
        if (!tieneValores) {
            valor1 = random(Ps_min, Ps_max + 1);
            valor2 = random(Pd_min, Pd_max + 1);
        }
        break;
    default:
        Serial.println("Error: Letra no reconocida.");
        return;
}

// Construir y publicar JSON
mqtt_topic_in = device + "/" + id_paciente + "/" + type + "/" + id_device +
"/in";
Serial.println("Publicando en el topic:");
Serial.println(mqtt_topic_in);

String jsonOutput = generarJsonObservacion(type, valor1, valor2, paciente,
device, med, id_paciente, id_device, id_med); // se construye en JSON

// Parpadeo del LED al enviar un dato
digitalWrite(LED_PIN, HIGH);
delay(300);
digitalWrite(LED_PIN, LOW);

```

```

    mqtt_client.publish(mqtt_topic_in.c_str(), jsonOutput.c_str()); // se publica
    el Json en el topico construido
    Serial.println("Mensaje enviado (JSON):");
    Serial.println(jsonOutput);
}

void enviarDatoAutomatico() {
    if (tipoConfigurado == "") {
        Serial.println("Aún no se ha configurado un tipo de dato.");
        return;
    }

    int v1, v2;

    if (tipoConfigurado == "T") {
        type = "temperature";
        v1 = random(T_min, T_max + 1);
        v2 = 0;
    }
    else if (tipoConfigurado == "F") {
        type = "heart_rate";
        v1 = random(F_min, F_max + 1);
        v2 = 0;
    }
    else if (tipoConfigurado == "R") {
        type = "respiratory_rate";
        v1 = random(R_min, R_max + 1);
        v2 = 0;
    }
    else if (tipoConfigurado == "O") {
        type = "oxygen_saturation";
        v1 = random(O_min, O_max + 1);
        v2 = 0;
    }
    else if (tipoConfigurado == "G") {
        type = "glycemia";
        v1 = random(G_min, G_max + 1);
        v2 = 0;
    }
    else if (tipoConfigurado == "P") {
        type = "blood_pressure";
        v1 = random(Ps_min, Ps_max + 1);
        v2 = random(Pd_min, Pd_max + 1);
    }

    // Parpadeo del LED al enviar un dato
    digitalWrite(LED_PIN, HIGH);
    delay(300);
    digitalWrite(LED_PIN, LOW);

    // Construir topic y enviar
    mqtt_topic_in = device + "/" + id_paciente + "/" + type + "/" + id_device +
    "/in";

    String jsonOutput = generarJsonObservacion(
        type, v1, v2,
        paciente, device, med,
        id_paciente, id_device, id_med
    );

    mqtt_client.publish(mqtt_topic_in.c_str(), jsonOutput.c_str());
    Serial.println("Dato automático enviado:");
    Serial.println(jsonOutput);
}

```

```

void loop() {
    if (!mqtt_client.connected()) {
        connectToMQTT();
    }
    mqtt_client.loop();

    // Actualizar la hora del NTPClient
    timeClient.update();

    // En caso se presione el boton designado se envia un dato random en el tipo
    de dato fisiologico por defecto

    // --- DETECCIÓN DE RESET DE CONFIGURACIÓN (mantener 3 segundos) ---
    if (digitalRead(0) == LOW) {
        if (!botonEnReset) {
            botonEnReset = true;
            tiempoPresionado = millis();
        } else {
            // Si pasaron 3 segundos presionado, se hace RESET
            if (millis() - tiempoPresionado > 3000) {
                EEPROM.writeString(0, "");
                EEPROM.commit();
                tipoConfigurado = "";
                Serial.println("***** CONFIGURACIÓN RESETEADA *****");
                // Parpadeo rápido del LED para indicar reset
                for (int i = 0; i < 5; i++) {
                    digitalWrite(LED_PIN, HIGH);
                    delay(100);
                    digitalWrite(LED_PIN, LOW);
                    delay(100);
                }
                // Esperar a que suelte el botón para evitar reenviar datos
                while (digitalRead(0) == LOW) {}
                botonEnReset = false;
                return; // SALIR DEL LOOP ESTA VEZ
            }
        }
    }

    // Si el botón se suelta: Accion normal
    else {
        if (botonEnReset && millis() - tiempoPresionado < 3000) {
            enviarDatoAutomatico();
        }
        botonEnReset = false;
    }
}

String generarJsonObservacion(String tipo, float valor1, float valor2,
                               String paciente, String device, String med,
                               String id_paciente, String id_device, String
id_med) {
    String json = "{\n";
    json += "  \"resourceType\": \"Observation\",\n";
    json += "  \"id\": \"" + tipo + "\",\n";

    // Definir el perfil FHIR según el tipo de observación
    if (tipo == "temperature") {
        json += "    \"meta\":{\n";
        json += "      \"profile\":\n";
        json += "    },\n";
        json += "  },\n";
    } else if (tipo == "blood_pressure") {
        json += "    \"meta\":{\n";

```

```

        json += "                                \"profile\":\n";
["http://hl7.org/fhir/StructureDefinition/bp"]\n";
        json += "    },\n";
        } else if (tipo == "heart_rate") {
            json += "        \"meta\":{\n";
            json += "                                \"profile\":\n";
["http://hl7.org/fhir/StructureDefinition/vitalsigns"]\n";
            json += "    },\n";
            } else if (tipo == "respiratory_rate") {
                json += "        \"meta\":{\n";
                json += "                                \"profile\":\n";
["http://hl7.org/fhir/StructureDefinition/resprate"]\n";
                json += "    },\n";
                } else if (tipo == "oxygen_saturation") {
                    json += "        \"meta\":{\n";
                    json += "                                \"profile\":\n";
["http://hl7.org/fhir/StructureDefinition/oxygensat"]\n";
                    json += "    },\n";
                    } else if (tipo == "glycemia") {
                        json += "        \"meta\":{\n";
                        json += "                                \"profile\":\n";
["http://hl7.org/fhir/StructureDefinition/Observation"]\n";
                        json += "    },\n";
                    }

        json += "    \"text\": {\n";
        json += "        \"status\": \"generated\", \n";
        json += "                                \"div\": \"<div
xmlns=\\\"http://www.w3.org/1999/xhtml\\\">Observation for " + paciente +
"</div>\n";
        json += "    },\n";
        json += "        \"status\": \"final\", \n";
        json += "        \"category\": [\n";
        json += "            {\n";
        json += "                \"coding\": [\n";
        json += "                    {\n";
        json += "                                \"system\":\n";
["http://terminology.hl7.org/CodeSystem/observation-category", \n";
        json += "                \"code\": \"vital-signs\", \n";
        json += "                \"display\": \"Vital Signs\" \n";
        json += "            } \n";
        json += "        ], \n";
        json += "        \"text\": \"Vital Signs\" \n";
        json += "    } \n";
        json += "    ], \n";

// Código LOINC específico
if (tipo == "temperature") {
    json += "        \"code\": {\n";
    json += "            \"coding\": [\n";
    json += "                {\n";
    json += "                    \"system\": \"http://loinc.org\", \n";
    json += "                    \"code\": \"8310-5\", \n";
    json += "                    \"display\": \"Body temperature\" \n";
    json += "                } \n";
    json += "            ] \n";
    json += "        }, \n";
} else if (tipo == "blood_pressure") {
    json += "        \"code\": {\n";
    json += "            \"coding\": [\n";
    json += "                {\n";
    json += "                    \"system\": \"http://loinc.org\", \n";
    json += "                    \"code\": \"85354-9\", \n";
    json += "                    \"display\": \"Blood pressure panel with all children
optional\" \n";
    json += "                } \n";

```

```

        json += "    ]\n";
        json += "  },\n";
    } else if (tipo == "heart_rate") {
        json += "    \"code\": {\n";
        json += "      \"coding\": {\n";
        json += "        {\n";
        json += "          \"system\": \"http://loinc.org\", \n";
        json += "          \"code\": \"8867-4\", \n";
        json += "          \"display\": \"Heart rate\" \n";
        json += "        }\n";
        json += "      }\n";
        json += "    },\n";
    } else if (tipo == "respiratory_rate") {
        json += "    \"code\": {\n";
        json += "      \"coding\": {\n";
        json += "        {\n";
        json += "          \"system\": \"http://loinc.org\", \n";
        json += "          \"code\": \"9279-1\", \n";
        json += "          \"display\": \"Respiratory rate\" \n";
        json += "        }\n";
        json += "      }\n";
        json += "    },\n";
    } else if (tipo == "oxygen_saturation") {
        json += "    \"code\": {\n";
        json += "      \"coding\": {\n";
        json += "        {\n";
        json += "          \"system\": \"http://loinc.org\", \n";
        json += "          \"code\": \"2708-6\", \n";
        json += "          \"display\": \"Oxygen saturation in Arterial blood\" \n";
        json += "        }\n";
        json += "      }\n";
        json += "    },\n";
    } else if (tipo == "glycemia") {
        json += "    \"code\": {\n";
        json += "      \"coding\": {\n";
        json += "        {\n";
        json += "          \"system\": \"http://loinc.org\", \n";
        json += "          \"code\": \"2339-0\", \n";
        json += "          \"display\": \"Glucose [Mass/volume] in Blood\" \n";
        json += "        }\n";
        json += "      }\n";
        json += "    },\n";
    }
}

json += "  \"device\": {\n";
json += "    \"reference\": \"Device/\"+ id_device+ "\", \n";
json += "    \"display\": \"\" + device + "\" \n";
json += "  },\n";
json += "  \"subject\": {\n";
json += "    \"reference\": \"Patient/\"+ id_paciente+ "\", \n";
json += "    \"display\": \"\" + paciente + "\" \n";
json += "  },\n";
json += "  \"performer\": {\n";
json += "    {\n";
json += "      \"reference\": \"Performer/\"+ id_med+ "\", \n";
json += "      \"display\": \"\" + med + "\" \n";
json += "    }\n";
json += "  },\n";
json += "  \"effectiveDateTime\": \"\" + obtenerHoraActualISO() + "\" \n";

// Sección de valores específicos para cada tipo de observación
if (tipo == "temperature") {
    json += "    \"valueQuantity\": {\n";
    json += "      \"value\": \" + String(valor1) + ", \n";
    json += "      \"unit\": \"°C\", \n";
    json += "      \"system\": \"http://unitsofmeasure.org\", \n";

```

```

json += "    \"code\": \"Cel\"\n";
json += "  }\n";
} else if (tipo == "blood_pressure") {
    json += "    \"component\": [\n";
    json += "      {\n";
    json += "        \"code\": {\n";
    json += "          \"coding\": [\n";
    json += "            {\n";
    json += "              \"system\": \"http://loinc.org\", \n";
    json += "              \"code\": \"8480-6\", \n";
    json += "              \"display\": \"Systolic blood pressure\"\n";
    json += "            }\n";
    json += "          ]\n";
    json += "        }, \n";
    json += "        \"valueQuantity\": {\n";
    json += "          \"value\": \" + String(valor1) + \"\", \n";
    json += "          \"unit\": \"mmHg\", \n";
    json += "          \"system\": \"http://unitsofmeasure.org\", \n";
    json += "          \"code\": \"mm[Hg]\"\n";
    json += "        }\n";
    json += "      }, \n";
    json += "      {\n";
    json += "        \"code\": {\n";
    json += "          \"coding\": [\n";
    json += "            {\n";
    json += "              \"system\": \"http://loinc.org\", \n";
    json += "              \"code\": \"8462-4\", \n";
    json += "              \"display\": \"Diastolic blood pressure\"\n";
    json += "            }\n";
    json += "          ]\n";
    json += "        }, \n";
    json += "        \"valueQuantity\": {\n";
    json += "          \"value\": \" + String(valor2) + \"\", \n";
    json += "          \"unit\": \"mmHg\", \n";
    json += "          \"system\": \"http://unitsofmeasure.org\", \n";
    json += "          \"code\": \"mm[Hg]\"\n";
    json += "        }\n";
    json += "      }\n";
    json += "    ]\n";
} else if (tipo == "heart_rate") {
    json += "    \"valueQuantity\": {\n";
    json += "      \"value\": \" + String(valor1) + \"\", \n";
    json += "      \"unit\": \"beats/min\", \n";
    json += "      \"system\": \"http://unitsofmeasure.org\", \n";
    json += "      \"code\": \"1/min\"\n";
    json += "    }\n";
} else if (tipo == "respiratory_rate") {
    json += "    \"valueQuantity\": {\n";
    json += "      \"value\": \" + String(valor1) + \"\", \n";
    json += "      \"unit\": \"breaths per minute\", \n";
    json += "      \"system\": \"http://unitsofmeasure.org\", \n";
    json += "      \"code\": \"1/min\"\n";
    json += "    }\n";
} else if (tipo == "oxygen_saturation") {
    json += "    \"valueQuantity\": {\n";
    json += "      \"value\": \" + String(valor1) + \"\", \n";
    json += "      \"unit\": \"%O2\", \n";
    json += "      \"system\": \"http://unitsofmeasure.org\", \n";
    json += "      \"code\": \"%\"\n";
    json += "    }\n";
} else if (tipo == "glycemia") {
    json += "    \"valueQuantity\": {\n";
    json += "      \"value\": \" + String(valor1) + \"\", \n";
    json += "      \"unit\": \"mg/dL\", \n";
    json += "      \"system\": \"http://unitsofmeasure.org\", \n";
    json += "      \"code\": \"mg/dL\"\n";

```

```

        json += "  }\n";
    }

    json += "}\n";
    return json;
}

// Función para obtener la hora actual en formato ISO 8601
String obtenerHoraActualISO() {
    unsigned long epochTime = timeClient.getEpochTime();
    int year = 1970;
    int month, day;

    // Calcular año
    while (epochTime >= 31556926) {
        if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
            if (epochTime >= 31622400)
                epochTime -= 31622400;
            else
                break;
        } else {
            epochTime -= 31536000;
        }
        year++;
    }

    // Calcular mes y día
    int daysInMonth[] = {31, (year % 4 == 0 && year % 100 != 0) || year % 400
    == 0 ? 29 : 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    for (month = 0; month < 12; month++) {
        if (epochTime < daysInMonth[month] * 86400)
            break;
        epochTime -= daysInMonth[month] * 86400;
    }
    month++;
    day = epochTime / 86400 + 1;
    epochTime %= 86400;

    // Calcular hora, minuto y segundo
    int hour = epochTime / 3600;
    int minute = (epochTime % 3600) / 60;
    int second = epochTime % 60;

    // Construir el formato ISO 8601
    char isoString[30];
    sprintf(isoString, "%04d-%02d-%02dT%02d:%02d:%02d-05:00", year, month, day,
    hour, minute, second);
    return String(isoString);
}

```

A.2. Archivo de configuración de red

```

const char *ssid = "XXXX";           // Reemplaza con tu red WiFi
const char *password = "XXXXX";      // Reemplaza con tu contraseña WiFi
("XXXX")
const char *mqtt_broker = "XXX.XXX.XX.XX"; // IP de broker de MQTT

```

B. Códigos desarrollados para la API

B.1. Código del servicio Sunny Health API

```
import json
import requests
from werkzeug.security import generate_password_hash
from collections import OrderedDict
from flask import Flask, request, jsonify, Response
import psycpg2
from datetime import datetime
import pytz
import paho.mqtt.publish as publish
import mysql.connector

app = Flask(__name__)

# Broker de comunicacion al Arduino MQTT
MQTT_BROKER = "xxx.xxx.xx.xx"
MQTT_PORT = 1883

# URL de Merry (Oyente de Arduino)
MERRY_URL = "http://localhost:5001/suscribir" # Cambia el puerto si merry corre
en otro

# ----- Configuracion de acceso a base de datos -----
DB_CONFIG_GNU = {
    "dbname": "health",
    "user": "gnuhealth",
    "host": "localhost",
    "port": "5432"
}

DB_CONFIG_MARIA = {
    "host": "localhost",
    "user": "xxxx",
    "password": "xxxx",
    "database": "health_monitoring"
}

# ----- Funciones de ayuda en almacenamiento de datos en DB -----
def get_id_from_dni(cur, dni, table_name):
    try:
```

```

        cur.execute("SELECT id FROM party_party WHERE ref = %s", (dni,))
        party_result = cur.fetchone()
        if not party_result:
            return None
        party_id = party_result[0]
        cur.execute(f"SELECT id FROM {table_name} WHERE name = %s", (party_id,))
        table_result = cur.fetchone()
        return table_result[0] if table_result else None
    except Exception as e:
        print(f"Error al obtener ID desde DNI: {e}")
        return None

def save_to_db_GNU(data):
    conn1 = None
    try:
        conn1 = psycopg2.connect(**DB_CONFIG_GNU)
        cur1 = conn1.cursor()

        discharge_reason = "home" # fijo para demo

        # IDs de paciente y médico
        dni_patient = data.get("subject", {}).get("reference",
        "").replace("Patient/", "")
        dni_health_prof = data.get("performer", [{}])[0].get("reference",
        "").replace("Performer/", "")

        patient_id = get_id_from_dni(cur1, dni_patient, "gnuhealth_patient")
        health_prof = get_id_from_dni(cur1, dni_health_prof,
        "gnuhealth_healthprofessional")

        if not patient_id or not health_prof:
            print("Error: paciente o profesional no encontrados en GNU Health.")
            return False

        timestamp_iso = data.get("effectiveDateTime")

        if timestamp_iso:
            try:
                dt = datetime.fromisoformat(timestamp_iso)

                if dt.tzinfo is not None:
                    dt = dt.astimezone(pytz.utc)
                else:
                    dt = dt.replace(tzinfo=pytz.utc)

```

```

        evaluation_start = dt.strftime('%Y-%m-%d %H:%M:%S')

    except Exception:
        # Si viene mal formateado, se usa la hora local de registro del
dato en el sistema
        lima_tz = pytz.timezone("America/Lima")
        now_lima = datetime.now(lima_tz)
        evaluation_start = now_lima.astimezone(pytz.utc).strftime('%Y-
%m-%d %H:%M:%S')
    else:
        # Si no viene effectiveDateTime, se la hora local de registro del
dato en el sistema
        lima_tz = pytz.timezone("America/Lima")
        now_lima = datetime.now(lima_tz)
        evaluation_start = now_lima.astimezone(pytz.utc).strftime('%Y-%m-%d
%H:%M:%S')

# Variables fisiológicas
tipo = data.get("resourceType", "Observation")
temperature = bpm = systolic = diastolic = respiratory_rate = osat =
glycemia = None

if tipo == "Observation" and data["id"] == "temperature":
    temperature = data["valueQuantity"].get("value")
if tipo == "Observation" and data["id"] == "blood_pressure":
    systolic = data["component"][0]["valueQuantity"]["value"]
    diastolic = data["component"][1]["valueQuantity"]["value"]
if tipo == "Observation" and data["id"] == "heart_rate":
    bpm = data["valueQuantity"].get("value")
if tipo == "Observation" and data["id"] == "respiratory_rate":
    respiratory_rate = data["valueQuantity"].get("value")
if tipo == "Observation" and data["id"] == "oxygen_saturation":
    osat = data["valueQuantity"].get("value")
if tipo == "Observation" and data["id"] == "glycemia":
    glycemia = data["valueQuantity"].get("value")

# Insert
query = """
INSERT INTO gnuhealth_patient_evaluation
(patient, temperature, bpm, systolic, diastolic, discharge_reason,
evaluation_start, healthprof, respiratory_rate, osat, glycemia)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

```

```

        """
        values = (patient_id, temperature, bpm, systolic, diastolic,
discharge_reason, evaluation_start, health_prof, respiratory_rate, osat,
glycemia)

        curl.execute(query, values)
        conn1.commit()
        print("[Sunny] Datos insertados en GNU Health correctamente.")

        return True
    except Exception as e:
        if conn1:
            conn1.rollback()
            print("[Sunny] Error al insertar en GNU Health:", e)
            return False
    finally:
        if conn1:
            curl.close()
            conn1.close()

def save_to_db_Maria(data):
    conn = None
    try:
        conn = mysql.connector.connect(**DB_CONFIG_MARIA)
        cursor = conn.cursor()

        physiological_type = data.get("id", "Desconocido")
        patient_name = data["subject"].get("display", "Desconocido")
        patient_id = data["subject"].get("reference", "").split("/")[-1]
        doctor_name = data["performer"][0].get("display", "Desconocido") if
"performer" in data else "Desconocido"
        doctor_id = data["performer"][0].get("reference", "").split("/")[-1] if
"performer" in data else "N/A"
        device_name = data["device"].get("display", "Desconocido")
        device_id = data["device"].get("reference", "").split("/")[-1]
        full_message = json.dumps(data)

        # timestamp
        timestamp_iso = data.get("effectiveDateTime", None)
        if timestamp_iso:
            try:
                dt = datetime.fromisoformat(timestamp_iso)
                timestamp = dt.strftime('%Y-%m-%d %H:%M:%S')
            except ValueError:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

```

```

else:
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    value = None
    value2 = None
    unit = ""

    if "valueQuantity" in data:
        value = data["valueQuantity"].get("value", None)
        unit = data["valueQuantity"].get("unit", "Desconocido")

    if data["id"] == "blood_pressure":
        value = data["component"][0]["valueQuantity"]["value"]
        value2 = data["component"][1]["valueQuantity"]["value"]
        unit = data["component"][0]["valueQuantity"]["unit"]

    sql = """
        INSERT INTO observations (physiological_type, patient_name, patient_id,
        doctor_name, doctor_id,
                                device_name, device_id, unit, value, value2,
        full_message)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """
    values = (physiological_type, patient_name, patient_id, doctor_name,
    doctor_id, device_name, device_id, unit, value, value2, full_message)
    cursor.execute(sql, values)
    conn.commit()
    print("[Sunny] Datos insertados en MariaDB correctamente.")

    return True

except Exception as e:
    print(f"[Sunny] Error al insertar en MariaDB: {e}")
    return False

finally:
    if conn:
        cursor.close()
        conn.close()

## Funcion de conexion la base de datos en MariaDB
def get_db_connection():
    return mysql.connector.connect(**DB_CONFIG_MARIA)

```

```

@app.route('/')
def home():
    routes = {
        "/": "Muestra este mensaje con las rutas disponibles.",

        # Observaciones
        "/data [GET]": "Devuelve todos los datos almacenados en la base de
datos.",
        "/data/<id> [GET]": "Devuelve los detalles de un registro específico por
su ID.",

        # Pacientes
        "/patients [GET]": "Lista todos los pacientes únicos en la base de
datos.",
        "/patients/<patient_id> [GET]": "Lista todas las mediciones de un
paciente en específico.",
        "/patients/<patient_id>/latest [GET]": "Obtiene la última medición de
un paciente específico.",
        "/patients/<patient_id>/<physiological_type> [GET]": "Lista todas las
mediciones de un paciente según el tipo fisiológico.",
        "/patients/<patient_id>/<physiological_type>/latest [GET]": "Obtiene la
última medición de un tipo fisiológico de un paciente.",

        # Dispositivos
        "/devices [GET]": "Lista todos los dispositivos únicos en la base de
datos.",
        "/devices/<device_id> [GET]": "Lista todas las mediciones que ha
realizado un dispositivo en específico.",

        # Médicos
        "/doctors [GET]": "Lista todos los médicos únicos en la base de datos.",
        "/doctors/<doctor_id> [GET]": "Lista todas las mediciones que ha
realizado un médico en específico.",

        # Usuarios (Autenticación)
        "/users/<username> [GET]": "Obtiene el hash de contraseña de un usuario
(login).",
        "/users/<username> [PATCH]": "Actualiza la contraseña de un usuario
existente.",
        "/users [POST]": "Registra un nuevo usuario con nombre de usuario y
contraseña."
    }

    response_data = {

```

```

        "message": "Bienvenido a Sunny Health API, la plataforma para el
monitoreo de datos médicos.",
        "rutas": routes
    }

    response_json = json.dumps(response_data, ensure_ascii=False, indent=2)
    return Response(response_json, content_type="application/json; charset=utf-
8")

@app.route('/data', methods=['GET'])
def get_data():
    """Obtiene todos los registros en el orden personalizado, con JSON
formateado."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations
        """)
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results
        ]

        cursor.close()
        conn.close()

```

```

        # Devolver JSON con formato legible (pretty print)
        return app.response_class(
            response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
# Agregar indentación
            status=200,
            mimetype="application/json"
        )
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Obtener un registro por ID
@app.route('/data/<int:id>', methods=['GET'])

def get_data_by_id(id):
    """Devuelve un registro específico según su ID."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT    id,      physiological_type,    patient_name,    patient_id,
doctor_name, doctor_id,
                    device_name, device_id, value, value2, unit, timestamp,
full_message
            FROM observations WHERE id = %s
        """, (id,))
        row = cursor.fetchone() # Solo un registro

        cursor.close()
        conn.close()

        if not row:
            return jsonify({"Error": "El ID del registro ingresado en la ruta
no existe"}), 404

        # Convertir full_message de string a JSON
        try:
            full_message_json = json.loads(row["full_message"]) if
row["full_message"] else None
        except json.JSONDecodeError:
            full_message_json = row["full_message"] # Si hay error, devolver
como string

        # Convertir fila a OrderedDict y formatear timestamp

```

```

ordered_result = OrderedDict([
    ("id", row["id"]),
    ("physiological_type", row["physiological_type"]),
    ("patient_name", row["patient_name"]),
    ("patient_id", row["patient_id"]),
    ("doctor_name", row["doctor_name"]),
    ("doctor_id", row["doctor_id"]),
    ("device_name", row["device_name"]),
    ("device_id", row["device_id"]),
    ("value", row["value"]),
    ("value2", row["value2"]),
    ("unit", row["unit"]),
    ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None),
    ("full_message", full_message_json) # JSON indentado
])

# Devolver JSON con indentación
return app.response_class(
    response=json.dumps(ordered_result, ensure_ascii=False, indent=4),
    status=200,
    mimetype="application/json"
)

except Exception as e:
    return jsonify({"error": str(e)}), 500

# Listar los pacientes

@app.route('/patients', methods=['GET'])
def get_patients():
    """Lista todos los pacientes únicos registrados con su ID y nombre."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        # Obtener pacientes únicos (ID + Nombre)
        cursor.execute("""
            SELECT DISTINCT patient_id, patient_name
            FROM observations
            WHERE patient_id IS NOT NULL AND patient_name IS NOT NULL
            ORDER BY patient_name ASC
        """)
        results = cursor.fetchall()

```

```

        cursor.close()
        conn.close()

    # Formato de respuesta con mensaje adicional
    response = {
        "message": "Lista de pacientes que tienen un registro",
        "results": results
    }

    return jsonify(response)
except Exception as e:
    return jsonify({"error": str(e)}), 500

# Obtener registros de un paciente con el ID

@app.route('/patients/<int:patient_id>', methods=['GET'])
def get_patient_data(patient_id):
    """Devuelve todos los datos de un paciente específico."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE patient_id = %s
            """, (patient_id,))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results

```

```

    ]
    cursor.close()
    conn.close()

    # Devolver JSON con formato legible (pretty print)
    return app.response_class(
        response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
# Agregar indentación
        status=200,
        mimetype="application/json"
    )
except Exception as e:
    return jsonify({"error": str(e)}), 500

# Obtiene la ultima medicion del paciente segun sea el ID ingresado

@app.route('/patients/<int:patient_id>/latest', methods=['GET'])
def get_patient_latest_data(patient_id):
    """Devuelve la última medición de un paciente."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE patient_id = %s
            ORDER BY timestamp DESC LIMIT 1
        """, (patient_id,))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results

```

```

    ]
    cursor.close()
    conn.close()

    # Devolver JSON con formato legible (pretty print)
    return app.response_class(
        response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
# Agregar indentación
        status=200,
        mimetype="application/json"
    )
except Exception as e:
    return jsonify({"error": str(e)}), 500

# Filtrar por tipo de dato fisiologico asociado a un paciente

@app.route('/patients/<int:patient_id>/<string:physiological_type>',
methods=['GET'])
def get_patient_data_by_type(patient_id, physiological_type):
    """Devuelve todas las mediciones de un tipo específico de un paciente."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE patient_id = %s AND physiological_type = %s
            """, (patient_id, physiological_type))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results

```

```

    ]
    cursor.close()
    conn.close()

    # Devolver JSON con formato legible (pretty print)
    return app.response_class(
        response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
# Agregar indentación
        status=200,
        mimetype="application/json"
    )
except Exception as e:
    return jsonify({"error": str(e)}), 500

# Obtiene la ultima medicion del paciente segun sea el ID ingresado y el tipo
fisiologico especificado

@app.route('/patients/<int:patient_id>/<string:physiological_type>/latest',
methods=['GET'])
def get_latest_patient_data_by_type(patient_id, physiological_type):
    """Devuelve la última medición de un tipo específico de un paciente."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE patient_id = %s AND physiological_type = %s
            ORDER BY timestamp DESC LIMIT 1
            """, (patient_id, physiological_type))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)

```

```

        ])
        for row in results
    ]
    cursor.close()
    conn.close()

    # Devolver JSON con formato legible (pretty print)
    return app.response_class(
        response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
# Agregar indentación
        status=200,
        mimetype="application/json"
    )
except Exception as e:
    return jsonify({"error": str(e)}), 500

# Listar todos los dispositivos que han hecho un registro fisiologico

@app.route('/devices', methods=['GET'])
def get_devices():
    """Lista todos los dispositivos registrados."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        # Obtener pacientes únicos (ID + Nombre)
        cursor.execute("""
            SELECT DISTINCT device_id, device_name
            FROM observations
            WHERE device_id IS NOT NULL AND device_name IS NOT NULL
            ORDER BY device_name ASC
        """)
        results = cursor.fetchall()

        cursor.close()
        conn.close()

        # Formato de respuesta con mensaje adicional
        response = {
            "message": "Lista de dispositivos que tienen un registro",
            "results": results
        }

    return jsonify(response)

```

```

except Exception as e:
    return jsonify({"error": str(e)}), 500

# Obtiene todas las mediciones de un dispositivo específico

@app.route('/devices/<int:device_id>', methods=['GET'])
def get_device_data(device_id):
    """Devuelve todas las mediciones de un dispositivo específico."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE device_id = %s
            """, (device_id,))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results
        ]
        cursor.close()
        conn.close()

        # Devolver JSON con formato legible (pretty print)
        return app.response_class(
            response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
            # Agregar indentación
            status=200,
            mimetype="application/json"
        )
    except Exception as e:

```

```

        return jsonify({"error": str(e)}), 500

# Listar todos los medicos que han hecho un registro fisiologico

@app.route('/doctors', methods=['GET'])
def get_doctors():
    """Lista todos los médicos registrados."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        # Obtener pacientes únicos (ID + Nombre)
        cursor.execute("""
            SELECT DISTINCT doctor_id, doctor_name
            FROM observations
            WHERE doctor_id IS NOT NULL AND doctor_name IS NOT NULL
            ORDER BY doctor_name ASC
        """)
        results = cursor.fetchall()

        cursor.close()
        conn.close()

        # Formato de respuesta con mensaje adicional
        response = {
            "message": "Lista de medicos que han realizado un registro",
            "results": results
        }

        return jsonify(response)
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Devuelve todas las mediciones registradas por un médico en particular.

@app.route('/doctors/<int:doctor_id>', methods=['GET'])
def get_doctor_data(doctor_id):
    """Devuelve todas las mediciones registradas por un médico en particular."""
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

```

```

        cursor.execute("""
            SELECT id, patient_id, physiological_type, value, value2, unit,
timestamp
            FROM observations WHERE doctor_id = %s
        """, (doctor_id,))
        results = cursor.fetchall()

        # Convertir cada fila a OrderedDict y formatear timestamp
        ordered_results = [
            OrderedDict([
                ("id", row["id"]),
                ("patient_id", row["patient_id"]),
                ("physiological_type", row["physiological_type"]),
                ("value", row["value"]),
                ("value2", row["value2"]),
                ("unit", row["unit"]),
                ("timestamp", row["timestamp"].strftime("%Y-%m-%d %H:%M:%S") if
row["timestamp"] else None)
            ])
            for row in results
        ]
        cursor.close()
        conn.close()

        # Devolver JSON con formato legible (pretty print)
        return app.response_class(
            response=json.dumps(ordered_results, ensure_ascii=False, indent=4),
            # Agregar indentación
            status=200,
            mimetype="application/json"
        )
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Devuelve el hash de contraseña del usuario si existe.

@app.route('/users/<string:username>', methods=['GET'])
def get_user_by_username(username):
    try:
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        cursor.execute("SELECT username, password_hash FROM profile WHERE
username = %s", (username,))

```

```

        user = cursor.fetchone()

        cursor.close()
        conn.close()

        if user:
            return jsonify(user), 200
        else:
            return jsonify({"error": "Usuario no encontrado"}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/users', methods=['POST'])
def register_user():
    """Registra un nuevo usuario en la base de datos."""
    try:
        data = request.get_json()

        if not data or "username" not in data or "password" not in data:
            return jsonify({"error": "Faltan campos requeridos: username y password"}), 400

        username = data["username"].strip()
        password = data["password"]

        if len(password) < 8 or password.isdigit() or password.isalpha():
            return jsonify({"error": "La contraseña debe tener al menos 8 caracteres y combinar letras y números"}), 400

        hashed_password = generate_password_hash(password)

        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute("""
            INSERT INTO profile (username, password_hash)
            VALUES (%s, %s)
            """, (username, hashed_password))
        conn.commit()

        cursor.close()
        conn.close()

```

```

        return jsonify({"message": "Usuario registrado con éxito"}), 201

except mysql.connector.errors.IntegrityError:
    return jsonify({"error": "El usuario ya existe"}), 409

except Exception as e:
    return jsonify({"error": str(e)}), 500

@app.route('/users/<string:username>', methods=['PATCH'])
def update_user_password(username):
    """Actualiza la contraseña de un usuario."""
    try:
        data = request.get_json()
        new_password = data.get("new_password")

        if not new_password or len(new_password) < 8:
            return jsonify({"error": "La nueva contraseña debe tener al menos 8
caracteres."}), 400

        hashed_password = generate_password_hash(new_password)

        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute("""
            UPDATE profile SET password_hash = %s WHERE username = %s
            """, (hashed_password, username))
        conn.commit()

        updated_rows = cursor.rowcount

        cursor.close()
        conn.close()

        if updated_rows == 0:
            return jsonify({"error": "Usuario no encontrado."}), 404

        return jsonify({"message": "Contraseña actualizada correctamente."}),
200

except Exception as e:
    return jsonify({"error": str(e)}), 500

```

```

def validate_patient_and_med(id_paciente, id_medico):
    """
    Verifica si paciente y médico existen en GNU Health
    """
    conn = None
    try:
        conn = psycopg2.connect(**DB_CONFIG_GNU)
        cur = conn.cursor()

        # Validar paciente
        cur.execute("SELECT id FROM party_party WHERE ref = %s", (id_paciente,))
        patient_party = cur.fetchone()
        if not patient_party:
            return False, "Paciente no encontrado en GNU Health"
        cur.execute("SELECT id FROM gnuhealth_patient WHERE name = %s",
(patient_party[0],))
        patient = cur.fetchone()
        if not patient:
            return False, "El ID ingresado si existe en la base de datos de GNU
Health, pero no esta registrado como paciente"

        # Validar médico
        cur.execute("SELECT id FROM party_party WHERE ref = %s", (id_medico,))
        med_party = cur.fetchone()
        if not med_party:
            return False, "Médico no encontrado en GNU Health"
        cur.execute("SELECT id FROM gnuhealth_healthprofessional WHERE name =
%s", (med_party[0],))
        med = cur.fetchone()
        if not med:
            return False, "El ID ingresado si existe en la base de datos de GNU
Health, pero no esta registrado como paciente"

        return True, "Paciente y médico existen en GNU Health"

    except Exception as e:
        return False, f"Error validando en GNU Health: {e}"
    finally:
        if conn:
            cur.close()
            conn.close()

@app.route('/mqtt/setting', methods=['POST'])
def mqtt_send_setting():

```

```

try:
    data = request.get_json()
    if not data:
        return jsonify({"error": "No se recibió ningún JSON"}), 400

#----- VALIDACION DE QUE EXISTEN IDS EN GNU HEALTH-----
-----
    id_paciente = data.get("id_paciente")
    id_med = data.get("id_med")

    # 1. Validar existencia en GNU Health
    valido, msg = validate_patient_and_med(id_paciente, id_med)
    if not valido:
        return jsonify({
            "status": "error",
            "message": msg
        }), 400

#----- ENVIO DE PARAMETROS-----
-----
    # 1. Publicar en MQTT para que se configure el Arduino
    publish.single("sunnyhealth/setting",                json.dumps(data),
hostname=MQTT_BROKER, port=MQTT_PORT)

    # 2. Mandar también el JSON a Merry API
    try:
        response = requests.post(MERRY_URL, json=data, timeout=5)
        response.raise_for_status()
        merry_msg = response.json()
    except Exception as e:
        merry_msg = {"error": f"No se pudo contactar con Merry API: {str(e)}"}

    return jsonify({
        "message": "Parámetros enviados correctamente a Merry API",
        "merry_aswer": merry_msg,
        "mqtt": "Parametros publicados en topic: sunnyhealth/setting"
    }), 200

except Exception as e:
    return jsonify({"error": str(e)}), 500

@app.route('/mqtt/command', methods=['POST'])

```

```

def mqtt_send_command():
    try:
        data = request.get_json()

        if not data or "tipo_dato_fisiologico" not in data:
            return jsonify({"error": "Falta el campo 'tipo_dato_fisiologico'"}), 400

        # Validar si "valor" existe o no (es opcional)
        mensaje = {
            "tipo_dato_fisiologico": data["tipo_dato_fisiologico"]
        }

        if "valor" in data and data["valor"]:
            mensaje["valor"] = data["valor"]

        id_device = data.get("id_device")
        name_device = data.get("device")

        topic = f"sunnyhealth/{name_device}/{id_device}/command"

        publish.single(topic, json.dumps(mensaje), hostname=MQTT_BROKER,
port=MQTT_PORT)

        return jsonify({
            "message": "Comando enviado correctamente al topic segun
configuracion",
            "contenido_enviado": mensaje
        }), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/datos/ingresar', methods=['POST'])
def recibir_dato():
    contenido = request.json
    topic = contenido.get("topic")
    data = contenido.get("data")

    print(f"[Sunny] Recibido desde Merry: {topic}")
    gnu_ok = False
    maria_ok = False
    errors = []

```

```

# Guardar en GNU Health
try:
    gnu_ok= save_to_db_GNU(data)
    if not gnu_ok:
        errors.append("GNU Health validation or insert failed")
except Exception as e:
    errors.append(f"[Sunny] Error guardando en GNU Health: {str(e)}")

# Guardar en MariaDB
try:
    maria_ok=save_to_db_Maria(data)
    if not maria_ok:
        errors.append("MariaDB insert failed")

except Exception as e:
    errors.append(f"[Sunny] Error guardando en MariaDB: {str(e)}")
    maria_ok=False

# Evaluación global
if gnu_ok and maria_ok:
    http_status = 200
    status = "ok"
    message = "Data stored successfully in all systems"

elif gnu_ok or maria_ok:
    http_status = 207 # Multi-Status
    status = "partial"
    message = "Data stored partially"

else:
    http_status = 400
    status = "error"
    message = "Data rejected by Sunny API"

# Respuesta final
return jsonify({
    "status": status,
    "GNU_DB": "success" if gnu_ok else "error",
    "Maria_DB": "success" if maria_ok else "error",
    "message": message,
    "errors": errors
}), http_status

```

```
if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

B.2. Código del servicio Merry API

```
# merry.py
from paho.mqtt.client import Client
from flask import Flask, request
import threading
import json
import requests
from datetime import datetime

MQTT_BROKER = "localhost"
MQTT_PORT = 1883

# ----- MQTT -----
def on_connect(client, userdata, flags, rc):
    print("[Merry] Conectado al broker MQTT")

def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    topic = msg.topic
    print(f"[Merry] Recibido: {topic} -> {payload}")

    try:
        # Parsear payload como JSON
        json_data = json.loads(payload)

        # 1. Enviar a Sunny Health (API) por HTTP
        res = requests.post("http://localhost:5000/datos/ingresar", json={
            "topic": topic,
            "data": json_data
        })
        response = res.json()

        sunny_status = response.get("status", "unknown")

        # 2. Construir tópico dinámico de salida
        partes = msg.topic.split("/")

        # Esperamos algo como: [ESP32, 54127893, temperature, 2016, in]
        if len(partes) >= 5:
            device = partes[0]          # ESP32
            patient_id = partes[1]      # 54127893
```

```

        device_id = partes[3]          # 2016
        topic_out = f"{device}/{patient_id}/{device_id}/out"
    else:
        topic_out = "ESP32/unknown/unknown/out"

    # 3. Construir payload de confirmación
    out_payload = {
        "sunny_status": sunny_status,
        "GNU_DB": response.get("GNU_DB"),
        "Maria_DB": response.get("Maria_DB"),
        "message": response.get("message"),
        "errors": response.get("errors"),
        "timestamp": datetime.now().isoformat(),
        "source_topic": topic
    }

    # 4. Publicar en MQTT el tópico dinámico
    client.publish(topic_out, json.dumps(out_payload))
    print(f"[Merry] Publicado en {topic_out}: {out_payload}")

except json.JSONDecodeError:
    print(f"[Merry] Error: el payload no es un JSON válido -> {payload}")

except Exception as e:
    print(f"[Merry] Error al procesar mensaje: {e}")

# ----- TOPICS -----
def construir_topics(data):
    device = data["device"]
    id_paciente = data["id_paciente"]
    id_device = data["id_device"]

    topics = [
        f"{device}/{id_paciente}/temperature/{id_device}/in",
        f"{device}/{id_paciente}/blood_pressure/{id_device}/in",
        f"{device}/{id_paciente}/glycemia/{id_device}/in",
        f"{device}/{id_paciente}/respiratory_rate/{id_device}/in",
        f"{device}/{id_paciente}/oxygen_saturation/{id_device}/in",
        f"{device}/{id_paciente}/heart_rate/{id_device}/in",
    ]

    return topics

# ----- API -----
app = Flask(__name__)

```

```

@app.route('/suscribir', methods=['POST'])
def agregar_topico():
    data = request.json
    topics = construir_topics(data)
    for t in topics:
        client.subscribe(t)
        print(f"[Merry] Suscrito a {t}")
    return {"status": "ok", "topics": topics}

# ----- MAIN -----
if __name__ == '__main__':
    client = Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_BROKER, MQTT_PORT, 60)

    # Loop MQTT en un hilo
    threading.Thread(target=client.loop_forever, daemon=True).start()

    # API Flask
    app.run(port=5001)

```

C. Códigos desarrollados para la interfaz en Streamlit

C.1. Código principal de la interfaz de Streamlit

```
import streamlit as st
import login
import home
import register
import change_password
import Details
import patients
import patient_details
import data_view
import setting

# Inicializar el estado si no existe
if "page" not in st.session_state:
    st.session_state["page"] = "login"

# Barra lateral para la navegación
with st.sidebar:
    st.title("Navegación")
    if st.session_state.get("authenticated", False): # Solo mostrar si el
usuario está autenticado
        if st.button("Inicio"):
            st.session_state["page"] = "home"
            st.rerun()
        if st.button("Pacientes"):
            st.session_state["page"] = "patients"
            st.rerun()
        if st.button("Envio de Datos Microcontrolador"):
            st.session_state["page"] = "setting"
            st.rerun()
        if st.button("Cerrar sesión"):
            st.session_state["authenticated"] = False
            st.session_state["page"] = "login"
            st.rerun()

# Manejo de navegación entre páginas
if st.session_state["page"] == "login":
    login.show_login()
elif st.session_state["page"] == "register":
    register.show_register()
elif st.session_state["page"] == "home":
```

```

    home.show_home()
elif st.session_state["page"] == "change_password":
    change_password.show_change_password()
elif st.session_state["page"] == "Details":
    Details.show_details()
elif st.session_state["page"] == "patients":
    patients.show_patients()
elif st.session_state["page"] == "patient_details":
    patient_details.show_patient_details()
elif st.session_state["page"] == "data_view":
    data_view.show_data_view()
elif st.session_state["page"] == "setting":
    setting.show_setting()

```

C.2. Código de la pantalla principal (“Home”)

```

import os
import streamlit as st
import pandas as pd
import requests
from PIL import Image

# Configurar conexión a la API
API_URL = "http://localhost:5000" # Ajusta si tu API está en otra IP o puerto

# Función para obtener los datos desde la API
def fetch_data():
    try:
        response = requests.get(f"{API_URL}/data")
        if response.status_code == 200:
            data = response.json()
            df = pd.DataFrame(data)
            return df
        else:
            st.error(f"Error al obtener datos de la API: {response.status_code}")
            return pd.DataFrame()
    except requests.exceptions.ConnectionError:
        st.error("No se pudo conectar con la API. Asegúrate de que esté corriendo.")
        return pd.DataFrame()

```

```

def show_home():
    # Crear columnas para el título y la imagen
    col1, col2 = st.columns([3, 1]) # Ajusta la proporción según necesites
    with col1:
        st.title("Sunny Health - Interfaz de seguimiento de datos médicos")
    with col2:
        # Ruta relativa a la imagen
        current_dir = os.path.dirname(__file__)
        image_path = os.path.join(current_dir, "images", "image2.png")
        image = Image.open(image_path)
        resized_image = image.resize((326, 294)) # Ajusta las dimensiones
        st.image(resized_image)

    # Código JavaScript para hacer scroll automático al inicio
    scroll_to_top = """
    <script>
        window.scrollTo(0, 0);
    </script>
    """
    st.markdown(scroll_to_top, unsafe_allow_html=True)

    st.write(f"      ###      Bienvenido,      {st.session_state.get('username',
'Usuario')}!")

    # Obtener y mostrar datos
    data = fetch_data()

    if data.empty:
        st.warning("No hay registros en la base de datos.")
    else:
        # Asegurar formato de timestamp como string (por si no lo es)
        if "timestamp" in data.columns:
            data["timestamp"] = pd.to_datetime(data["timestamp"]).astype(str)

        # Filtrar columnas para la vista principal
        selected_columns = ["id", "patient_id", "physiological_type", "value",
"value2", "unit", "timestamp"]
        data_filtered = data[selected_columns]

        st.write("#### Registros Médicos insertados en GNU Health")
        st.dataframe(data_filtered, hide_index=True)

```

```

        selected_id = st.selectbox("Selecciona un ID para ver detalles",
data["id"])

    if st.button("Ver detalles"):
        st.session_state["selected_id"] = selected_id
        st.session_state["page"] = "Details"
        st.rerun()

# Botón para cerrar sesión
if st.button("Cerrar Sesión"):
    st.session_state["authenticated"] = False
    st.session_state["page"] = "login"
    st.rerun()

```

C.3. Código de la pantalla de detalles de registro (“Details”)

```

import streamlit as st
import pandas as pd
import requests

API_URL = "http://localhost:5000"

def show_details():
    """Interfaz de detalles del registro médico."""

    # Verificar autenticación
    if not st.session_state.get("authenticated", False):
        st.error("Debes iniciar sesión primero.")
        st.session_state["page"] = "login"
        st.rerun()

def fetch_data_by_id(record_id):
    """Obtiene los detalles de un registro específico desde la API."""
    try:
        response = requests.get(f"{API_URL}/data/{record_id}")
        if response.status_code == 200:
            return response.json()
        elif response.status_code == 404:
            st.warning("El ID del registro no existe.")
            return None
        else:
            st.error(f"Error al obtener datos de la API:
{response.status_code}")
            return None
    except requests.exceptions.ConnectionError:

```

```

        st.error("No se pudo conectar con la API. Asegúrate de que esté
corriendo.")
        return None

# Obtener el ID seleccionado
selected_id = st.session_state.get("selected_id")

st.title("Detalles del Registro")
# 🔥 Código JavaScript para hacer scroll automático al inicio
scroll_to_top = """
<script>
    window.scrollTo(0, 0);
</script>
"""
st.markdown(scroll_to_top, unsafe_allow_html=True)

if selected_id is None:
    st.warning("No se ha seleccionado ningún registro.")
else:
    data = fetch_data_by_id(selected_id)

    if data:
        # Separar el full_message para mostrarlo en JSON
        full_message = data.pop("full_message", {})

        st.write("### Información General")
        st.dataframe(
            pd.DataFrame.from_dict(data, orient="index", columns=["Valor"])
            .rename_axis("Descripción")
        )

        st.write("### Datos FHIR (full_message)")
        st.json(full_message)
    else:
        st.error("El registro seleccionado ya no está disponible.")

# Botón para regresar al home
if st.button("Volver a la tabla general"):
    st.session_state["page"] = "home" # 🔥 Define la página en
session_state
    st.rerun() # 🔄 Recargar la aplicación para ir a Home

```

C.4. Código de la pantalla de lista de pacientes (“Patients”)

```

import streamlit as st
import pandas as pd
import requests

API_URL = "http://localhost:5000"

def fetch_patients():
    """Obtiene la lista de pacientes desde la API."""
    try:
        response = requests.get(f"{API_URL}/patients")
        if response.status_code == 200:
            json_data = response.json()
            return pd.DataFrame(json_data.get("results", []))
        else:
            st.error(f"Error al obtener pacientes: {response.status_code}")
            return pd.DataFrame()
    except requests.exceptions.ConnectionError:
        st.error("No se pudo conectar con la API. Asegúrate de que esté corriendo.")
        return pd.DataFrame()

def show_patients():
    st.title("Lista de Pacientes")

    patients = fetch_patients()

    if patients.empty:
        st.warning("No hay pacientes registrados.")
        return

    # Renombrar las columnas
    patients = patients.rename(columns={"patient_id": "ID del Paciente",
    "patient_name": "Nombre del Paciente"})

    # Mostrar tabla con los títulos corregidos
    st.write("### Pacientes Registrados")
    st.dataframe(patients, hide_index=True)

    # Selector solo con el ID del paciente
    selected_id = st.selectbox("Selecciona un paciente por su ID:", patients["ID del Paciente"])

    # Botón para ver detalles
    if st.button("Ver detalles del paciente"):

```

```

        st.session_state["selected_patient"] = selected_id
        st.session_state["page"] = "patient_details"
        st.rerun()

    if st.button("Volver a la pagina principal"):
        st.session_state["page"] = "home"
        st.rerun()

```

C.5. Código de la pantalla de los parámetros del paciente (“Patients_details”)

```

import streamlit as st
from utils import get_patient_name

def show_patient_details():
    # Verificar que haya un paciente seleccionado
    if "selected_patient" not in st.session_state:
        st.error("No se ha seleccionado un paciente. Regresa a la página anterior y elige uno.")
        st.stop()

    patient_id = st.session_state["selected_patient"]
    patient_name = get_patient_name(patient_id)

    st.title(f"Datos fisiológicos del paciente: {patient_name or 'Nombre no encontrado'}")

    # Tipos de datos fisiológicos y su clave en la base de datos
    data_types = {
        "Temperatura": "temperature",
        "Frecuencia Cardiaca": "heart_rate",
        "Presión Arterial": "blood_pressure",
        "Glicemia": "glycemia",
        "Frecuencia Respiratoria": "respiratory_rate",
        "Saturación de Oxígeno": "oxygen_saturation",
    }

    st.subheader("Selecciona un tipo de dato para ver los registros:")
    for label, type_key in data_types.items():
        if st.button(label):
            st.session_state["selected_data_type"] = type_key
            st.session_state["page"] = "data_view"
            st.rerun()

```

C.6. Código de la pantalla de visualización de tendencias de un parámetro (“Data-view”)

```

import streamlit as st
import pandas as pd
import plotly.express as px
from utils import get_patient_observations_by_type, get_patient_name

def show_data_view():
    if "selected_patient" not in st.session_state or "selected_data_type" not
in st.session_state:
        st.error("No se ha seleccionado paciente o tipo de dato.")
        st.stop()

    patient_id = st.session_state["selected_patient"]
    data_type = st.session_state["selected_data_type"]
    patient_name = get_patient_name(patient_id)

    st.title(f"{data_type.replace('_', ' ').capitalize()} de {patient_name}")

    # Obtener datos desde la API a través de utils
    observations = get_patient_observations_by_type(patient_id, data_type)

    if not observations:
        st.warning("No hay datos registrados para este tipo.")
        return

    # Convertir a DataFrame
    df = pd.DataFrame(observations)

    if "timestamp" not in df.columns or "value" not in df.columns:
        st.error("Los datos obtenidos no tienen el formato esperado.")
        return

    df["timestamp"] = pd.to_datetime(df["timestamp"])

    # Mostrar tabla
    st.subheader("Datos Registrados")
    st.dataframe(df, hide_index=True)

    # Mostrar gráfico
    st.subheader("Gráfico de evolución")
    if data_type == "blood_pressure" and "value2" in df.columns:
        # Renombrar columnas temporalmente
        df = df.rename(columns={"value": "Sistólica", "value2": "Diastólica"})

    # Convertir a formato long

```

```

        df_long = df.melt(id_vars="timestamp", value_vars=["Sistólica",
"Diastólica"],
                           var_name="Presión", value_name="Valor")

        fig = px.line(df_long, x="timestamp", y="Valor", color="Presión",
                      labels={"timestamp": "Fecha"}, title="Presión Arterial")

    else:
        fig = px.line(
            df,
            x="timestamp",
            y="value",
            labels={"value": "Valor", "timestamp": "Fecha"},
            title=data_type.replace("_", " ").capitalize()
        )

    st.plotly_chart(fig, use_container_width=True)

    if st.button("Volver a detalles del paciente"):
        st.session_state["page"] = "patient_details"
        st.rerun()

```

C.7. Código de funciones de ayuda para consultas a la API Sunny

```

import requests

API_URL = "http://localhost:5000"

def get_patient_name(patient_id):
    """Obtiene el nombre del paciente desde la API."""
    try:
        response = requests.get(f"{API_URL}/patients")
        if response.status_code == 200:
            patients = response.json().get("results", [])
            for patient in patients:
                if patient["patient_id"] == patient_id:
                    return patient["patient_name"]
    except Exception as e:
        print(f"Error al obtener nombre del paciente: {e}")
    return None

def get_patient_observations(patient_id):
    """Obtiene todas las observaciones de un paciente desde la API."""
    try:
        response = requests.get(f"{API_URL}/patients/{patient_id}")

```

```

        if response.status_code == 200:
            return response.json()
    except Exception as e:
        print(f"Error al obtener observaciones del paciente: {e}")
    return []

def get_patient_observations_by_type(patient_id, physiological_type):
    """Obtiene observaciones filtradas por tipo desde la API."""
    try:
        response = requests.get(f"{API_URL}/patients/{patient_id}/{physiological_type}")
        if response.status_code == 200:
            return response.json()
    except Exception as e:
        print(f"Error al obtener observaciones por tipo: {e}")
    return []

```

C.8. Código de la pantalla de acceso a la interfaz (“Login”)

```

import os
import streamlit as st
import requests
from werkzeug.security import check_password_hash

API_URL = "http://localhost:5000" # Ajusta si es necesario

def authenticate_user(username, password):
    """Valida credenciales usando la API."""
    try:
        response = requests.get(f"{API_URL}/users/{username.strip()}")
        if response.status_code == 200:
            user_data = response.json()
            stored_password_hash = user_data.get("password_hash")
            return check_password_hash(stored_password_hash, password)
        elif response.status_code == 404:
            return False
        else:
            st.error(f"Error al autenticar: {response.status_code}")
            return False
    except requests.exceptions.RequestException as e:
        st.error(f"No se pudo conectar a la API: {e}")
        return False

```

```

def show_login():
    current_dir = os.path.dirname(__file__)
    image_path = os.path.join(current_dir, "images", "image1.png")

    st.title("Inicio de Sesión - Sunny Health")
    st.image(image_path, use_container_width=True)

    st.markdown("<script>window.scrollTo(0, 0);</script>",
unsafe_allow_html=True)

    username = st.text_input("Usuario").strip()
    password = st.text_input("Contraseña", type="password")

    if st.button("Iniciar Sesión"):
        if not username or not password:
            st.error("⚠ Debes ingresar un usuario y contraseña.")
            return

        if authenticate_user(username, password):
            st.session_state["authenticated"] = True
            st.session_state["username"] = username
            st.session_state["page"] = "home"
            st.rerun()
        else:
            st.error("❌ Usuario o contraseña incorrectos.")

    if st.button("Registrarse"):
        st.session_state["page"] = "register"
        st.rerun()

    if st.button("¿Olvidaste tu contraseña?"):
        st.session_state["page"] = "change_password"
        st.rerun()

```

C.9. Código de la pantalla de registro de nuevos usuarios (“Register”)

```

import streamlit as st
import requests

API_URL = "http://localhost:5000"

def show_register():
    st.title("Registro de Usuario")

    scroll_to_top = ""

```

```

<script>
    window.scrollTo(0, 0);
</script>
"""
st.markdown(scroll_to_top, unsafe_allow_html=True)

username = st.text_input("Usuario").strip()
password = st.text_input("Contraseña", type="password")

if st.button("Registrar"):
    if not username or not password:
        st.error("⚠ Debes ingresar un usuario y contraseña.")
        return

    if len(password) < 8 or password.isdigit() or password.isalpha():
        st.error("⚠ La contraseña debe tener al menos 8 caracteres y
combinar letras y números.")
        return

    try:
        response = requests.post(f"{API_URL}/users", json={
            "username": username,
            "password": password
        })

        if response.status_code == 201:
            st.success("✅ Usuario registrado con éxito! Redirigiendo al login...")
            st.session_state["page"] = "login"
            st.rerun()
        elif response.status_code == 409:
            st.error("❌ El usuario ya existe. Intenta con otro nombre.")
        else:
            st.error(f"Error al registrar usuario:
{response.json().get('error', 'Error desconocido')}")

    except requests.exceptions.ConnectionError:
        st.error("No se pudo conectar con la API. ¿Está corriendo?")

if st.button("Volver al login"):
    st.session_state["page"] = "login"
    st.rerun()

```

C.10. Código de la pantalla de cambio de contraseña (“Change-password”)

```
import streamlit as st
```

```

import requests

API_URL = "http://localhost:5000"

def show_change_password():
    st.title("🔒 Cambiar Contraseña")

    # 🔥 Código JavaScript para hacer scroll automático al inicio
    scroll_to_top = """
        <script>
            window.scrollTo(0, 0);
        </script>
    """
    st.markdown(scroll_to_top, unsafe_allow_html=True)

    username = st.text_input("Usuario").strip()
    new_password = st.text_input("Nueva Contraseña", type="password")

    if st.button("Actualizar Contraseña"):
        if not username or not new_password:
            st.error("⚠️ Debes ingresar un usuario y una nueva contraseña.")
            return

        if len(new_password) < 8:
            st.error("🔑 La contraseña debe tener al menos 8 caracteres.")
            return

        try:
            response = requests.patch(
                f"{API_URL}/users/{username}",
                json={"new_password": new_password}
            )

            if response.status_code == 200:
                st.success("✅ Contraseña actualizada con éxito. Ahora inicia sesión.")
                st.session_state["page"] = "login"
                st.rerun()
            elif response.status_code == 404:
                st.error("❌ Usuario no encontrado. Verifica el nombre.")
            else:
                st.error(f"⚠️ Error del servidor: {response.json().get('error', 'Error desconocido')}")

```

```

except requests.exceptions.ConnectionError:
    st.error("🚫 No se pudo conectar con la API. Asegúrate de que esté corriendo.")

if st.button("Volver al login"):
    st.session_state["page"] = "login"
    st.rerun()

```

C.11. Código de la pantalla de control remoto de nodo Simulador (“Setting”)

```

import streamlit as st
import requests

API_URL = "http://localhost:5000" # Cambia si tu API Flask está en otro host o puerto

def show_setting():

    # Verificar autenticación
    if not st.session_state.get("authenticated", False):
        st.error("Debes iniciar sesión primero.")
        st.session_state["page"] = "login"
        st.rerun()

    # Diccionario de opciones visibles → código corto a enviar
    mapa_datos = {
        "Temperatura corporal": "T",
        "Presión arterial": "P",
        "Frecuencia cardiaca": "F",
        "Frecuencia respiratoria": "R",
        "Saturacion de oxígeno": "O",
        "Glicemia (glucosa en sangre)": "G"
    }

    # Inicializar estado de configuración si no existe
    if "configurado" not in st.session_state:
        st.session_state.configurado = False

    # INSTRUCCIONES
    st.title("Sunny Health - Plataforma de envío de datos fisiológicos")
    st.info("""
    **Instrucciones de uso:**

```

1. Primero, completa y envía los parámetros del paciente, médico y dispositivo.

2. Luego podrás seleccionar el tipo de dato fisiológico para enviar y al enviarlo se enviara un dato aleatorio.

3. Por otro lado, puedes ingresar manualmente un valor si lo deseas (por ejemplo: 37 para temperatura, 120 80 para presión arterial, etc.).

⚠ No podrás enviar datos hasta que los parámetros estén correctamente configurados.

```
""")

# Sección: Configuración de parámetros
st.subheader("Configuración del paciente, médico y dispositivo")

name_paciente = st.text_input("Nombre del paciente")
id_paciente = st.text_input("ID del paciente")
name_medical = st.text_input("Nombre del médico")
id_medical = st.text_input("ID del médico")
name_device = st.text_input("Nombre del dispositivo")
id_device = st.text_input("ID del dispositivo")

if st.button("Enviar parámetros"):
    if all([name_paciente, id_paciente, name_device, id_device,
name_medical, id_medical]):
        mensaje_parametros = {
            "paciente": name_paciente,
            "id_paciente": id_paciente,
            "med": name_medical,
            "id_med": id_medical,
            "device": name_device,
            "id_device": id_device
        }
        try:
            response = requests.post(f"{API_URL}/mqtt/setting",
json=mensaje_parametros)
            response.raise_for_status()

            st.success("✅ Parámetros configurados correctamente.")
            st.session_state.configurado = True
            st.session_state.device = name_device
            st.session_state.id_device = id_device
            st.json(response.json())

        except requests.exceptions.HTTPError:
```

```

        st.error(f"⚠ Error del servidor: {response.text}")

    except requests.exceptions.RequestException as e:
        st.error(f"❌ Error de conexión con la API: {e}")

    else:
        st.warning("Por favor, completa todos los campos para enviar los
parámetros.")

# Sección: Selección de tipo de dato
st.subheader("Enviar dato fisiologico con valor aleatorizado")
tipo_dato_fisiologico_visible = st.selectbox(
    "Selecciona el tipo de dato fisiológico",
    list(mapa_datos.keys())
)

codigo_dato = mapa_datos[tipo_dato_fisiologico_visible]

if st.button("Enviar dato"):
    if st.session_state.configurado:
        mensaje_comando = {
            "tipo_dato_fisiologico": codigo_dato,
            "device": st.session_state.device,
            "id_device": st.session_state.id_device
        }
        try:
            response = requests.post(f"{API_URL}/mqtt/command",
json=mensaje_comando)
            if response.status_code == 200:
                st.success("Tipo de dato enviado correctamente.")
                st.json(response.json())
            else:
                st.error(f"Error al enviar comando: {response.text}")
        except requests.exceptions.RequestException as e:
            st.error(f"Error de conexión con la API: {e}")
    else:
        st.error("Error: Debes configurar primero los parámetros antes de
enviar datos.")

# Sección: Ingreso manual de valor
st.subheader("Enviar dato fisiológico con valor ingresado")

tipo_dato_manual = st.selectbox("Tipo de dato a ingresar con valor",
list(mapa_datos.keys()), key="dato_con_valor")

```

```

    valor_ingresado = st.text_input("Ingresa el valor (ej. 37 para temperatura
o 120 80 para presión arterial)")

    if st.button("Enviar dato personalizado"):
        if st.session_state.configurado:
            if valor_ingresado:
                mensaje_valor = {
                    "tipo_dato_fisiologico": mapa_datos[tipo_dato_manual],
                    "valor": valor_ingresado,
                    "device": st.session_state.device,
                    "id_device": st.session_state.id_device
                }
                try:
                    response = requests.post(f"{API_URL}/mqtt/command",
json=mensaje_valor)
                    if response.status_code == 200:
                        st.success("Dato con valor enviado correctamente.")
                        st.json(response.json())
                    else:
                        st.error(f"Error al enviar comando con valor:
{response.text}")
                except requests.exceptions.RequestException as e:
                    st.error(f"Error de conexión con la API: {e}")
                else:
                    st.warning("Por favor, ingresa un valor para enviarlo.")
            else:
                st.error("Error: Debes configurar primero los parámetros antes de
enviar datos.")

    if st.button("Volver a la página principal"):
        st.session_state.page = "home"
        st.rerun()

```