Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Healthsync-AI: Health Monitoring and AI-Driven Emergency System

Ayush Nanekar
Dr. D. Y. Patil Technical Campus
Anant Kumbhar
Dr. D. Y. Patil Technical Campus
Shubham Mutkule
Dr. D. Y. Patil Technical Campus
Sarthak Wagh
Dr. D. Y. Patil Technical Campus
Soni Ragho
sonirragho@gmail.com
Dr. D. Y. Patil Technical Campus

Research Article

Keywords: HealthSync AI, Artificial Intelligence (AI), Internet of Things (IoT), Health Monitoring, Bluetooth Low Energy (BLE), Outbreak Analysis

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11555807/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Chapter I: INTRODUCTION

PROBLEM STATEMENT

Right now, our healthcare system is fundamentally reactive, meaning people usually only end up in a hospital after their symptoms have already become severe. Most healthcare apps available today are pretty limited, functioning mostly as digital filing cabinets for storing medical records or simple tools for booking appointments. They completely miss the mark when it comes to everyday life because they fail to provide continuous health tracking, early warnings, or real-time emergency response. This leaves many individuals—especially those living in rural areas or dealing with sudden emergencies—completely stranded without immediate medical guidance.

On a broader scale, public health authorities are constantly fighting an uphill battle. Because medical data is scattered across different systems and reporting is often delayed, officials struggle to spot disease outbreaks at an early stage when intervention could actually save lives. Ultimately, we have plenty of tech for managing paperwork and schedules, but we lack an intelligent, continuous safety net that protects individuals day-to-day and helps communities stay ahead of health crises.

Healthcare is traditionally reactive, with patients typically visiting hospitals only after their symptoms become severe. Most existing healthcare apps are limited to storing medical records or scheduling appointments, leaving a critical gap in continuous monitoring, early disease detection, and real-time emergency response. This leaves many individuals—particularly in rural areas or during urgent situations—without immediate medical guidance, while health authorities frequently struggle to identify disease outbreaks early due to fragmented data and delayed reporting. HealthSync AI addresses these challenges by offering a comprehensive, intelligent platform designed for continuous health monitoring, smart symptom analysis, predictive risk assessment, and rapid emergency support, ultimately empowering patients, clinicians, and public health officials to make timely, data-driven decisions.

MOTIVATION

Building HealthSync AI began with an extensive review of existing research literature and open-source health-monitoring platforms. Studying these academic papers and projects provided invaluable insights into the current landscape of digital healthcare, highlighting both the immense strengths of prior work and the specific gaps that inspired this project.

Reviewing the literature revealed two distinct, highly informative approaches in the academic space. Many studies focused deeply on advanced machine learning models—utilizing algorithms like Random Forest, Support Vector Machines, and Principal Component Analysis for data cleaning and classification—while others concentrated primarily on the hardware mechanics of sensor data collection. Analyzing these studies made it clear how powerful algorithmic data processing and robust sensor integration can be, directly guiding the analytical foundations built into HealthSync AI.

At the same time, exploring open-source health-tracking projects offered a clear view of practical limitations in current builds. Most existing repositories tend to focus narrowly on tracking single metrics, such as basic heart rate and SpO2 levels. Recognizing this limitation sparked the ambition to scale up significantly, leading HealthSync AI to ingest and process eleven distinct telemetry streams to provide a much fuller, more comprehensive picture of a patient's health rather than just a single data point.

Further research into existing commercial and open-source applications uncovered other missing pieces in the ecosystem. While some platforms excel at storing digital medical documents and others offer quick ways to find nearest-hospital contacts, very few attempt live, continuous tracking, and virtually none incorporate outbreak analysis. Traditional health systems suffer from severely fragmented data, leaving authorities struggling to spot outbreaks early. Additionally, many modern apps rely heavily on generic third-party API wrapper chatbots rather than deeply integrated, domain-specific intelligence.

Learning from the strengths of these foundational works and addressing their structural limitations shaped HealthSync AI into a unifying solution. Instead of leaving vital features scattered across disconnected apps, this platform bridges the divides by merging comprehensive multi-parameter telemetry, local SQLite persistence, a native PySide6 desktop interface, emergency response features, and proactive outbreak analysis into a single, cohesive, and modular ecosystem.

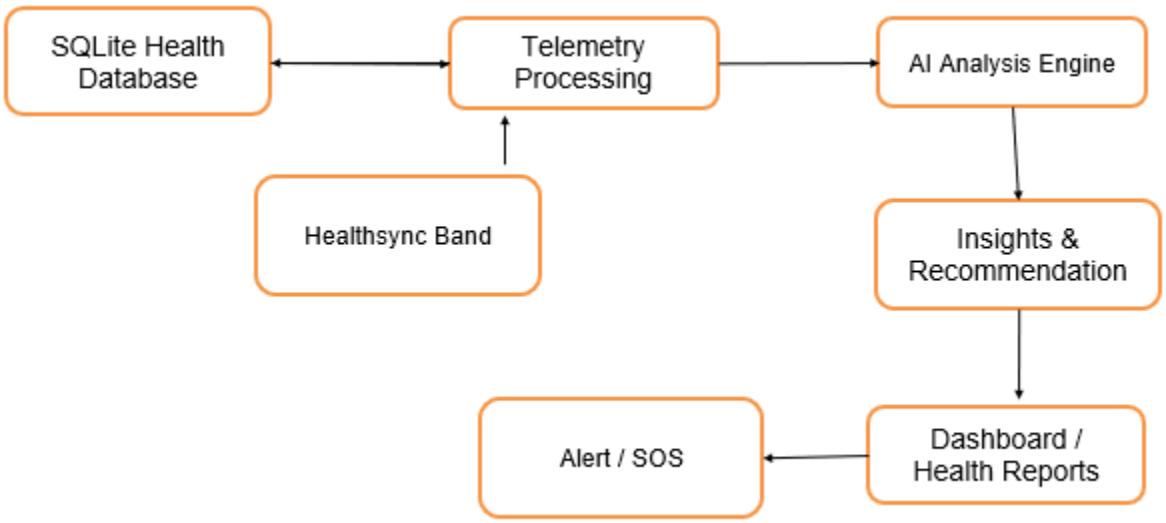
PROPOSED SOLUTION

HealthSync AI is engineered directly around the gaps found in existing healthcare technology, turning a fragmented and reactive landscape into a unified, proactive ecosystem. Rather than forcing users to juggle separate tools for digital document filing, basic heart rate tracking, and emergency lookups, the platform brings everything together into a seamless desktop experience built with a modular Python and PySide6 architecture.

To move far beyond the single-metric limitations of typical open-source projects, HealthSync AI is designed to ingest and process eleven distinct telemetry streams simultaneously, offering a complete and continuous window into a user's physiological state. This multi-parameter data doesn't just sit in a silo; it is securely managed through local SQLite persistence and channeled into an intelligent backend framework that processes structured health observations rather than relying on superficial, generic API wrapper chatbots.

The platform also tackles the broader systemic issues of delayed reporting and fragmented medical data by introducing integrated outbreak analysis. While traditional systems leave public health authorities struggling to spot community-level risks in time, HealthSync AI bridges individual continuous monitoring with aggregated insights to help track potential health threats at an early stage. Combined with built-in emergency response features and a clean separation of responsibilities across its BLE, database, and UI layers, HealthSync AI transforms personal wellness tracking and community health intelligence into a single, cohesive, and future-ready solution.

DIAGRAM



OBJECTIVES

Monitor real-time health and activity data: Instead of making you wait until a doctor's visit, the system constantly keeps tabs on how your body is doing right now, streaming live updates on both your physical vitals and everyday movements.

Detect abnormal vital signs and unusual health patterns: The app doesn't just collect numbers blindly; it pays close attention in the background to spot any sudden spikes, drops, or irregular behaviors that might mean something is off.

Analyze heart rate, SpO_2, blood pressure, and temperature: It digs deep into the core vitals that matter most, keeping a steady watch on your heart rhythm, oxygen saturation levels, blood pressure ranges, and body heat.

Monitor activity such as steps, distance, calories, and active time: Beyond medical vitals, it tracks your daily fitness journey by counting your steps, measuring how far you've moved, estimating calories burned, and logging your active minutes.

Analyze historical health data and identify trends: By looking back at your past logs over days and weeks, the system connects the dots to show you long-term patterns in your health and fitness habits.

Generate personalized AI-based health insights and recommendations: Rather than throwing generic advice at you, the built-in intelligence looks at your unique data to offer custom tips and suggestions tailored specifically to your body.

Provide early warnings for unusual or potentially concerning readings: If your numbers start heading in a risky direction, the app gives you a heads-up early on so you can take action before a minor issue turns into a real problem.

Trigger emergency alerts when critical conditions are detected: In severe situations where vitals cross safe boundaries, the platform acts immediately by sending out critical alerts to make sure help isn't delayed.

Maintain user health records and monitoring statistics: It acts as your reliable digital journal, safely storing all your past readings and check-in stats so you can easily review your health history whenever needed.

Provide outbreak analysis using data visualization: Taking a step beyond individual care, the platform uses clear visual charts to map out broader health trends, helping catch potential community outbreaks early on.

Chapter II: PROPOSED METHODOLGY

BACKGROUND WORK

Understanding the architecture of HealthSync AI requires examining the core technologies, protocols, and engineering principles that make continuous digital health tracking possible. Rather than relying on a single monolithic system, modern digital health applications depend on the convergence of real-time wireless communication, modular desktop design, secure local persistence, and intelligent data analysis.

In continuous health monitoring, collecting real-time data efficiently is a primary engineering hurdle. Traditional Bluetooth connections often consume too much power for wearable devices designed to run for days or weeks. Bluetooth Low Energy (BLE) solves this by using low-power radio frequencies and short connection intervals to stream vital telemetry—such as heart rate, SpO_2, blood pressure, and motion data—without draining battery life. Integrating a dedicated BLE layer allows the software to establish stable connections with both physical medical devices and simulated IoT nodes, creating a reliable stream of live physical data.

While web applications are popular, local desktop applications built with native graphical user interface (GUI) toolkits offer significant advantages for continuous telemetry processing. Python combined with PySide6 (the official Qt framework bindings) provides a high-performance environment capable of rendering live charts, managing asynchronous background threads, and handling immediate user interactions without lagging. Using a dedicated UI framework ensures that the presentation layer remains completely separated from backend data collection and processing rules.

Health applications handle sensitive, highly continuous data streams that require instant access and reliable storage. Local, serverless database engines like SQLite provide an ideal solution for storing personal health logs, session statistics, and configuration settings locally on the client machine. This approach minimizes latency, ensures that basic system features remain functional even without an active internet connection, and gives users direct control over where their health records live.

A core principle in software engineering is separating distinct duties into clear software modules. In health applications, bundling database access, Bluetooth data parsing, and user interface rendering into a single script creates code that is difficult to debug, maintain, or scale. Designing the software around modular services ensures that each layer—from BLE telemetry reception and SQLite storage to intelligent analytics and PySide6 presentation—operates independently. This modular foundation simplifies future development, making it straightforward to adapt backend services for mobile platforms like Android or iOS down the road.

EXISITNG TECHNOLOGIES

Examining open-source health platforms and digital tracking solutions reveals important progress in modern digital health, but also highlights clear boundaries where current tools stop short.

In the open-source hardware domain, Project_HeartBeat created by AashiDutt serves as an end-to-end pulse monitoring workflow. It interfaces directly with an Arduino microcontroller connected to an optical pulse sensor. When the hardware streams raw data over a serial connection, a Flask web dashboard runs backend Python scripts that organize the incoming signals into Pandas data frames. Moving average algorithms estimate peak pulse intervals and calculate overall beats per minute. Matplotlib then plots the heart rate waveforms, and Report Lab compiles the visual charts, patient metadata, and captured readings into a static PDF report for manual offline download.

Taking a different approach, LifeRhythm created by nishanth1707 focuses on an embedded IoT setup for real-time vital tracking. Built around an ESP32 microcontroller wired to biomedical sensors like the MAX30102, it captures live heart rate and blood oxygen saturation readings. The ESP32 broadcasts these metrics wirelessly over Bluetooth Low Energy and Wi-Fi. Data packets stream through WebSockets into a Web dashboard built with React and TypeScript, giving users live visual feedback on pulse waveforms and vital status updates.

At the public health scale, disease surveillance technologies like DHIS2 and spatial epidemiology platforms like HealthMap operate far above individual consumer apps. These platforms gather anonymized case reports, regional symptom spikes, and official health logs. Their engines process these data streams into geospatial heatmaps and time-series charts, applying statistical algorithms to spot regional clusters and alert health authorities to emerging outbreaks before emergency centers become overwhelmed.

Looking closely at existing open-source health-tracking tools and digital medical platforms reveals how far health technology has come, but also highlights where critical gaps remain. Open-source repositories like *Project_HeartBeat* (AashiDutt) and *LifeRhythm* (nishanth1707) have laid great groundwork for integrating IoT devices with software, providing practical ways to capture baseline health metrics like heart rate and pulse levels. Similarly, various academic and public health repositories focus on standalone epidemiological tracking or localized disease trend monitoring.

HOW HEALTHSYNC AI OUTSTANDS

However, looking at these existing solutions side-by-side reveals significant limitations:

Narrow Metric Tracking: Most community health-tracking projects focus exclusively on one or two basic vitals, such as heart rate and SpO_2, missing the broader picture that comes from tracking comprehensive physiological telemetry.

Disconnected Features: While some applications handle basic medical record filing, and others offer emergency contact lookups or single-purpose fitness logs, users are typically forced to juggle multiple completely separate apps to manage their overall well-being.

The Absence of Outbreak Intelligence in Personal Apps: Specialized epidemiological tools exist for public health professionals, but personal health-tracking applications almost never incorporate outbreak analysis or data visualization for community health tracking, leaving individual users disconnected from broader public health trends.

Reliance on Third-Party APIs: Many modern smart health apps lean heavily on external, third-party commercial APIs or generic wrapper chatbots to process user data and give feedback, limiting their independence and depth.

HealthSync AI bridges these disconnected areas into a single, cohesive desktop platform, standing out through four main architectural advancements:

1. Comprehensive Multi-Parameter Telemetry: While projects like LifeRhythm and Project_HeartBeat limit tracking to basic heart rate and blood oxygen levels, HealthSync AI expands data collection to process 11 distinct physiological and activity telemetry streams simultaneously.
2. In-House Trained Models: Many commercial applications lean on third-party commercial API wrapper chatbots or basic external scripts to generate recommendations. HealthSync AI operates using its own natively trained model, converting validated health metrics into structured observations, summaries, and actionable guidance directly within its backend logic.
3. Merging Personal Care with Community Outbreak Analysis: Instead of keeping personal wellness tracking separate from public health surveillance, HealthSync AI integrates built-in outbreak analysis visualization. It converts local continuous data into visual trend models, giving individuals and health authorities early visibility into potential community-wide outbreaks.
4. Unified, Desktop-Native Architecture: Rather than forcing users to juggle disconnected web scripts, separate filing tools, or web dependencies, HealthSync AI combines BLE IoT data streaming, local SQLite persistence, emergency response triggers, and outbreak visualization dashboards into one modular, reliable Python application built with PySide6.

LITERATURE SURVEY

No.	Paper title	Author(s)	Publication date	Short summary	Link
1	Recent Advances on IoT-Assisted Wearable Sensor Systems for Healthcare Monitoring	Shwetank Dattatraya Mamdiwar, Akshith R., Zainab Shakruwala, Utkarsh Chadha, Kathiravan Srinivasan, Chuan-Yu Chang	4 October 2021	Reviews wearable sensors, IoT architecture, communication, privacy, and healthcare monitoring challenges.	Read paper
2	Machine Learning for Healthcare Wearable Devices: The Big Picture	Farida Sabry, Tamer Eltaras, Wadha Labda, Khawla Alzoubi, Qutaibah Malluhi	18 April 2022	Explains how ML can analyse wearable health data and discusses reliability, privacy, and deployment challenges.	Read paper
3	The Emergence of AI-Based Wearable Sensors for Digital Health Technology: A Review	Shaghayegh Shajari, Kirankumar Kuruvinashetti, Amin Komeili, Uttandaraman Sundararaj	29 November 2023	Reviews AI-enabled wearable sensors, physiological signal processing, personalized monitoring, and sensor limitations.	Read paper
4	Leveraging Machine Learning for Personalized Wearable Biomedical Devices: A Review	Ali Olyanasab, Mohsen Annabestani	13 February 2024	Explores ML methods for personalized monitoring using electrical, chemical, and mechanical biomedical signals.	Read paper
5	Machine Learning Applied to Edge Computing and Wearable Devices for Healthcare: Systematic Mapping of the Literature	Carlos Vinicius Fernandes Pereira, Edvard Martins de Oliveira, Adler Diniz de Souza	29 September 2024	Analyses ML and edge-computing architectures for wearable healthcare applications, including monitoring and disease prediction.	Read paper
6	Fusing Wearable Biosensors with Artificial Intelligence for	Ali Kargarandehkordi, Shizhe Li, Kaiying Lin, Kristina T. Phillips,	21 March 2025	Reviews AI analysis of wearable signals and highlights data variability, battery	Read paper

	Mental Health Monitoring: A Systematic Review	Roberto M. Benzo, Peter Washington		consumption, and real-world validation.	
7	AI-Driven Wearable Bioelectronics in Digital Healthcare	Guangqi Huang, Xiaofeng Chen, Caizhi Liao	26 June 2025	Examines AI-enabled wearable bioelectronics for real-time physiological and biochemical monitoring and personalized healthcare.	Read paper
8	Role of Artificial Intelligence in Health Monitoring Using IoT Based Wearable Sensors: A Survey	Laxmi Shaw, Hardik A. Gohel	November 2025	Surveys AI in wearable IoT monitoring, anomaly detection, energy management, communication, privacy, and interoperability.	Read paper
9	AI-Powered Wearable Sensors for Health Monitoring and Clinical Decision Making	Shovito Barua Soumma , Abdullah Mamun , Hassan Ghasemzadeh	December 2025	Reviews AI-enabled biosensors for cardiovascular health, diabetes, mental health, and personalized interventions.	Read paper
10	Wearable Smart Sensors Integration with AI and Machine Learning for Tracking Human Health	Shi Xianyong , Dinesh Kumar Mishra	December 2025	Reviews multiple wearable sensor types and how AI/ML supports signal processing, detection, and personalized monitoring.	Read paper
11	Integrating Artificial Intelligence with Wearable Sensors for Advanced Health Monitoring and Diagnosis	Dongyoun Kim, Syed Saad Ahmed, Amirhossein Amjad, Kwanghee Won, Xiaojun Xian	18 June 2026	Reviews multimodal sensor fusion, AI prediction, explainability, edge deployment, privacy, and clinical validation.	Read paper

Chapter III: SYSTEM ARCHITECHTURE

HealthSync AI is structured around a clean, service-oriented multi-layered model that separates presentation logic, business rules, local data storage, and external device communication. Rather than bundling everything into a single tangled script, the application is organized into distinct logical tiers where each component has a specific, well-defined responsibility.

At the top level, the Presentation Layer is built using Python with PySide6 (Qt for Python) to render a modern, platform-independent desktop interface. This layer contains all user-facing screens—such as the main dashboard, health profile viewer, device scanner, AI analysis window, emergency triggers, hospital locator, and outbreak maps. Crucially, the UI pages do not contain database rules, network protocols, or math formulas; they simply request information and display what the backend services provide.

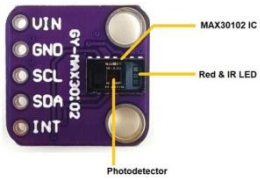
Beneath the interface lies the Application and Service Layer, which acts as the core brain of the platform. This layer contains independent service modules for user authentication, Bluetooth Low Energy (BLE) connectivity, health data validation, analytical calculations, hospital lookup via open mapping APIs, and emergency management. Because these services operate independently of the graphical widgets, the core business logic remains modular and reusable.

For data persistence, the platform uses a Local SQLite Database Layer. SQLite provides a lightweight, serverless relational database that securely stores user credentials, session logs, health profiles, device records, historical vitals, and activity measurements locally on the machine. All database queries enforce strict user-isolation based on authenticated session IDs to ensure personal information never crosses user accounts.

At the physical boundary, the IoT and BLE Layer connects the software to hardware sensors or software simulators. Using Python's asynchronous BLE libraries, the application scans for nearby devices, discovers communication characteristics, streams live telemetry packets, decodes raw hardware feeds, and normalizes them into standard data models before passing them to the database and analysis engines.

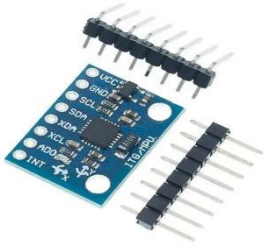
HealthSync AI Band — Circuit Diagram & Component Specifications

The HealthSync AI smart band will use an ESP32 microcontroller, health sensors, motion tracking, a rechargeable battery, and Bluetooth Low Energy (BLE) to collect health and activity data and send it to your HealthSync AI desktop application.



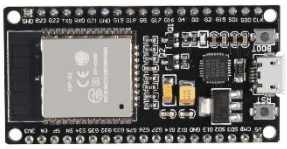
1. Health monitoring

MAX30102 collects pulse and SpO_2 estimates; MAX30205 collects temperature measurements.



2. Activity tracking

MPU6050 readings are processed to estimate steps and movement. Distance and calories are estimates requiring calibration.



3. Data processing and BLE

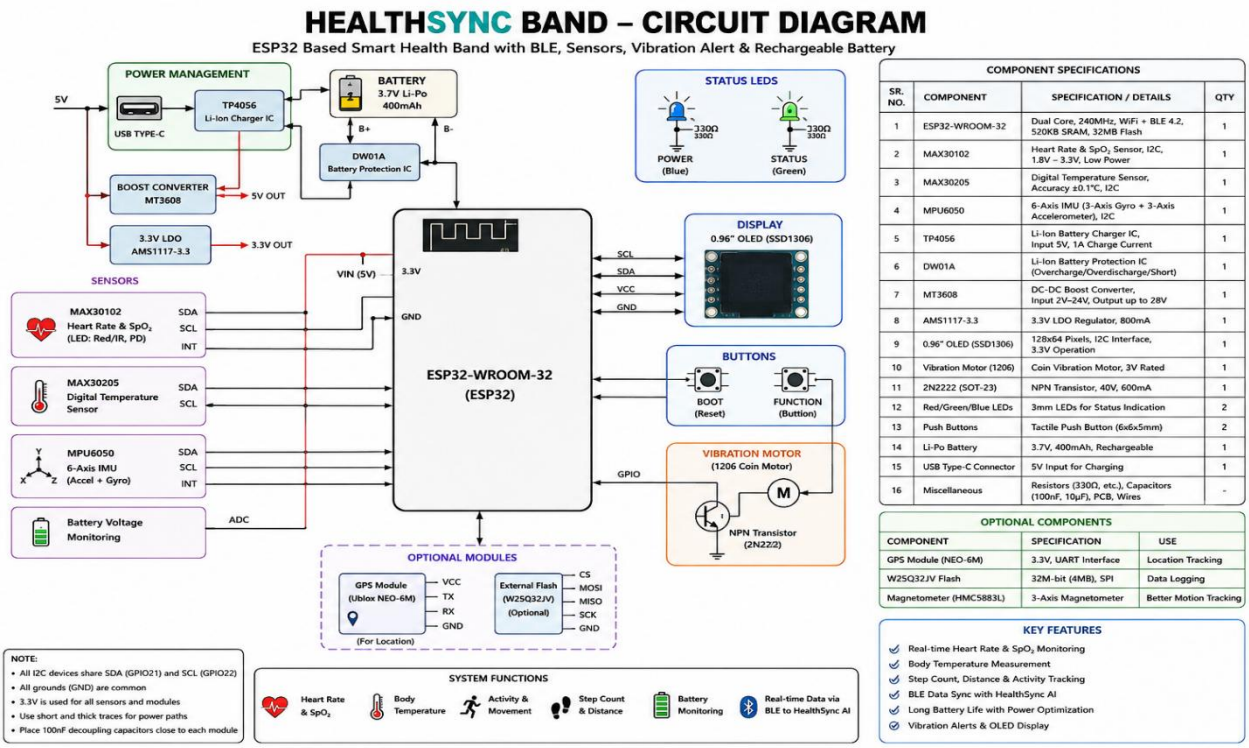
ESP32 reads the sensors and transmits structured readings to the desktop application using your HealthSync-compatible BLE service and telemetry format.



4. Alerts and display

The OLED shows basic information and the vibration motor provides local alerts. HealthSync AI can analyse received data and show observations and recommendations.

Core components: ESP32, MAX30102, MAX30205 and MPU6050.



The circuit diagram is a prototype design reference.

1. Complete component list and specifications

Component	Specification	Purpose
ESP32-WROOM-32 DevKit	Dual-core, up to 240 MHz, 2.4 GHz Wi-Fi, Bluetooth 4.2/BLE, 3.3 V GPIO	Main controller; processes sensor readings and sends data to HealthSync AI
MAX30102	Optical heart-rate and SpO_2 sensor; I^2C; low-power operation	Measures heart rate and estimates blood oxygen saturation
MAX30205	Digital temperature sensor; I^2C; specified accuracy approximately ±0.1°C over its supported measurement range	Measures temperature; skin readings are not equivalent to core body temperature
MPU6050	3-axis accelerometer and 3-axis gyroscope; I^2C	Detects movement, steps, orientation and activity patterns
0.96-inch OLED (SSD1306)	128 × 64 pixels; I^2C; typically 3.3 V-compatible modules	Displays readings, connection status and alerts
Coin vibration motor	Typically 3 V, current depends on the model	Provides a physical alert when the band detects a configured condition

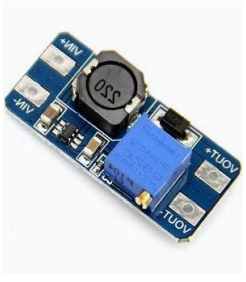
Power supply and supporting components

Component	Recommended specification	Purpose
Li-Po battery	3.7 V nominal, 400–500 mAh, rechargeable	Powers the band
TP4056 charger module	5 V USB input, adjustable charging current; select a protected version	Charges the battery
5 V boost converter	MT3608 module, output adjusted and verified at 5.0 V	Raises battery voltage to power the ESP32 development board through its 5V/VIN input
NPN transistor	2N2222 or BC337, rated for the motor's current	Switches the vibration motor safely
Base resistor	Approximately 1 kΩ, subject to motor current and transistor design	Limits transistor base current
Flyback diode	1N4148 for small motors or a suitably rated alternative	Suppresses motor switching voltage spikes
Battery measurement resistors	Two 100 kΩ resistors for a voltage divider	Scales battery voltage for an ESP32 ADC input
Decoupling capacitors	100 nF ceramic near sensor power pins; additional bulk capacitance as needed	Reduces supply noise
USB-C connector/cable	5 V USB power	Charges the battery through the charger module
PCB, wires and connectors	Compact PCB, short I²C connections	Provides reliable electrical connections



Li-Po battery


TP4056 charger module



5 V boost converter



NPN Transistor


FlyBack Diod

2. Corrected circuit connections

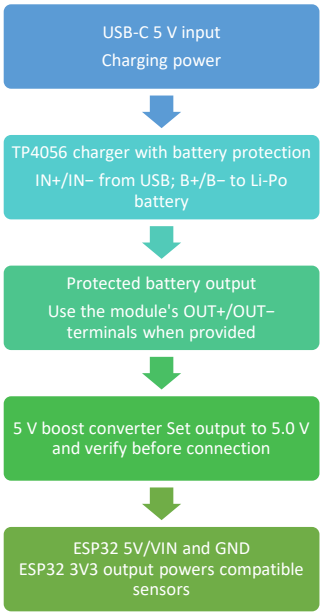
Use these connections as the initial wiring plan for a standard ESP32-WROOM-32 development board.

Sensor connections

Module pin	ESP32 connection
MAX30102 SDA	GPIO 21 (SDA)
MAX30102 SCL	GPIO 22 (SCL)
MAX30102 INT	GPIO 19 (optional)
MAX30205 SDA	GPIO 21 (SDA)
MAX30205 SCL	GPIO 22 (SCL)
MPU6050 SDA	GPIO 21 (SDA)
MPU6050 SCL	GPIO 22 (SCL)
MPU6050 INT	GPIO 18 (optional)
OLED SDA	GPIO 21 (SDA)
OLED SCL	GPIO 22 (SCL)
Sensor and OLED VCC	3.3 V, where supported by the exact breakout
All sensor grounds	Common GND

All four modules can share the I²C bus provided their addresses are compatible and there are no conflicting pull-ups. Check the breakout-board documentation before connecting power.

Battery and power connections



Do not connect the battery directly to the ESP32 3V3 pin. Do not connect the 5 V boost output to a sensor's 3.3 V supply pin. Avoid powering the board simultaneously from USB and an external 5 V source unless the board's power-path design supports it.

Vibration motor circuit

Connect the ESP32 control GPIO to the transistor base through the base resistor. Connect the motor between the appropriate positive supply and the transistor's switched terminal, with the transistor emitter connected to GND. Add a fly back diode across the motor with its cathode toward the positive supply. Verify the transistor pinout and motor current before powering it.

3. Optional components

Component	Specification	When to add it
GPS module, e.g. u-blox NEO-6M	UART; check supply and logic-level requirements	If outdoor location tracking is required
External flash, e.g. W25Q32JV	32 Mbit (4 MB), SPI	If local data logging is needed
Magnetometer, e.g. HMC5883L-compatible module	3-axis magnetic field sensing	If heading or compass information is needed
Battery fuel gauge	MAX17048-compatible module	For more reliable battery-percentage estimates

GPS is not necessary for step counting. The MPU6050 can provide motion data, while distance can be estimated from steps and an individually calibrated stride length. GPS also consumes additional power and may not work well indoors.

4. How the band will work with HealthSync AI

1. Health monitoring: MAX30102 collects pulse and SpO_2 estimates; MAX30205 collects temperature measurements.

2. Activity tracking: MPU6050 readings are processed to estimate steps and movement. Distance and calories are estimates requiring calibration.

3. Data processing and BLE: ESP32 reads the sensors and transmits structured readings to the desktop application using your HealthSync-compatible BLE service and telemetry format.

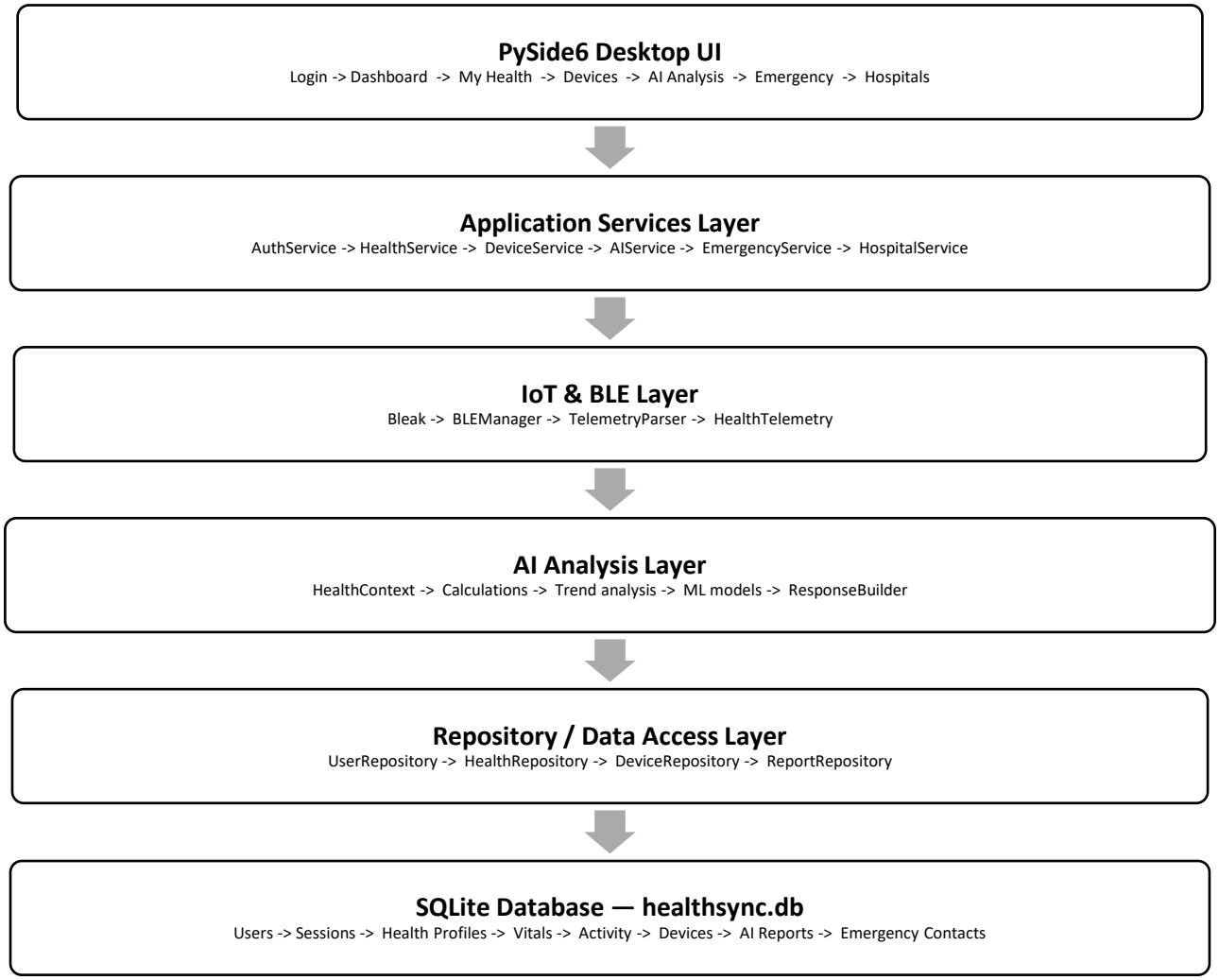
4. Alerts and display: The OLED shows basic information and the vibration motor provides local alerts. HealthSync AI can analyse received data and show observations and recommendations.

Database and Backend Architecture

HealthSync AI's database and backend form the core of the application. The ESP32 band collects health data, the backend validates and processes it, the database stores it securely, and the AI engine analyses it to produce health observations and recommendations.

Since your project uses Python, PySide6, SQLite, BLE and locally developed AI models, we can implement this architecture without paid APIs or a separate database server.

1. Complete backend architecture diagram



The main principle is that UI pages should not execute database queries or implement medical calculations directly. They call services, services use repositories and other domain components, and the data layer communicates with SQLite.

2. Database design and specifications



Database engine: SQLite

Local relational database stored in `healthsync.db`.

SQLite is a good fit for desktop application because it is free, requires no separate database server, supports SQL transactions and foreign keys, and works directly with Python's built-in sqlite3 module.

Database tables

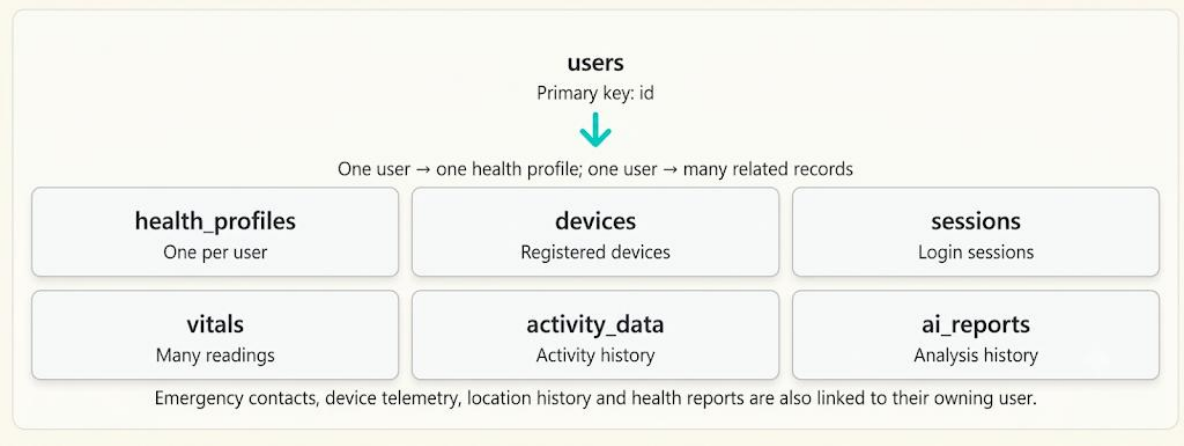
Project architecture includes user accounts, sessions, health profiles, devices, measurements and reports. The following is the recommended logical schema, building on those existing tables rather than replacing your authentication system.

Table	Important columns	Purpose
`users`	`id`, `name`, `email`, `password_hash`, `created_at`	Stores account information and password hashes
`sessions`	`id`, `user_id`, `token_hash`, `created_at`, `expires_at`	Maintains login sessions
`health_profiles`	`id`, `user_id`, `dob`, `gender`, `blood_group`, `height`, `weight`, `allergies`, `conditions`, `medications`	Stores personal health profile
`devices`	`id`, `user_id`, `device_name`, `device_address`, `device_type`, `status`	Tracks registered BLE bands and connection information
`vitals`	`id`, `user_id`, `device_id`, `heart_rate`, `spo2`, `temperature`, `systolic_bp`, `diastolic_bp`, `recorded_at`	Stores health measurements
`activity_data`	`id`, `user_id`, `device_id`, `steps`, `distance`, `calories`, `active_seconds`, `recorded_at`	Stores physical activity data
`device_telemetry`	`id`, `user_id`, `device_id`, `battery_level`, `source`, `received_at`	Records device status and incoming telemetry metadata
`location_history`	`id`, `user_id`, `latitude`, `longitude`, `location_label`, `recorded_at`	Stores location history when location tracking is enabled
`emergency_contacts`	`id`, `user_id`, `name`, `phone`, `relationship`	Stores each user's emergency contacts

`ai_reports`	`id`, `user_id`, `summary`, `observations`, `recommendations`, `created_at`	Stores generated AI health analyses
`health_reports`	`id`, `user_id`, `report_type`, `report_data`, `created_at`	Stores exported or structured health reports
`schema_version`	`version`, `applied_at`	Tracks database schema migrations

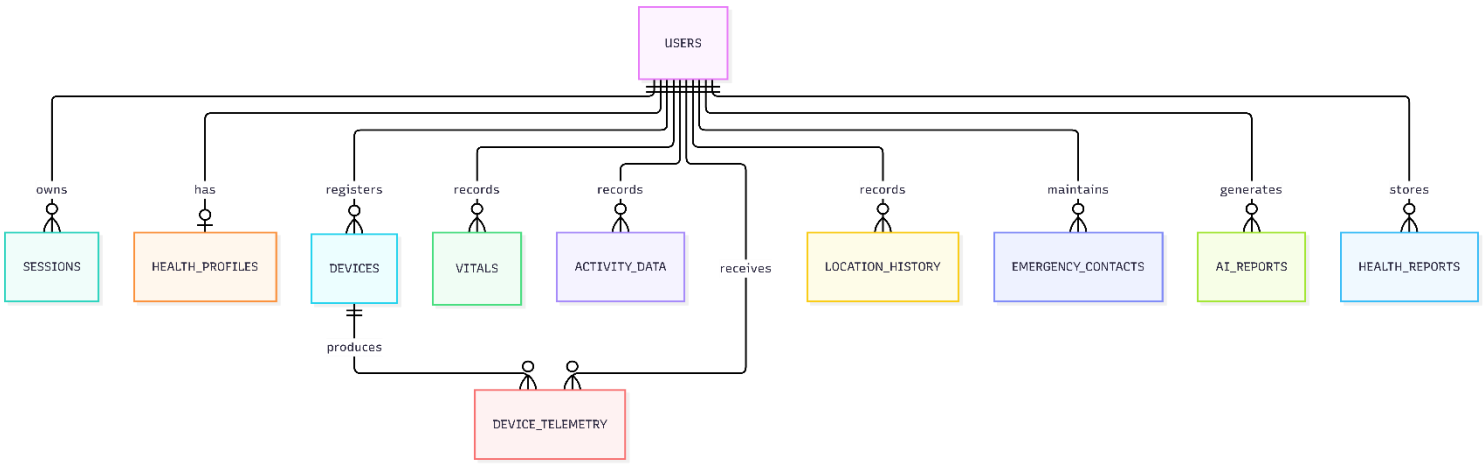
This table lists the target design, not a claim that every column or table is already implemented in your current database. The exact schema should be verified before making changes.

How the tables are connected



relationships use primary keys and foreign keys. For example, `health_profiles.user_id` should reference `users.id`, with a uniqueness constraint if each account is allowed only one profile.

Critical security rule: Every query that retrieves or modifies a user's health information must be scoped to the authenticated `user_id`. Knowing another user's record ID must not grant access to their data.



3. Backend component specifications

The backend is not one large Python file. It is a collection of modules, each responsible for a specific job.

Component	Technology	Responsibility
Application entry point	Python	Initializes the database, services and main window
Authentication service	Python, password hashing	Registration, login, session validation and logout
Database connection layer	`sqlite3`	Opens connections, enables foreign keys and handles transactions
Repository layer	SQL and Python	Performs database CRUD operations
Health service	Python	Saves and retrieves profiles and health measurements
BLE manager	Bleak, `asyncio`	Scans for devices, connects and subscribes to BLE notifications
Telemetry parser	Python	Validates incoming payloads and converts them into typed readings
Device service	Python	Registers devices and manages their status
AI analysis service	Python, NumPy, pandas, scikit-learn as needed	Builds analysis context and coordinates calculations and models
Emergency service	Python	Evaluates configured alert rules and coordinates notifications
Hospital service	HTTP client and OpenStreetMap/Overpass	Retrieves nearby hospital information
Location service	Python	Handles location data and optional location history
Validation and security utilities	Python	Validates inputs, normalizes values and handles secure operations
Report service	Python	Builds reports from stored readings and analysis results

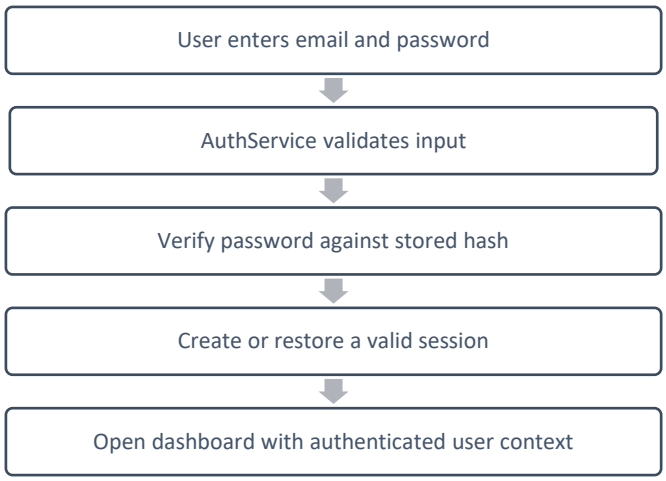
The exact modules should follow the structure already present in your repository. These are responsibilities, not instructions to recreate every module from scratch.

project structure

```
HealthSync-AI/
├── main.py
├── healthsync.db
├── app/
│   ├── database/
│   │   ├── connection.py
│   │   ├── schema.py
│   │   └── repositories/
│   │       ├── user_repository.py
│   │       ├── health_repository.py
│   │       ├── vitals_repository.py
│   │       └── report_repository.py
│   ├── services/
│   │   ├── auth_service.py
│   │   ├── health_service.py
│   │   ├── device_service.py
│   │   ├── ai_service.py
│   │   └── emergency_service.py
│   ├── iot/
│   │   ├── ble_manager.py
│   │   └── telemetry.py
│   ├── ai/
│   │   ├── analysis_engine.py
│   │   ├── models/
│   │   └── response_builder.py
│   ├── utils/
│   │   ├── security.py
│   │   └── validators.py
│   └── ui/
│       ├── main_window.py
│       └── pages/
└── tests/
```

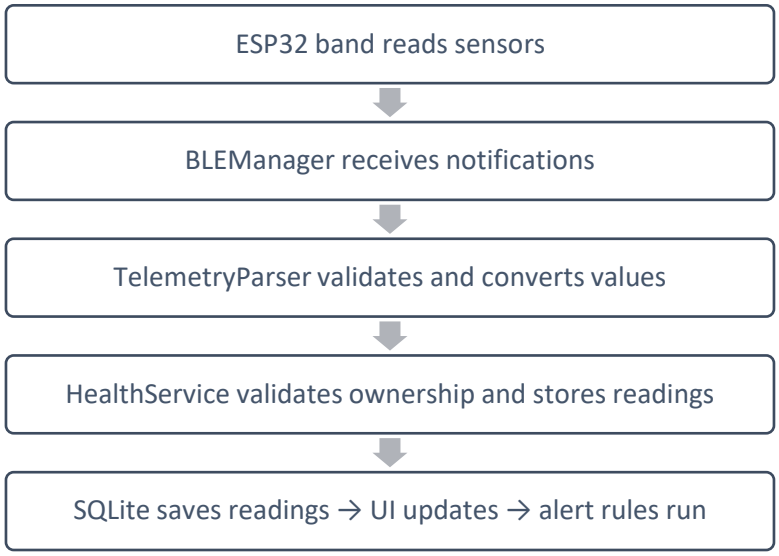
4. How the backend processes data

A. Login and authentication flow



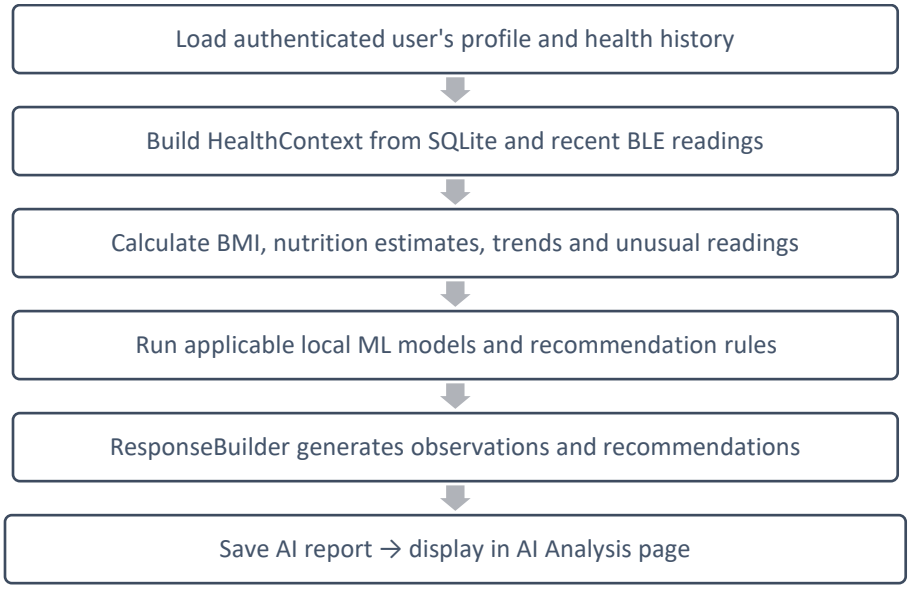
The backend should never store plaintext passwords. It should use a proven password-hashing implementation, such as Argon2id or scrypt, and must verify session expiration and revocation. The authenticated user ID is established by the session—not by trusting a user ID supplied by a UI page.

B. IoT data collection flow



Each reading should have a timestamp, source and associated device. The backend should distinguish missing readings from actual zero values, reject malformed payloads, and handle connection interruptions without freezing the PySide6 interface.

C. AI health analysis flow



This approach fits requirement for a locally developed AI engine instead of relying on paid AI APIs. ML outputs must be interpreted cautiously: a risk-model probability is not a diagnosis, and sensor readings may contain noise or inaccuracies.

Backend and database configuration

Setting	Recommendation
Language	Python 3.14, subject to compatibility testing for each dependency
Desktop UI	PySide6
Database	SQLite
Database file	`healthsync.db` in the project root
Database access	Python `sqlite3`
Foreign keys	Enable `PRAGMA foreign_keys = ON` for every connection
Transactions	Commit successful operations; rollback failed operations
Connections	Use short-lived connections with reliable cleanup
Schema management	Versioned migrations; preserve existing data
BLE	Bleak with asynchronous device communication
AI execution	Local Python calculations and compatible saved models
External hospital data	Free OpenStreetMap/Overpass service, with timeouts and error handling

A separate REST API server is not required for current single-desktop-app architecture. The desktop UI can call the Python services directly. An API can be introduced later if you want multiple clients or remote access.

6. Security requirements

- Authentication: Store salted password hashes, never raw passwords.
- Data isolation: Scope all health, device, contact and report operations to the authenticated account.
- SQL safety: Use parameterized queries instead of string concatenation.
- Data integrity: Use foreign keys, constraints and transactions.
- BLE validation: Do not trust every packet merely because it arrives over Bluetooth; validate payloads and device identity.
- Privacy: Collect location only when required and clearly disclose its use.
- Error handling: Do not expose password hashes, session secrets or sensitive health information in logs.
- Database protection: Restrict access to the database file and consider encrypted storage or an appropriate encrypted database solution if stronger at-rest protection is required.
- Medical safety: Treat emergency thresholds as configurable screening rules, not definitive medical conclusions.

7. Testing checklist

Before integrating the band and AI engine, we test the backend in this order:

- [ ] Registration, login, logout and session restoration.
- [ ] Creating and updating a health profile.
- [ ] Ensuring one account cannot access another account's records.
- [ ] Saving and retrieving vitals and activity measurements.
- [ ] Handling invalid, missing and out-of-range telemetry.
- [ ] Disconnecting and reconnecting the BLE device.
- [ ] Generating and retrieving AI analysis reports.
- [ ] Triggering configured emergency rules and testing failure cases.
- [ ] Handling unavailable hospital services without crashing the UI.
- [ ] Preserving existing data during database schema upgrades.

Development order: first verify the current database schema and authentication/session flow, then verify repositories and health services, then connect BLE telemetry persistence, and finally integrate the AI engine and emergency workflow. This minimizes the risk of breaking existing HealthSync AI functionality.

\

Chapter IV: IMPLEMENTATIONS

The implementation of HealthSync AI (HSAI) focuses on bringing together health monitoring, artificial intelligence, IoT communication, and secure data management within a single desktop application. The system is designed to help users monitor their health information, review changes in their readings, and receive AI-assisted health insights.

The development follows a modular approach, in which each major function is handled by a separate component. This makes the application easier to maintain, test, and improve without requiring changes to the entire system.

The implementation covers system design, dataset preparation, machine learning model development, integration of software modules, and testing. The desktop interface is developed using Python and PySide6, while SQLite is used for local data storage. Bluetooth Low Energy (BLE) provides the communication mechanism for compatible wearable devices.

System Design and Use Case Diagrams

System design defines the interactions between the user and the different features of HealthSync AI. Before implementing individual modules, the major functions of the application are identified to establish a clear development structure.

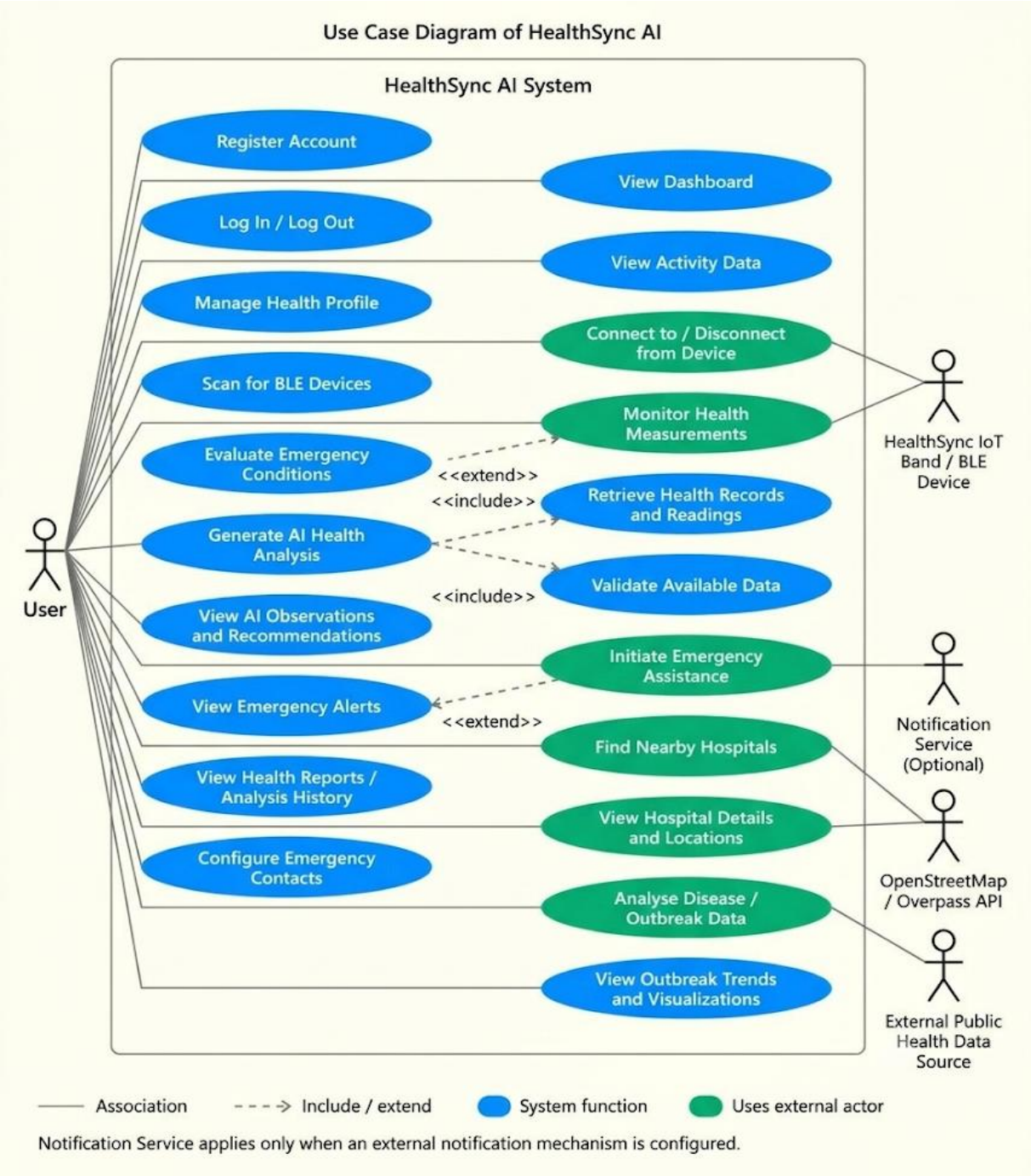
The primary actor is the user, who accesses the application through the desktop interface. After authentication, the user can manage personal health information, connect a compatible device, view collected readings, generate AI-assisted analysis, and access emergency and hospital-related features.

The main use cases of HealthSync AI include:

- **User authentication:** Allows users to register, log in, and access their individual accounts.
- **Health profile management:** Enables users to maintain personal details, medical conditions, allergies, medications, and other relevant information.
- **Device management:** Allows users to scan for compatible Bluetooth devices and initiate a connection.
- **Health monitoring:** Displays available physiological measurements and activity information received from connected devices.
- **AI health analysis:** Examines available health records and readings to generate observations and recommendations.
- **Emergency assistance:** Provides the interface for emergency-related actions based on configured conditions and contact information.
- **Hospital lookup:** Helps users locate nearby hospitals using available geographical data services.
- **Outbreak analysis:** Provides a dedicated module for examining relevant disease or outbreak information when suitable data sources are available.

The use case diagram illustrates these interactions and defines the functional boundaries of the application. It also helps identify which components require authentication, access to health records, device connectivity, or external data services.

Use Case Diagram of HealthSync AI



Dataset Collection and Preprocessing

Machine learning models require suitable datasets to identify patterns in health-related information. For HealthSync AI, the selection of a dataset depends on the purpose of the model, the availability of relevant features, and the quality of the recorded information.

One of the datasets explored during development is the CDC Diabetes Health Indicators dataset. It contains 253,680 records and 21 features, including the binary diabetes target used for classification.

The dataset contains health indicators and demographic or lifestyle-related information that can be used to study patterns associated with diabetes. It provides a basis for developing a screening-oriented risk estimation component within the broader AI analysis system.

Data preprocessing

Before model training, the dataset must be examined to understand its structure, feature types, target distribution, and data quality. The preprocessing workflow is designed to ensure that the input data is suitable for machine learning.

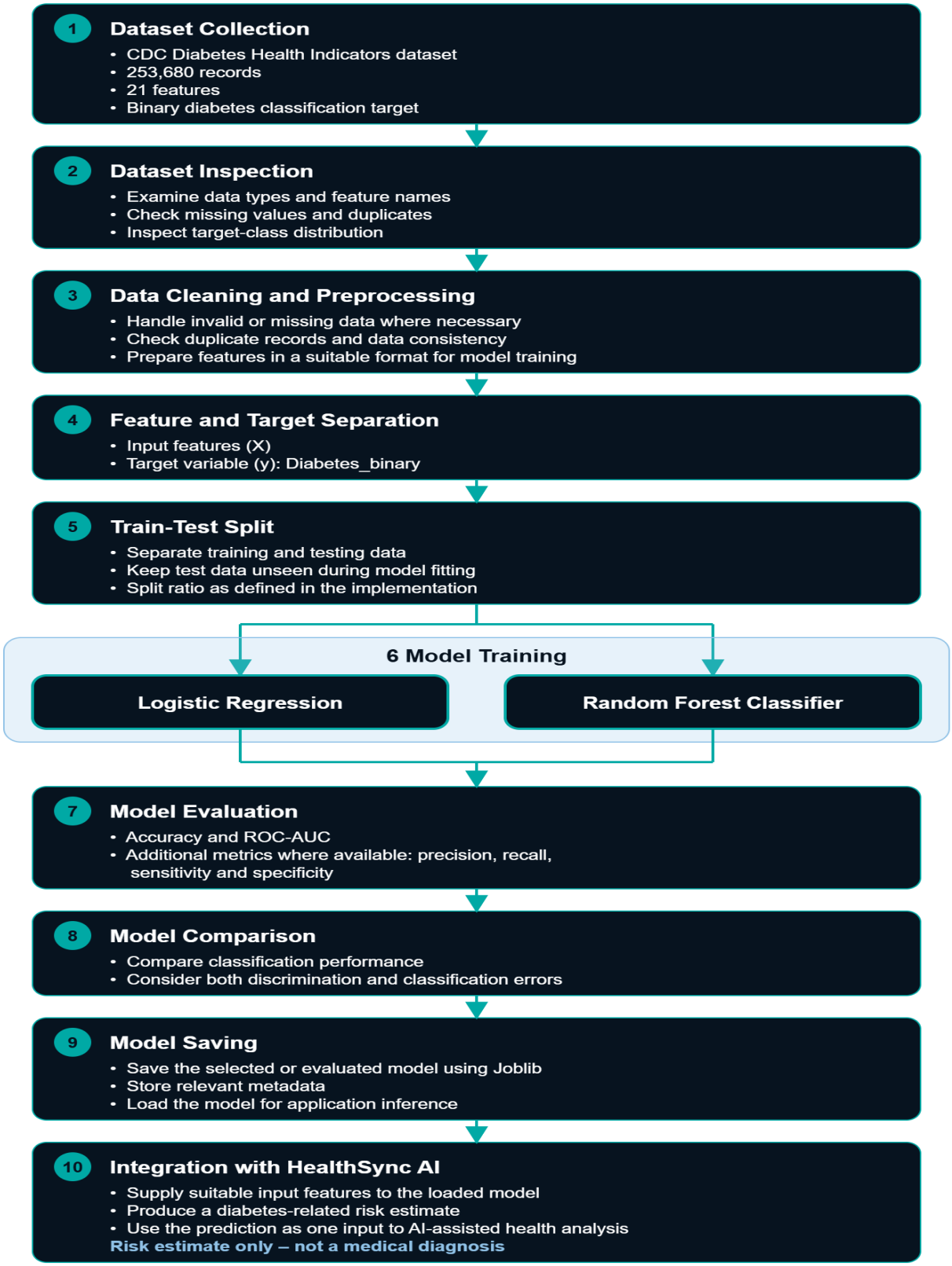
The general stages include:

1. **Data inspection:** Examine the dataset dimensions, feature names, data types, and target variable.
2. **Data quality checks:** Identify missing values, invalid entries, duplicates, and potential inconsistencies.
3. **Feature selection:** Retain features relevant to the intended prediction task.
4. **Target preparation:** Separate the diabetes classification target from the input features.
5. **Dataset splitting:** Divide the data into training and testing subsets so that model performance can be evaluated on records not used for fitting.
6. **Model preparation:** Apply the preprocessing required by each algorithm and maintain consistency between training and inference.

The exact preprocessing operations, split ratio, and random seed should be documented according to the actual training script. These details are important because different preprocessing choices can affect model performance.

The trained model is intended to support the analysis of relevant health information. Its output should not be treated as a confirmed diagnosis, and predictions should be interpreted within the limitations of the dataset and model.

Dataset Preparation Workflow



AI/ML Models and Training Methodology

Artificial intelligence is one of the central components of HealthSync AI. Rather than presenting health conditions as a simple list of disease probabilities, the intended analysis system combines available measurements, personal health information, activity records, and historical trends to produce a readable summary of observations and recommendations.

The AI implementation distinguishes between conventional machine learning predictions and health analysis based on recorded measurements. These functions serve different purposes and should not be confused with one another.

1. Logistic Regression

Logistic Regression was evaluated as a baseline classification algorithm for the diabetes prediction task. It estimates the probability of a binary outcome using a mathematical relationship between the input features and the target class.

One advantage of this algorithm is that it provides a relatively simple baseline against which more complex models can be compared. Its performance also helps establish whether the selected features contain useful predictive information.

During training, the model learns its coefficients from the training dataset. The resulting model is then evaluated using the held-out test data to estimate how well it performs on previously unseen records.

2. Random Forest

Random Forest was also evaluated for diabetes classification. It combines the predictions of multiple decision trees rather than relying on a single tree.

This approach can capture nonlinear relationships and interactions between input features. It was included in the experiments to determine whether an ensemble-based method could improve classification performance compared with the baseline model.

The trained Random Forest model was evaluated using the same general classification task. Comparing the two algorithms provides a more informed basis for selecting a model than relying on a single performance measure.

3. Model evaluation

The following results were recorded during the model experiments:

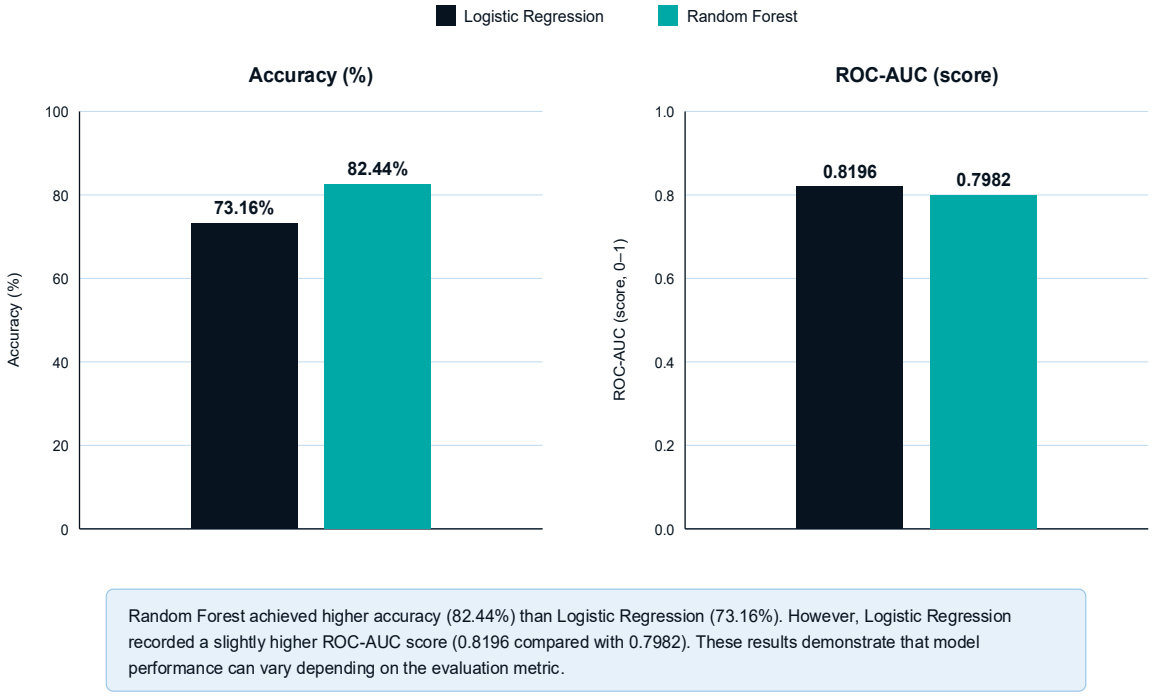
Evaluation metric	Logistic Regression	Random Forest
Accuracy	73.16%	82.44%
ROC-AUC	0.8196	0.7982

The Random Forest experiment achieved higher accuracy, whereas Logistic Regression achieved a higher ROC-AUC score. This difference illustrates why model selection should not depend on accuracy alone.

Accuracy measures the proportion of correctly classified records, while ROC-AUC evaluates how effectively a model distinguishes between the two classes across different classification thresholds. For a health-related prediction task, sensitivity, specificity, precision, recall, calibration, and the consequences of false-positive and false-negative predictions should also be considered.

The reported values describe the recorded experimental results and do not establish the clinical reliability of either model. Further validation would be necessary before using such predictions in real-world healthcare decisions.

Figure 5.3: Comparison of Logistic Regression and Random Forest



Random Forest achieved higher accuracy (82.44%) than Logistic Regression (73.16%). However, Logistic Regression recorded a slightly higher ROC-AUC score (0.8196 compared with 0.7982). These results demonstrate that model performance can vary depending on the evaluation metric.

5.4.4 Model storage and application integration

Following training, the model artifacts were saved for later use. The diabetes model was stored as a Joblib file, accompanied by a metadata file containing relevant model information.

Separating model training from application execution avoids the need to train the classifier every time the desktop application starts. The application can instead load the saved model when the corresponding analysis function is invoked.

The broader AI Health Analysis Engine is designed to combine several types of information, including:

- Current health measurements received from compatible devices.
- Personal health information maintained by the user.
- Historical vital readings and activity records.

- Changes and unusual patterns observed over time.
- Relevant model predictions, when the required inputs are available.
- Nutrition-related calculations and general health recommendations.

These inputs can be organized into a structured health context before the response is prepared. The resulting output should explain important observations in understandable language, identify missing information when necessary, and provide appropriate general recommendations.

The diabetes classifier is one component of this broader design; its presence does not mean that every health condition is supported by a trained predictive model.

IoT and BLE Implementation

The IoT component of HealthSync AI is designed to collect health and activity information from a compatible wearable device and transfer it to the desktop application. The proposed hardware platform uses an ESP32 microcontroller with suitable physiological and movement sensors.

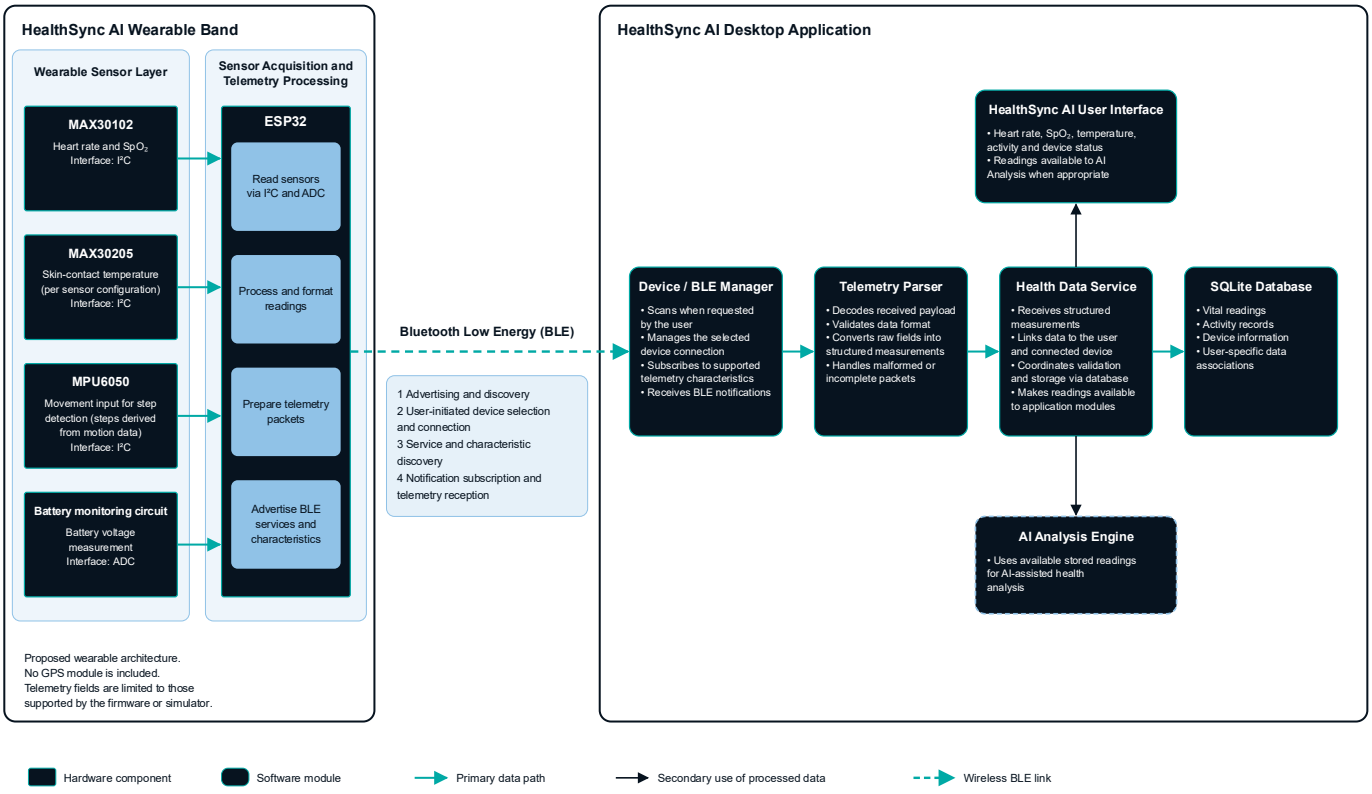
Bluetooth Low Energy provides the communication link between the wearable device and the application. On the software side, device discovery, connection management, telemetry parsing, and health-data processing are separated into individual components.

The device workflow begins when the user initiates a scan from the Devices page. After selecting a compatible device, the application can establish a connection and subscribe to the supported BLE characteristics. Incoming data is interpreted and converted into structured measurements before being passed to the health-data processing layer.

The same software architecture can be exercised using the HealthSync BLE simulator during development. Simulator-based verification is useful for checking the communication and parsing workflow before relying on physical hardware.

The implementation status of the physical ESP32 band should be reported separately from simulator testing. Sensor accuracy, battery performance, and end-to-end operation require validation with the actual hardware.

IoT Data Acquisition and BLE Communication



5.6 Database and Backend Implementation

HealthSync AI uses SQLite for local storage and Python services for application logic. The database maintains user accounts, sessions, health profiles, device information, vital readings, activity records, emergency contacts, and AI-generated reports.

The backend separates data access from the user interface. This allows individual modules to retrieve or update information through defined services rather than implementing database operations directly within every screen.

User-specific health information is associated with the authenticated user's identifier. Database relationships and access checks are intended to prevent one user from accessing another user's personal records.

The existing database architecture and its relationships are explained in the earlier database design section. This subsection therefore focuses on how the database supports the application rather than repeating the full schema.

System Integration and Data Flow

System integration connects the individual modules so that information can move through HealthSync AI in a consistent manner. Each module has a defined responsibility, while the service layer coordinates communication between components.

The main data flow begins with information supplied by the user or received from a connected wearable device. The incoming readings are parsed and passed to the health-data service, which can store the measurements in the database. When an analysis is requested, the AI engine retrieves the relevant available records and prepares a structured representation of the user's health information.

The analysis stage examines available readings and historical patterns. Where a trained model is applicable and the necessary inputs are present, its prediction can contribute to the analysis. Other calculations and observations can be produced directly from the available measurements.

The response-building component then organizes the findings into a readable summary, observations, recommendations, and any relevant limitations. The desktop interface displays the resulting information to the user.

The overall integration workflow can be represented as follows:

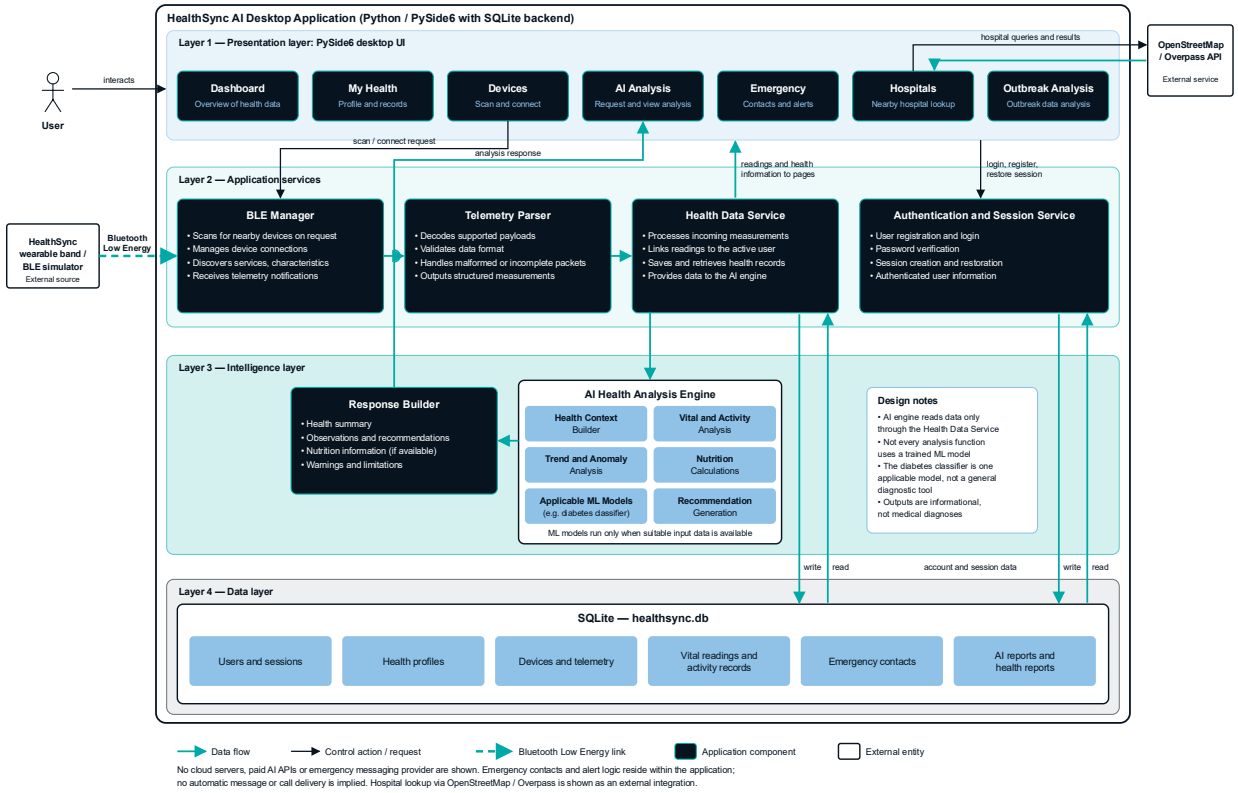
Data input → Processing → Storage → Analysis → Response generation → User interface

The principal components involved in this process are:

1. **Input layer:** Receives information from the user, BLE devices, and supported external data sources.
2. **Communication and processing layer:** Handles device connections, parses incoming telemetry, and validates available measurements.
3. **Data layer:** Stores and retrieves information associated with the authenticated user.
4. **Analysis layer:** Performs calculations, examines trends, and invokes applicable trained models.
5. **Response layer:** Converts structured results into understandable observations and recommendations.
6. **Presentation layer:** Displays the results through the relevant desktop application pages.

This modular structure allows individual components to be modified or tested without redesigning the entire application. It also provides a basis for future improvements, such as additional sensor types, more analysis methods, and alternative data providers.

Overall HealthSync AI System Architecture and Data Flow



The diagram must represent a Python and PySide6 desktop application with a modular backend, rather than a web-based application.

Components to include

A. Desktop User Interface — Python / PySide6

Show the following application pages within the presentation layer:

- Dashboard
- My Health
- Devices
- AI Analysis
- Emergency
- Hospitals
- Outbreak Analysis

The interface allows users to manage health information, initiate device scans, request analysis, and view available results.

B. Authentication and Session Service

Show a dedicated authentication component responsible for:

- User registration and login
- Password verification
- Session creation and restoration
- Access to the authenticated user's information

Connect this component to the SQLite database for account and session operations.

C. BLE Manager

Represent the component responsible for:

- Scanning for nearby BLE devices when requested
- Managing device connections
- Discovering supported services and characteristics
- Receiving telemetry notifications

Show a connection to the external HealthSync wearable band or BLE simulator. Label the connection “Bluetooth Low Energy.”

D. Telemetry Parser

Show this component receiving data from the BLE Manager and converting supported payloads into structured health measurements.

Include data-format validation and handling of malformed or incomplete packets.

E. Health Data Service

Represent the service responsible for coordinating health-data operations, including:

- Processing incoming measurements
- Associating readings with the appropriate authenticated user
- Saving and retrieving health records
- Providing relevant data to the AI analysis component

F. SQLite Database

Show a single database component labelled “SQLite — healthsync.db”.

Include the major categories of stored information:

- Users and sessions
- Health profiles
- Devices and telemetry

- Vital readings and activity records
- Emergency contacts
- AI reports and health reports

Use directional connectors to indicate which services read from or write to the database. Do not connect every UI page directly to the database.

G. AI Health Analysis Engine

Show a modular AI analysis component containing these logical functions:

- Health Context Builder
- Vital and Activity Analysis
- Trend and Anomaly Analysis
- Nutrition Calculations
- Applicable Machine Learning Models
- Recommendation Generation

The engine receives relevant information from the Health Data Service and invokes trained models only when suitable input data is available.

Show the diabetes classification model as one applicable model, not as a model that diagnoses every health condition.

H. Response Builder

Represent the component that organizes the analysis results into a structured response containing:

- Health summary
- Observations
- Recommendations
- Nutrition information, when available
- Relevant warnings and limitations

Connect the Response Builder to the AI Analysis Engine and then to the AI Analysis page in the desktop interface.

Data-flow connections

Use directional arrows to illustrate these relationships:

1. User → Desktop UI → Authentication and Session Service.
2. Authentication Service ↔ SQLite Database for account and session operations.
3. Desktop UI → BLE Manager when the user initiates a device scan or connection.
4. Wearable Band / BLE Simulator → BLE Manager through Bluetooth Low Energy.
5. BLE Manager → Telemetry Parser → Health Data Service.
6. Health Data Service ↔ SQLite Database for supported health-record storage and retrieval.
7. Health Data Service → AI Health Analysis Engine.
8. AI Health Analysis Engine → Response Builder.
9. Response Builder → AI Analysis page in the Desktop UI.
10. Health Data Service → relevant UI pages for available readings and health information.

Diagram layout

Layer 1 — Presentation Layer

- PySide6 Desktop UI
- Application pages

Layer 2 — Application Services

- Authentication and Session Service
- BLE Manager
- Telemetry Parser
- Health Data Service

Layer 3 — Intelligence Layer

- AI Health Analysis Engine
- Health Context Builder
- Applicable ML Models
- Response Builder

Layer 4 — Data Layer

- SQLite Database
- User records, health data, device readings, and reports

Testing and Performance Evaluation

Testing is required to verify that the different modules behave as intended and that information is processed consistently as it moves through the system. HealthSync AI requires both functional testing of its application features and evaluation of its machine learning experiments.

The testing process should cover authentication, health-record management, device communication, database operations, AI analysis, and the behavior of emergency-related features.

Functional testing

Functional testing checks whether a feature produces the expected result for a given input or action.

The following test cases can be used to document the verification process:

Test case	Verification objective	Expected result
User authentication	Test valid and invalid login details	Access is granted or rejected appropriately
Health profile	Add and retrieve personal health information	The correct user's information is displayed
Device scanning	Initiate a BLE scan	Available devices are listed when discoverable
Device connection	Connect to a compatible device	The connection state is reported correctly
Telemetry processing	Supply a valid supported reading	The reading is parsed into the appropriate fields
Data persistence	Save and retrieve a health record	The saved record can be retrieved correctly
AI analysis	Request analysis with sufficient input data	The analysis produces a structured result
Missing input	Request analysis when information is incomplete	Missing information is handled appropriately
Emergency condition	Simulate a configured alert condition	The expected alert workflow is triggered, if implemented
Hospital lookup	Request hospital information	Results are displayed when the external service returns suitable data

Machine learning evaluation

The classification models were evaluated using accuracy and ROC-AUC, as described earlier Additional evaluation should examine class-wise precision and recall, sensitivity, specificity, and the effect of the selected decision threshold.

It is also important to check for data leakage and ensure that the test records remain separate from the data used to fit the models. If the model is later used to analyze real users' health information, its limitations and suitability for that population must be examined.

Integration testing

Integration testing examines whether connected modules work together correctly. For example, a reading received from a BLE device should be parsed correctly, stored under the appropriate user, and made available to the analysis engine when requested.

Similarly, the application should handle situations such as a disconnected device, missing readings, unavailable external services, or incomplete health profiles without presenting fabricated results.

The findings from these tests should be recorded alongside the software version, test conditions, and any corrections made. Quantitative performance claims should be included only when supported by actual measurements.

Chapter V: RESULT ANALYSIS

HealthSync AI is being developed as a Python desktop application using PySide6, with SQLite for local data storage and Bluetooth Low Energy (BLE) as the communication approach for compatible devices. The project also includes a machine-learning experiment using the CDC-derived Diabetes Health Indicators dataset. Some functions have been tested with software-generated or simulator data, while the planned ESP32 wearable is a later hardware phase. This distinction is important: simulator results demonstrate the software data path, but they do not establish the accuracy of a physical sensor or a medical device.

Summary of Implementation Results

The project brings together several functions that are often handled by separate tools. The desktop interface provides navigation between the dashboard, personal health information, device management, AI analysis, emergency support, hospital lookup, and outbreak analysis. The modular structure is intended to keep the interface separate from authentication, database operations, BLE communication, telemetry parsing, and analysis logic. This makes it easier to test individual services and replace a data provider without redesigning the whole application.

The authentication and storage design uses a local SQLite database. User records and health-related information are intended to be associated with the authenticated user rather than shared across all accounts. Passwords should be stored as hashes, not as readable text, and database connections enable foreign-key enforcement. These are useful foundations for a privacy-aware desktop application, although they do not replace a full security audit.

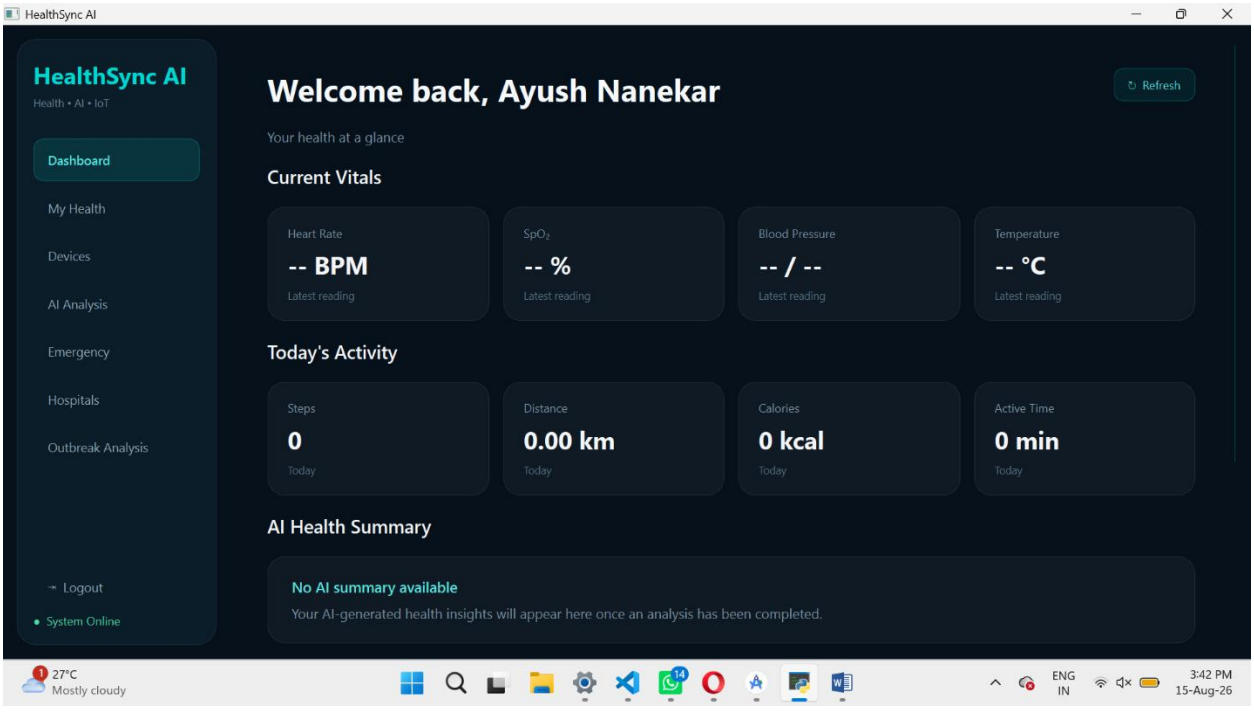
Area	Result demonstrated	Evidence to include
Desktop interface	Navigation and page layout for the project's main modules	Screenshots of the dashboard and each completed page
Authentication and database	Local user/session handling and database structure	Login/register screen and database test output
BLE/telemetry workflow	Data-model and parser workflow tested with simulator or test payloads	Device page, connection state, and sample telemetry
AI/ML experiment	Two trained diabetes-risk classifiers evaluated on a dataset	Metrics table, ROC curve or evaluation output
Hospital lookup	OpenStreetMap/Overpass lookup returned hospital results during a connectivity test	Search results with test date and location

User Interface Results and Screenshot Analysis

The user interface is an important part of HealthSync AI because health information is useful only when a person can find and interpret it without unnecessary effort. The design uses a dark navy background with cyan/teal accents and cards that visually separate information. The same visual language across pages helps the application feel like one system instead of a collection of unrelated screens. The analysis below should be matched to the current build: if a page is still a prototype, describe it as a prototype rather than as a fully operational feature.

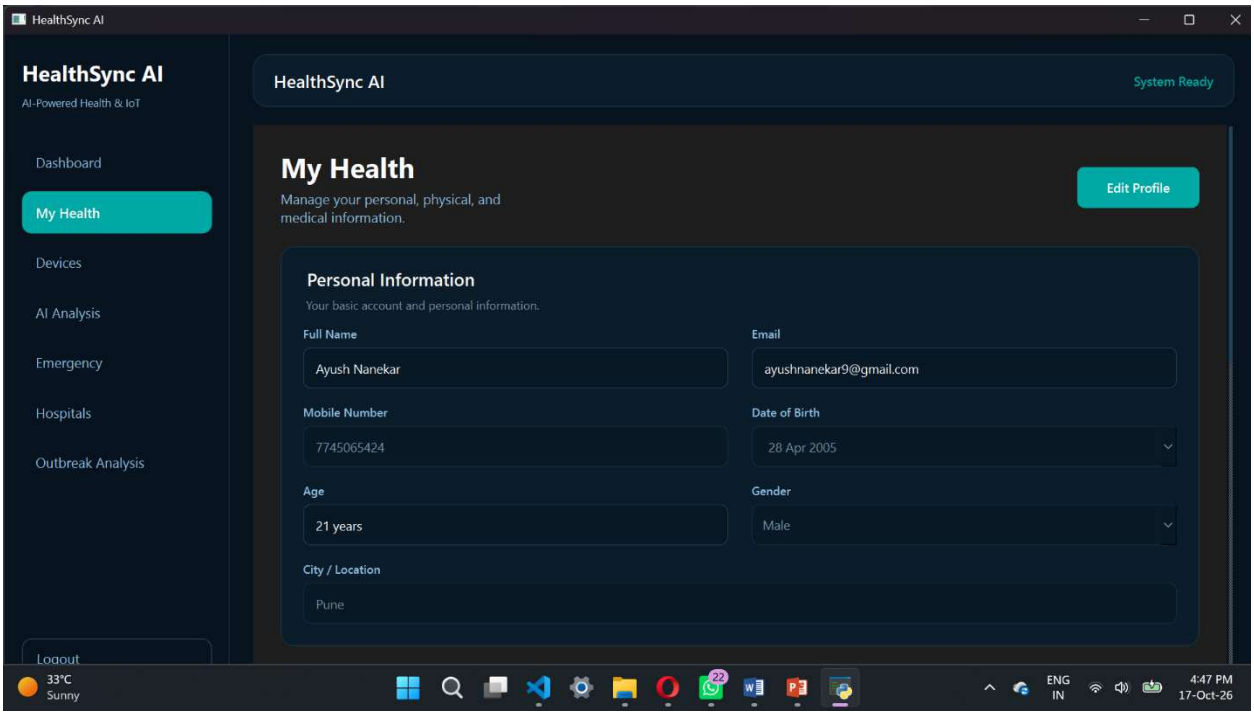
5.3.1 Dashboard

The dashboard is intended to provide a quick entry point to the user's health information and the other modules. A dashboard is most effective when it gives a concise overview and allows the user to move to detailed information with a small number of actions. During evaluation, check whether labels are understandable, important values are easy to distinguish, and missing data is shown as unavailable rather than replaced with invented readings. If the dashboard currently displays sample values, label them clearly as demonstration data.



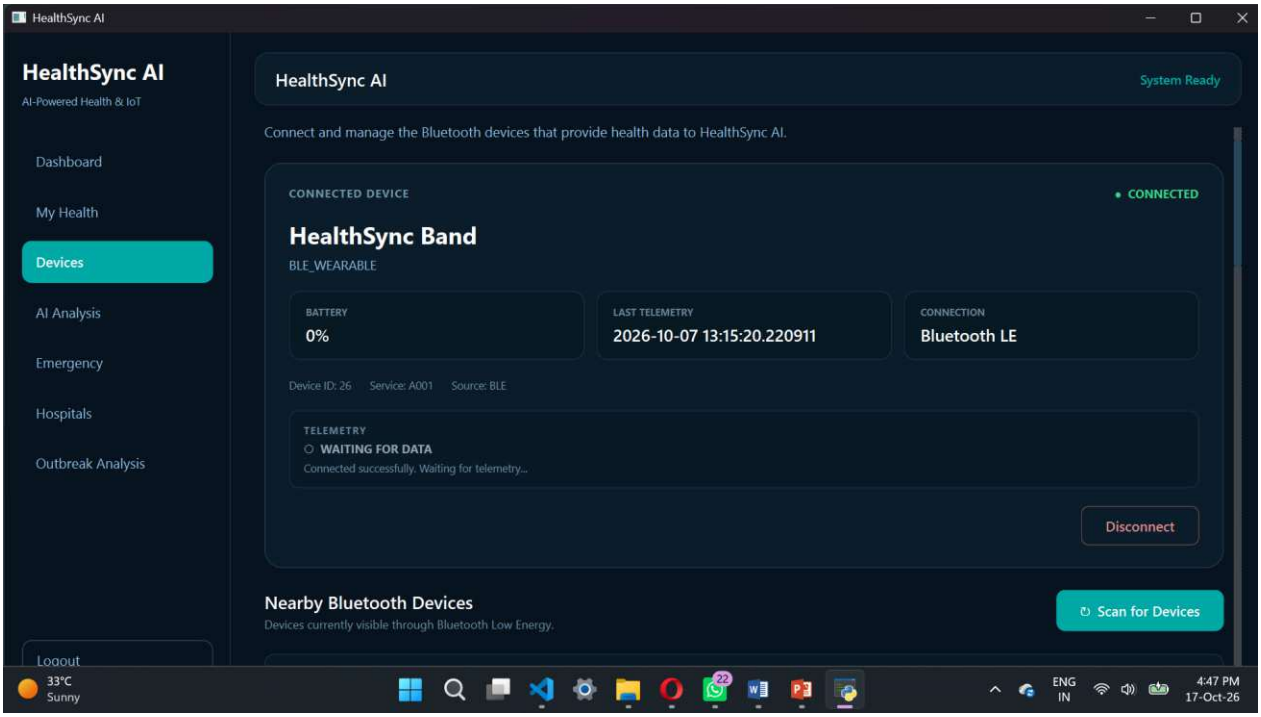
5.3.2 My Health

The My Health page is designed to hold profile details such as age, date of birth, gender, blood group, height, weight, medical conditions, allergies, and medications. Keeping these details in a structured profile can provide context for later analysis and reduce the need to re-enter information. The page should make it clear which fields are optional, validate basic input formats, and avoid presenting user-entered information as medically verified. The report should include a screenshot showing the fields that are present in the current implementation.



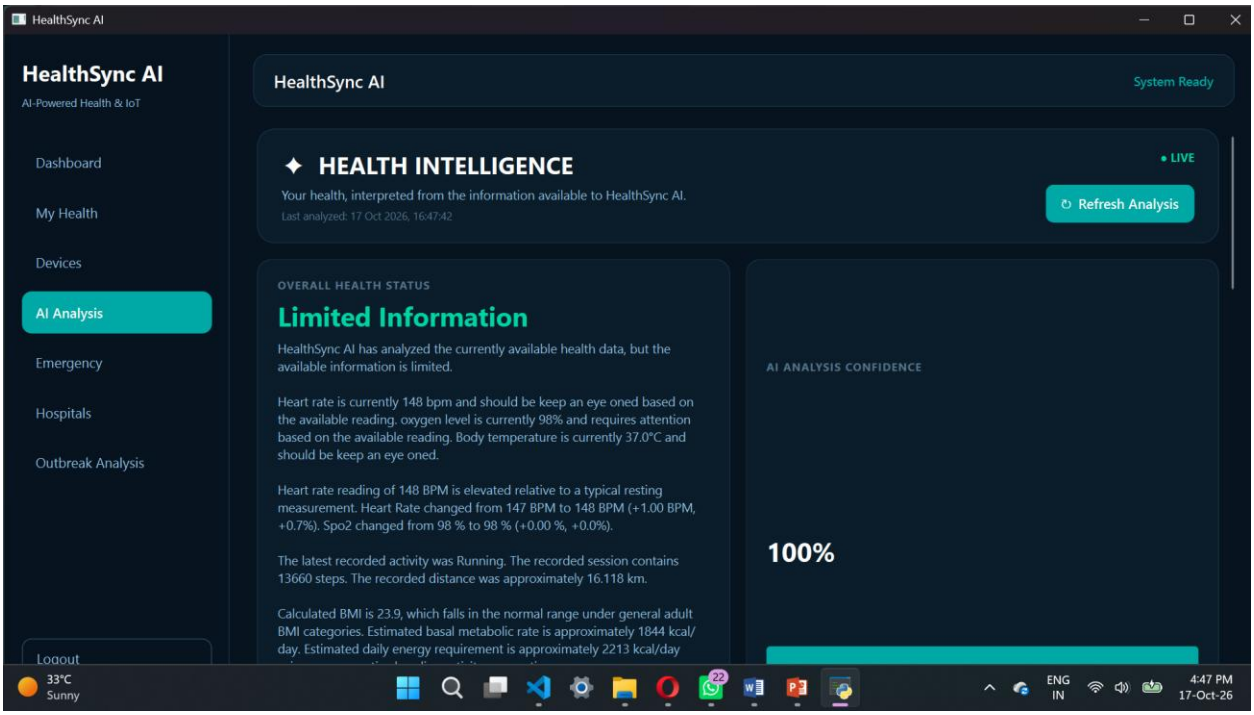
5.3.3 Devices and Health Readings

The Devices page is intended to let the user initiate a scan and choose a compatible BLE device, rather than automatically connecting without user action. A clear connection status is important because an empty or disconnected device should not be mistaken for a live health reading. The screen should distinguish values received from a device or simulator from fields that have not been supplied. If the Android simulator is used, the screenshot caption should identify it as a simulator and not as a physical ESP32 wearable.



5.3.4 AI Analysis, Emergency Support, and Hospital Lookup

The AI Analysis page is intended to turn available profile, vital-sign, activity, and historical information into a readable summary with observations and practical recommendations. The result should explain the basis of the output and include a reminder that it is informational screening support, not a diagnosis. The Emergency page and hospital lookup page should also be evaluated for clarity: users should be able to identify the available action, understand whether a message or call is actually supported, and see when internet access is required for external search results.



For each screenshot, use a figure number and a short caption, for example: “Figure 5.1: Dashboard view of HealthSync AI.” Follow the figure with two or three sentences describing what the screenshot demonstrates. Do not include screenshots without analysis, and do not claim a button or workflow works unless it was tested.

5.4 Functional Testing and Validation

Functional testing checks whether each component behaves as expected for normal inputs, missing information, and common error cases. It is especially important in a health-related application because a display error, account mix-up, or misleading connection status could cause a user to misunderstand the information shown. The following table is a suggested test record. Keep the status column aligned with the tests you have actually run; change any "Pass when verified" entry to Pass or Fail only after repeating the test on the current build.

Test ID	Test case	Expected behaviour	Status
T01	Launch application	Main window opens without an unhandled exception	✔
T02	Register with valid details	A user record is created and password is not stored as plain text	✔
T03	Login with valid credentials	Correct account is authenticated	✔
T04	Login with invalid credentials	Access is denied and a clear error is shown	✔
T05	Restore or expire a session	Session behaviour follows the configured policy	✔
T06	Save and reopen health profile	Saved fields remain associated with the same user	✔
T07	Scan for BLE devices	Nearby compatible devices or a clear empty result are shown	✔
T08	Connect/disconnect a test device	Connection state updates and disconnect is handled safely	✔
T9	Handle missing or malformed telemetry	Missing values remain unavailable; malformed data does not crash the UI	✗
T10	Generate AI analysis	Available data is summarized and limitations are disclosed	✔
T11	Search hospitals online	Results are shown when the external service responds; failure is handled gracefully	✗

A test should include the input, the expected output, the observed output, and the final status. A screenshot of a successful screen is useful evidence, but it does not replace testing invalid inputs or failure conditions. For database isolation, create two test accounts and confirm that one account cannot view or change the other account's profile or health records. For BLE, repeat the test with the device unavailable and with incomplete data. For hospital lookup, test both a successful internet connection and a failed request.

The testing results should be summarized only after the current version has been exercised. If a feature has been designed but not tested, classify it as "Not tested" or "Planned" rather than "Pass." This reporting practice makes the results credible and helps identify the next development tasks.

5.5 AI Model Results and Interpretation

A machine-learning experiment was carried out using the CDC-derived Diabetes Health Indicators dataset, which contains 253,680 survey records and 21 input features in the selected binary classification version. The experiment compares logistic regression with a random forest classifier. The purpose is to evaluate whether health-indicator patterns can be used to estimate a diabetes-related risk category in a research dataset. It is not evidence that the model can diagnose diabetes in an individual or that it has been clinically validated.

Model	Accuracy	ROC-AUC	Interpretation
Logistic Regression	0.7316 (73.16%)	0.8196	Lower accuracy than the random forest in this experiment, but higher ROC-AUC; useful as a simpler baseline.
Random Forest	0.8244 (82.44%)	0.7982	Higher observed accuracy, but lower ROC-AUC than logistic regression; performance depends on threshold and class distribution.

The random forest achieved the higher accuracy in the recorded experiment, while logistic regression achieved the higher ROC-AUC. These measures describe different aspects of performance. Accuracy is the proportion of test cases classified correctly at a chosen threshold. ROC-AUC measures how well the model ranks positive cases above negative cases across different thresholds. Therefore, it would be misleading to declare one model universally superior from these two values alone.

The trained model and its metadata were saved for use by the application's local analysis workflow. A recorded example produced a probability of 68.72% for the model's positive class at a 0.5 threshold. This figure should be described as a model output for that particular input, not as a confirmed probability that a person has diabetes. It is sensitive to the input values, the training dataset, class definitions, preprocessing, and the model's calibration. It must not be used to make a diagnosis or to replace laboratory testing or professional medical advice.

A further limitation is that the selected dataset is based on survey indicators rather than direct continuous readings from the HealthSync wearable. As a result, the diabetes model and the BLE monitoring workflow are separate evidence streams: the former evaluates a dataset-trained classifier, while the latter tests how readings can move through the application. They should not be presented as though the diabetes model was trained on the ESP32 sensor data.

5.6 IoT and BLE Data-Flow Analysis

The IoT component is designed around a sequence of clear stages. A compatible device or simulator provides a payload; the BLE manager handles discovery, connection, and subscription; a telemetry parser converts incoming values into a structured health-data object; and the health service can then store or expose the values to the dashboard and AI analysis module. Separating these stages makes the system easier to debug. For example, if a reading is missing, the developer can determine whether the problem occurred during connection, characteristic subscription, payload parsing, storage, or display.

During software-level testing, a simulator can provide predictable readings to confirm that supported fields are parsed and passed to the application. A demonstration payload might include a heart rate of 82 beats per minute, SpO_2 of 97%, temperature of 36.7 °C, blood pressure of 118/78 mmHg, 120 steps, 0.08 km of distance, 6.5 calories, 300 seconds of active time, and a battery level of 91%. These values are illustrative test data only. If used in a screenshot, the figure caption should say “simulated telemetry example” and must not imply that they were measured from a real person or validated by a medical sensor.

The physical wearable is planned around an ESP32 and suitable sensors. Until the assembled device has been tested, software simulation can verify the data-handling pipeline but cannot confirm sensor accuracy, BLE range, battery life, measurement stability, or performance during movement. Physical testing should compare the wearable readings against an appropriate reference device, document the test conditions, and report the number of samples and observed differences. Any comparison involving medical measurements should be supervised appropriately and should not be used to claim clinical accuracy without a suitable validation study.

The system should also tolerate intermittent connectivity. If the device disconnects, the interface should clearly indicate that new readings are no longer arriving and should preserve the last measurement's timestamp. Missing values should remain missing rather than being filled with zeros, because zero may be interpreted as a real measurement. These safeguards are important for trustworthy monitoring.

5.7 Data Storage, Privacy, and Reliability

SQLite provides a lightweight local database that suits a desktop prototype and avoids requiring a paid hosted database. The database design separates user accounts, sessions, health profiles, devices, vitals, activity data, telemetry, location history, emergency contacts, and analysis reports into logical entities. Foreign-key constraints help maintain relationships between records. Keeping each user's records linked to an authenticated user identifier supports account-level data separation.

Local storage also has limitations. Data on the same computer may be exposed if the operating-system account or project files are not protected. Database relationships and password hashing are useful safeguards, but they do not by themselves provide encryption at rest, secure backup, tamper detection, or protection from a compromised device.

Reliability also depends on external services. Hospital lookup using OpenStreetMap data through Overpass can return useful nearby results, but the response depends on network access, query limits, map completeness, and the quality of the underlying map data. Search results should be presented as informational results and users should verify emergency details through official sources. Outbreak analysis should only display data from a traceable source with a date and geographic scope. If reliable data is unavailable, the interface should say so instead of generating a heatmap from fabricated values.

5.8 Advantages of HealthSync AI

1 Unified Desktop Workflow

The main advantage of HealthSync AI is its attempt to bring several related tasks into one application. A user can navigate between a health profile, device monitoring, AI analysis, emergency information, and hospital lookup without needing to open a separate tool for every task. The modular architecture also makes the project easier to extend as more components become operational.

2 Local Data Management

A local SQLite database reduces dependence on a continuously available remote database and keeps the prototype usable without subscription fees. It also gives the developer direct control over the data schema and testing process. Local storage is not automatically secure, however, so access control, operating-system protection, backup practices, and privacy safeguards remain necessary.

3 Low-Cost and Replaceable Components

The project uses open-source software and a service-layer approach where practical. BLE devices can be tested with a simulator before the physical ESP32 band is ready, which lowers early development costs and allows the user interface and parsing logic to be tested independently. Open hospital data can also be used without paying for a commercial lookup API, although service availability and data coverage are not guaranteed.

4 Scope for Personalised Analysis

Combining profile details, activity data, vital readings, and historical records creates a foundation for more context-aware summaries than a page of isolated numbers. The quality of this advantage depends on the completeness and reliability of the incoming data and on the validation of the analysis rules. HealthSync AI should therefore be presented as a monitoring and informational support tool, not as a replacement for healthcare professionals.

5.9 Limitations and Challenges

Limitation	Effect on results	Suggested improvement
Hardware validation is incomplete	Simulator results do not prove the accuracy of the physical band	Assemble the ESP32 device and compare readings with suitable reference equipment
Dataset mismatch	The diabetes model uses survey indicators, not live wearable sensor streams	Document feature mapping and evaluate on an appropriate, ethically sourced dataset
No clinical validation	Model outputs cannot be treated as diagnoses or clinical risk estimates	Conduct independent validation with qualified oversight before any clinical use
BLE compatibility and disconnection	Different devices may use different services, characteristics, or payload formats	Test supported devices, timeouts, reconnection, and malformed payloads
Internet-dependent lookup	Hospital results may be delayed or unavailable during network/service failure	Add clear error states, timestamps, and verified fallback contact information
Local database security	Local files may be exposed on an inadequately protected computer	Review permissions, sensitive logging, backup, encryption, and deletion policies
Emergency action limitations	A desktop app may not be able to place calls or send messages automatically on every system	Clearly show which actions are implemented and test the chosen communication method
Outbreak data availability	A map without a reliable source may misrepresent the situation	Use dated, traceable public data and show source, update time, and geographic coverage

The most important limitation is the difference between a working software prototype and a validated health product. A program can correctly save a reading or display a model result without proving that the reading is accurate or the prediction is clinically meaningful. This report should distinguish implementation success from medical validation and should avoid unsupported claims about accuracy, early diagnosis, emergency response, or disease prevention.

5.11 Overall Discussion and Conclusion

The results indicate that HealthSync AI has a practical foundation for a low-cost desktop health-monitoring workflow. Its architecture brings together a PySide6 interface, local data storage, a BLE-oriented telemetry pipeline, and a dataset-trained machine-learning experiment. The use of a simulator makes it possible to test parts of the data path before physical hardware is complete, while the diabetes model experiment provides measurable classification results that can be discussed quantitatively.

The machine-learning results show a trade-off between the tested models: random forest produced higher accuracy, whereas logistic regression produced a higher ROC-AUC. This difference demonstrates why model evaluation should use more than one metric. Likewise, a successful simulator-to-interface test shows that the software can handle a test payload, but it does not prove that a physical sensor provides medically accurate readings. These distinctions are central to a fair interpretation of the project.

The next development priorities are to complete and test the ESP32 wearable, validate sensor readings against suitable reference devices, expand the automated test suite, improve handling of missing and disconnected data, and ensure that every external result has a traceable source. The AI module should be evaluated with additional metrics and independent data before it is used for anything beyond educational screening support. Emergency and outbreak functions should be described according to their actual implementation status.

In conclusion, HealthSync AI demonstrates how desktop software, IoT communication, local data management, and machine learning can be organised into one extensible project. Its strongest result is the integration approach and the ability to test individual modules within a single workflow. Further hardware testing, data validation, usability evaluation, and security review are required before the system could be considered suitable for real-world health monitoring.

References

[1] S. D. Mamdiwar, A. R., Z. Shakruwala, U. Chadha, K. Srinivasan, and C.-Y. Chang, "Recent advances on IoT-assisted wearable sensor systems for healthcare monitoring," Biosensors, vol. 11, no. 10, Art. no. 372, Oct. 2021, doi: 10.3390/bios11100372.

[2] F. Sabry, T. Eltaras, W. Labda, K. Alzoubi, and Q. Malluhi, "Machine learning for healthcare wearable devices: The big picture," Journal of Healthcare Engineering, vol. 2022, Art. no. 4653923, 2022, doi: 10.1155/2022/4653923.

[3] S. Shajari, K. Kuruvinashetti, A. Komeili, and U. Sundararaj, "The emergence of AI-based wearable sensors for digital health technology: A review," Sensors, vol. 23, no. 23, Art. no. 9498, 2023, doi: 10.3390/s23239498.

[4] R. Qureshi et al., "Artificial intelligence and biosensors in healthcare and its clinical relevance: A review," IEEE Access, vol. 11, pp. 61600–61620, Jun. 2023, doi: 10.1109/ACCESS.2023.3285596.

[5] M. Nasr et al., "Smart healthcare in the age of AI: Recent advances, challenges, and future prospects," IEEE Access, vol. 9, pp. 145248–145270, Oct. 2021, doi: 10.1109/ACCESS.2021.3118960.

[6] S. C. Mukhopadhyay, N. K. Suryadevara, and A. Nag, "Wearable sensors for healthcare: Fabrication to application," Sensors, vol. 22, no. 14, Art. no. 5137, Jul. 2022, doi: 10.3390/s22145137.

[7] K. T. Putra, A. Z. Arrayyan, N. Hayati, F. Firdaus, C. Damarjati, A. Bakar, and H.-C. Chen, "A review on the application of Internet of Medical Things in wearable personal health monitoring: A cloud-edge artificial intelligence approach," IEEE Access, vol. 12, pp. 21437–21452, Jan. 2024, doi: 10.1109/ACCESS.2024.3358827.

[8] "A comprehensive review on wireless healthcare monitoring: System components," IEEE Access, vol. 12, pp. 35008–35032, Feb. 2024, doi: 10.1109/ACCESS.2024.3349547.

[9] H. S. Saad, J. F. W. Zaki, and M. M. Abdelsalam, "Employing of machine learning and wearable devices in healthcare system: Tasks and challenges," Neural Computing and Applications, vol. 36, pp. 17829–17849, Aug. 2024, doi: 10.1007/s00521-024-10197-z.

[10] "IoT based health monitoring: A systematic review," BOHR International Journal of Internet of Things, Artificial Intelligence and Machine Learning, vol. 3, no. 1, pp. 29–37, Aug. 2024, doi: 10.54646/bijiam.2024.22.

[11] "IoT & ML-based healthcare monitoring system—Review," in Proc. 2022 IEEE International Symposium on Smart Electronic Systems (iSES), Warangal, India, Dec. 2022, doi: 10.1109/iSES54909.2022.00137.

[12] “Edge intelligence and Internet of Things in healthcare: A survey,” IEEE Access, vol. 9, pp. 45–59, Dec. 2020, doi: 10.1109/ACCESS.2020.3045115.

[13] “Medical sensors and their integration in wireless body area networks for pervasive healthcare delivery: A review,” IEEE Sensors Journal, vol. 22, no. 5, pp. 3860–3877, Mar. 2022, doi: 10.1109/JSEN.2022.3141064.

[14] “The application of wearable smart sensors for monitoring the vital signs of patients in epidemics: A systematic literature review,” Journal of Ambient Intelligence and Humanized Computing, vol. 14, no. 5, pp. 6027–6041, 2023, doi: 10.1007/s12652-020-02656-x.

[15] “Leveraging machine learning for personalized wearable biomedical devices: A review,” Journal of Personalized Medicine, vol. 14, no. 2, Art. no. 203, Feb. 2024, doi: 10.3390/jpm14020203.

[16] S. Baratchi et al., “Wearable sensors: At the frontier of personalised health monitoring, smart prosthetics and assistive technologies,” Biosensors and Bioelectronics, vol. 176, Art. no. 112946, 2021, doi: 10.1016/j.bios.2020.112946.

[17] “Machine learning for cardiovascular outcomes from wearable data: Systematic review from a technology readiness level point of view,” JMIR mHealth and uHealth, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/35044316/.

[18] M. B. Kulkarni, S. Rajagopal, B. Prieto-Simón, and B. W. Pogue, “Recent advances in smart wearable sensors for continuous human health monitoring,” Talanta, vol. 272, Art. no. 125817, May 2024, doi: 10.1016/j.talanta.2024.125817.

[19] F. Saeedi, R. Ansari, and M. Haghgoo, “Recent development in wearable sensors for healthcare applications,” Nano-Structures & Nano-Objects, Art. no. 101473, May 2025, doi: 10.1016/j.nanoso.2025.101473.

[20] “AI-powered wearable sensors for health monitoring and clinical decision making,” Current Opinion in Biomedical Engineering, vol. 36, Art. no. 100628, Dec. 2025, doi: 10.1016/j.cobme.2025.100628.

[21] “Role of artificial intelligence in health monitoring using IoT based wearable sensors: A survey,” Internet of Things, vol. 34, Art. no. 101761, Nov. 2025, doi: 10.1016/j.iot.2025.101761.

[22] “Wearable smart sensors integration with AI and machine learning for tracking human health,” Biosensors and Bioelectronics: X, vol. 27, Art. no. 100711, Dec. 2025, doi: 10.1016/j.biosx.2025.100711.

[23] “Wearable sensors for health monitoring: Current applications, trends, and future directions,” Biosensors and Bioelectronics: X, vol. 28, Art. no. 100727, Mar. 2026, doi: 10.1016/j.biosx.2025.100727.

[24] “A comprehensive review of wearable devices for non-invasive biosensing,” TrAC Trends in Analytical Chemistry, vol. 193, Art. no. 118425, Dec. 2025, doi: 10.1016/j.trac.2025.118425.

[25] “AI-driven real-time monitoring of cardiovascular conditions with wearable devices: Scoping review,” JMIR mHealth and uHealth, vol. 13, 2025, doi: 10.2196/73846.

[26] Centers for Disease Control and Prevention, “2015 BRFSS survey data and documentation,” 2015. [Online]. Available: https://www.cdc.gov/brfss/annual_data/annual_2015.html.

[27] B. B. McFee et al., “Diabetes Health Indicators Dataset,” based on the CDC Behavioral Risk Factor Surveillance System (BRFSS) 2015 survey. [Online]. Available: https://github.com/BabsMatt/Diabetes-Health-Indicators.

[28] OpenStreetMap contributors, “OpenStreetMap.” [Online]. Available: https://www.openstreetmap.org/.

[29] Python Software Foundation, “Python 3 documentation.” [Online]. Available: https://docs.python.org/3/.

[30] scikit-learn developers, “scikit-learn: Machine learning in Python.” [Online]. Available: https://scikit-learn.org/stable/.