Ingénierie des Systèmes d'Information
Vol. 31, No. 7, July, 2026, pp. 2463-2475
Journal homepage: http://iieta.org/journals/isi

An IoT-Enabled Cross-Platform Mobile Health Framework for Real-Time Cardiac Monitoring, Data Visualization, and Intelligent Alerting

Fatima De Los Angeles Pacheco-Anton[1,2], Fabrizzio Alejandro Pauca-Muñoz[1,2], Allimson Julissa Borromeo-Quispe[1,2], Alex Jhoan Lozano-Villalobos[1,2], José Miguel Pariona-Sánchez[1,2], Maritza Raquel Cabana-Cáceres[1,2], Cristian Castro-Vargas[1,2*]

[1] Escuela Profesional de Ingeniería Informática, Facultad de Ingeniería Electrónica e Informática (FIEI), Universidad Nacional Federico Villarreal (UNFV), Lima 15082, Peru
[2] Comunidad de Conocimiento en Sistemas Inteligentes e IoT para la Formación en Ingeniería, Facultad de Ingeniería Electrónica e Informática (FIEI), Universidad Nacional Federico Villarreal (UNFV), Lima 15082, Peru

Corresponding Author Email: ccastrov@unfv.edu.pe

https://doi.org/10.18280/isi.310727

Received: 26 April 2026
Revised: 29 June 2026
Accepted: 15 July 2026
Available online: 31 July 2026

Keywords:
Internet of Things, mobile health, cardiovascular monitoring, cloud-based healthcare, cross-platform application, real-time monitoring, alert systems

ABSTRACT

The increasing demand for continuous cardiovascular monitoring has accelerated the development of Internet of Things (IoT)-enabled mobile health solutions capable of delivering accessible and timely physiological information. However, many IoT healthcare systems focus primarily on sensor development, while the mobile interaction layer responsible for data interpretation, synchronization, and user notification receives comparatively limited attention. This study presents an IoT-enabled cross-platform mobile health framework for real-time cardiac monitoring, visualization, and alerting, implemented as the CardioSense application. The proposed framework integrates a Flutter-based mobile interface with a Firebase cloud back-end to support user authentication, real-time data synchronization, historical trend analysis, and automated notifications from a companion IoT sensing device. The application architecture consists of four functional modules: real-time status monitoring, historical records, alert management, and user profile services. A scenario-based functional evaluation was conducted through live demonstration sessions involving adult volunteers, covering six essential operations, including registration, authentication, real-time visualization, alert generation, historical tracking, and email notification. All tested functions were successfully completed without execution errors. During the evaluation period, the application correctly displayed device-generated physiological states, including normal and arrhythmia classifications, and processed 28 recorded measurements across a seven-day demonstration window. The proposed framework demonstrates the feasibility of combining cross-platform mobile development and cloud-based IoT connectivity for exploratory cardiovascular self-monitoring applications. Further studies involving usability assessment, security evaluation, and clinical validation are required before healthcare deployment.

1. INTRODUCTION

Cardiovascular diseases remain the leading cause of death worldwide: the World Health Organization estimates that 19.8 million people died from cardiovascular causes in 2022, representing approximately 32% of all global deaths [1]. This burden is unevenly distributed, with over three-quarters of cardiovascular deaths occurring in low- and middle-income countries where continuous, low-cost monitoring infrastructure remains scarce. In Peru, national health-survey data indicate a hypertension prevalence of approximately 23% among adults, based on analysis of the nationally representative Demographic and Family Health Survey (ENDES) [2], underscoring a concrete, local need for accessible tools capable of supporting preventive cardiovascular screening outside specialized clinical settings.

Interactive mobile technologies have become a central channel through which Internet of Things (IoT) sensing devices deliver value to end users, supporting real-time monitoring by translating raw sensor data into accessible, actionable visual information, a need recurrently identified across reviews of IoT-based smart-healthcare technologies [3]. In the domain of mobile health (mHealth) and broader e-health applications, the mobile application is often the primary touchpoint through which a patient, caregiver, or student researcher interprets continuous physiological data. User interface design, data synchronization reliability, and timely alerting are therefore as critical to the system's value as the sensing hardware itself, a point emphasized in recent work on UI/UX design principles specific to mobile health applications [4]. A substantial body of work addresses the hardware design of low-cost IoT vital-sign sensors, including comprehensive wearable cardiovascular monitoring systems [5] and IoT platforms engineered to handle large volumes of patient-

monitoring data [6]. Comparatively less attention is given to the mobile-application layer that mediates between the device and the end user as a distinct interactive mobile technology. A poorly designed interface, an unreliable synchronization mechanism, or the absence of timely alert systems can undermine the practical usefulness of an otherwise accurate sensing device, a concern reinforced by umbrella-review evidence on the usability criteria most frequently neglected in mHealth application design [7]. It should be noted that many recent mHealth and IoT healthcare studies already incorporate mobile dashboards, cloud synchronization, and alert mechanisms. The present work does not claim this integration is absent from the field. Instead, it reports the design, implementation, and functional verification of one such mobile-application layer, CardioSense, as a companion contribution to an existing IoT sensing device. This layer is documented in sufficient technical detail to be independently assessed and reused.

This work addresses the following applied research question: whether a cross-platform mobile-application layer, built with a commodity UI framework and a real-time cloud back-end, can correctly and reliably surface device-generated cardiac classifications and alerts to an end user without introducing its own synchronization or interpretation errors. It further asks which architectural decisions are required to achieve this. To this end, the aim of this paper is to design, implement, and functionally evaluate CardioSense, a cross-platform mobile application for the real-time visualization and alerting of cardiac vital signs acquired by a companion IoT device. The principal contributions of this work are as follows:

(1) The design of an interactive, mobile-first user interface organized around patient status, historical trends, and alerts, intended to minimize navigation depth and support rapid, at-a-glance interpretation by non-specialist users.

(2) The implementation of secure user authentication and cloud-based real-time data synchronization using a Flutter and Firebase architecture, allowing near-instantaneous propagation of sensor-derived classifications to the user interface.

(3) The implementation of an automated, multi-channel alerting mechanism, combining in-app notifications and email alerts, for out-of-range physiological readings reported by the companion sensing device.

(4) A functional, scenario-based evaluation of the application under live demonstration conditions, verifying the correctness of registration, authentication, real-time status display, alerting, and historical data review.

The remainder of this paper is organized as follows. Section 2 reviews related work on mobile health applications and interactive mobile technologies. Section 3 describes the methodology and software architecture of the application. Section 4 presents the results of the functional evaluation. Section 5 formalizes the aggregation and severity-thresholding logic executed by the application. Section 6 discusses the findings in relation to related interactive mobile health technologies. Section 7 concludes the paper and outlines future research directions.

2. RELATED WORK

The design of mobile health applications for IoT-based vital-sign monitoring has been addressed from multiple angles in the recent literature. Galavi et al. [7] conducted an umbrella review of usability evaluation criteria for mHealth applications and identified that the most frequently overlooked criteria are related to interface simplicity, navigation depth, and timely feedback, findings directly relevant to the navigation-architecture decisions in CardioSense. Ashraf et al. [8] presented an open-source IoT health-monitoring system with cloud integration, while Bhagat et al. [9] evaluated Flutter as a cross-platform mobile development framework, reporting that its widget library and hot-reload cycle substantially reduce development iteration time relative to native approaches. Saraf et al. [10] reviewed Firebase as a mobile back-end service, identifying its real-time database listener model as a key architectural advantage for data-driven applications that require low-latency screen updates. Lovrić et al. [11] validated Flutter's suitability for health-domain interfaces in an independent evaluation using a psycho-oncological counselling application, reporting adequate tooling maturity for health-sector use despite the framework's comparatively young age. Bhimanapati et al. [4] proposed UI/UX design principles specifically for mobile health applications, emphasizing the importance of minimizing cognitive load through flat navigation hierarchies and color-coded status indicators. These principles informed the four-module tab structure and the traffic-light alert escalation scheme implemented in this work.

On the underlying IoT messaging layer, Wytrębowicz et al. [12] compared messaging protocols for IoT systems and reported latency and overhead advantages for publish-subscribe alternatives over simple request-response messaging at larger deployment scales. Kim et al. [13] proposed a design and evaluation framework specifically for mHealth applications, emphasizing minimal navigation depth and immediate status visibility upon login, principles reflected in the CardioSense interface. Regarding the companion sensing hardware, Barral Vales et al. [14] documented fine time-synchronization techniques for ESP32-based IoT devices. Mujeeb Rahman et al. [15] presented an ESP32-based wireless patient monitor validated under live patient-data conditions, a real-time field-testing protocol comparable to the one followed for the device underlying this application. Filgueiras et al. [16] evaluated the accuracy of low-cost wearable sensors against medical-grade equipment, while Saha et al. [17] reviewed machine-learning deployment workflows for microcontroller-class hardware relevant to the companion device's on-device classifier. Finally, Georgieva-Tsaneva et al. [18] reported an IoT-based cardio-monitoring system for underserved contexts, and Uddin and Koo [19] reviewed multi-hop IoT patient-monitoring architectures aimed at extending remote-monitoring reach.

Regarding the companion sensing hardware specifically, several ESP32/MAX30102-based prototypes have been reported without an equivalent dedicated mobile-application layer. Dobbertin-Sanchez and Chinchay-Espino [20] presented a low-cost IoT oximeter built on a NodeMCU and MAX30102 sensor, targeting Peru's non-specialized population, but limited to a local web server without cloud alerting. Contardi et al. [21] coupled an ESP32 to a MAX30102 sensor for continuous point-of-care oxygen-saturation and heart-rate monitoring through a similar local-web-server architecture. Ya'acob et al. [22] reported a real-time ESP32-based wireless health-monitoring system with stable IoT communication but no local display or mobile client. At a higher level of analytical sophistication, Alnajjar [5] described a wearable IoT platform integrating multi-

parameter cardiovascular risk prediction, achieving greater diagnostic depth at substantially higher hardware cost and design complexity. Relative to these antecedents, the present work contributes a functionally validated, end-to-end implementation that integrates all components, namely Flutter-based UI, Firebase back-end, multi-channel alerting, and IoT sensor coupling, into a single verified system. The interactive mobile-technology stack itself, rather than the underlying sensor, is the primary documented contribution here.

To make the positioning of this work relative to the studies discussed above more explicit, Table 1 consolidates the application purpose, sensing platform, mobile-side component, evaluation method, and reported limitations of each closely related system. As the table shows, most of the directly comparable ESP32/MAX30102-based prototypes either omit a dedicated mobile client entirely or do not report its evaluation method; this is the specific gap this work addresses. Broader mHealth literature not restricted to this hardware pairing does include mobile dashboards and alerting, so this table should be read as a comparison within the narrower ESP32/MAX30102 hardware family, not as a claim about the mHealth field as a whole.

Table 1. Structured comparison of related mobile-enabled Internet of Things (IoT) vital-sign monitoring systems

Study	Application Purpose	Sensing Platform	Mobile Component	Evaluation Method	Reported Limitations
This work (CardioSense)	Real-time visualization and multi-channel alerting for cardiac vital signs	ESP32 + MAX30102	Dedicated cross-platform client (Flutter)	Functional scenario testing (6 functions, live demo)	No usability instrument, no security audit, no participant demographics recorded
Alnajjar [5]	Multi-parameter cardiovascular risk prediction	Wearable IoT + advanced analytics	Not specified	Not specified in source	Higher hardware cost and design complexity
Ashraf et al. [8]	Open-source IoT health monitoring with cloud integration	ESP32 + sensors	Not specified	Not specified in source	Mobile-layer evaluation not reported
Dobbertin-Sanchez and Chinchay-Espino [20]	Low-cost SpO_2/HR oximetry for underserved population	NodeMCU + MAX30102	None (local web server only)	Bench comparison vs reference oximeter	No mobile client; no remote alerting
Contardi et al. [21]	Continuous point-of-care oxygen-saturation and heart-rate monitoring	ESP32 + MAX30102	None (local web server only)	Bench accuracy comparison	No mobile client; no cloud sync
Ya'acob et al. [22]	Real-time wireless health monitoring	ESP32	Local display only, no dedicated mobile client	Live patient-data field test	No mobile-side interface or remote alerting

3. MATERIALS AND METHODS

3.1 Development approach

CardioSense was developed following a waterfall software-engineering process comprising requirements gathering, interface design, implementation, and verification phases, consistent with established practice for small-team mobile health application development reported for comparable open-source IoT health-monitoring systems [8].

The application was implemented in Flutter, Google's open-source UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter was selected because it is currently among the most widely adopted cross-platform mobile frameworks, offering a rich, customizable widget library, a hot-reload development cycle, and first-class support for cloud back-end integration [9], properties documented to reduce development time and maintenance cost relative to maintaining separate native Android and iOS codebases [10].

This choice is further supported by an independent evaluation of Flutter in the development of a psycho-oncological counselling application, which found the framework's widget ecosystem and tooling maturity well suited to health-domain interfaces despite its comparatively young age [11].

3.2 Requirements analysis

Functional requirements were defined as follows: the application must allow a new user to register and authenticate securely; it must display the current physiological status of the monitored individual in real time; it must provide access to a chronological history of past measurements; it must generate and display alerts when out-of-range values are detected; and it must allow the user to review and edit profile information. Non-functional requirements specified a simple, mobile-first interface requiring minimal training, synchronization latency imperceptible to the end user, and resilience to transient connectivity loss.

3.3 Software architecture and cloud integration

As an interactive mobile technology, the application follows a client-cloud architecture in which Flutter acts as the presentation and interaction layer, while a cloud back-end, Firebase, provides authentication, cloud synchronization for live status updates, and a document-oriented store for historical records and user profiles. Firebase was selected for this role because its real-time listener model allows UI widgets to subscribe directly to data changes without manual polling. This design choice was motivated by general messaging-protocol comparisons for IoT systems, which report overhead advantages for publish-subscribe patterns over conventional request-response polling. That finding is not specific to

healthcare applications, and no dedicated overhead measurement was performed for CardioSense itself [12], while its authentication module and per-path security rules provide a standard mechanism for restricting read/write access to authenticated users only, consistent with security and connectivity requirements emphasized for connected-health monitoring ecosystems [23]. The resulting alert system, one of the application's core alert systems for critical-value notification, issues email-based push notifications automatically to the registered user when the companion IoT device reports a value outside the configured physiological range, providing an asynchronous alerting channel that does not require the application to remain in the foreground. Figure 1 summarizes the resulting client-cloud architecture, from sensor acquisition through Firebase synchronization to the CardioSense client and its alerting channels.

The security posture of this architecture rests on Firebase's built-in authentication service and declarative per-path security rules, which restrict read and write operations to authenticated users. No additional, application-level security measures were implemented or evaluated in this work. Specifically, data-at-rest and data-in-transit encryption were not independently verified beyond Firebase's default platform-level protections. No penetration testing or access-control testing was performed against the deployed security rules, and no formal privacy or vulnerability assessment was conducted. These security and privacy dimensions are treated here as an explicit limitation rather than an implicit assumption of adequacy, and a structured security evaluation is identified as necessary future work in Section 7.

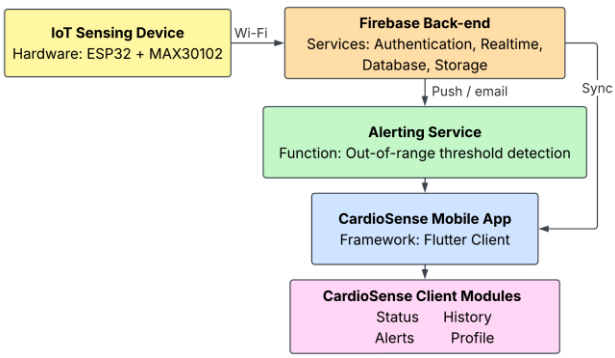

Figure 1. CardioSense client-cloud architecture

3.4 NoSQL data architecture and server-side synchronization

Beyond the general client-cloud pattern described above, the persistence layer of the companion IoT device follows a hierarchical NoSQL path convention designed to keep per-patient records isolated and efficiently queryable. Each reading is written under a path of the form /patients/{PATIENT_ID}/readings, where PATIENT_ID is a fixed identifier configured on the device; this hierarchical structure allows the mobile client to subscribe only to the subtree relevant to the authenticated user, rather than scanning a flat collection of all measurements across all patients. Figure 2 shows the function signature and the route-construction logic responsible for assembling this path on the device side, together with the use of a Firebase-generated server-side timestamp (the.sv = "timestamp" directive) when writing each reading.

```
void sendDataToFirebase(float bpm, float spo2, float temp, int prediction, float confidence) {

  // Safe route construction
  String path = "/patients/";
  path += String(PATIENT_ID);
  path += "/readings";

  Serial.print("📍 Ruta completa: ");
  Serial.println(path);

  Serial.print("🌐 Database URL: ");
  Serial.println(firebaseConfig.database_url.c_str());

  // Build JSON with explicit values
  FirebaseJson json;

  // Server timestamp
  json.set("timestamp/.sv", "timestamp");
```

Figure 2. Route-construction logic and server-side timestamp used when writing a reading to the NoSQL hierarchy

The use of a server-side timestamp, rather than a timestamp generated locally on the microcontroller, is a deliberate architectural choice. It removes any dependency on the device's internal clock, which is not synchronized to a network time source in the current prototype. It also ensures that the chronological ordering of historical readings reflects the moment of arrival at the cloud back-end rather than a potentially drifting local clock. This convention is consistent with the broader event-driven synchronization model described in Section 3.3, in which the mobile client reacts to changes detected by the cloud back-end rather than polling the device directly [12].

3.5 Session-based processing and connectivity resilience

Figure 3 summarizes the device-side communication flow underlying the architecture described above: the cyclic acquisition-and-filtering loop, the session-based accumulation window, the branch between local display and cloud synchronization, and the conditional alert trigger when a reading falls outside the configured physiological range. The same one-second acquisition cycle continues for as long as the device remains powered, regardless of whether a given iteration closes a measurement session or triggers an alert.

To mitigate transient sensor noise and avoid transmitting unstable readings to the cloud back-end, the companion device accumulates samples into a measurement session rather than forwarding each individual reading as soon as it is captured. Figure 4 shows the corresponding MeasurementSession data structure, which holds running sums for heart rate, SpO_2, and temperature, a sample counter, a session-start timestamp, and the final prediction and confidence once the session is closed. The session is finalized only after a minimum sample count is reached, at which point the running sums are converted into averages, the heart-rate-variability feature is computed, the on-device classifier is invoked, and the resulting averaged reading, rather than any single noisy sample, is the one that is published to the cloud database.

Figure 5 shows the corresponding finalization routine, which guards against premature closure by requiring a minimum sample quorum before computing the averages and chaining the classifier invocation directly into the cloud-synchronization call. This sequencing, namely averaging followed by classification followed by transmission, ensures that the value written to the NoSQL hierarchy described in Section 3.4 always corresponds to a complete, quorum-validated session rather than a single instantaneous sample.

Beyond session-level filtering, the system's network layer is designed to degrade gracefully rather than fail outright when cloud connectivity is unavailable. Upon Wi-Fi association, the device attempts Firebase authentication with a bounded 20-

second timeout. If the cloud back-end does not respond within this window, the firmware falls back to a local-only operating mode (firebaseReady = false). In this mode, sensor acquisition, on-device classification, and local TFT display continue uninterrupted, while cloud synchronization is simply skipped until connectivity is restored. Figure 6 shows the corresponding timeout-and-fallback logic.

3.6 Interface design and navigation structure

The user interface design is organized around a persistent bottom navigation bar exposing four primary modules, summarized in Table 2. This flat, tab-based navigation pattern was chosen to minimize the number of taps required to reach any core function. This information-architecture principle was identified as a key factor of user experience and satisfaction in a validated mHealth user-experience evaluation scale [13].

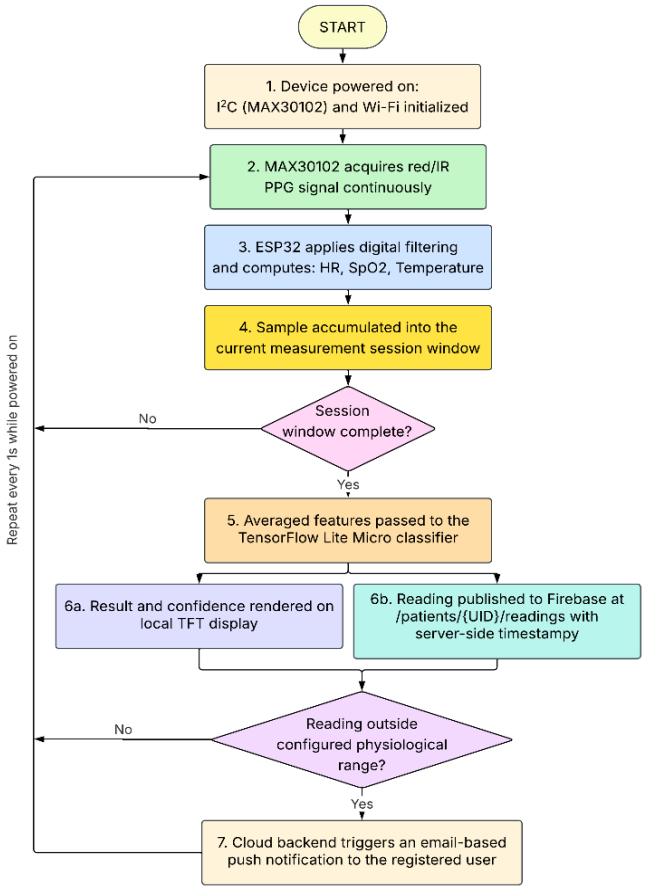

Figure 3. Device-side communication flow, from signal acquisition to cloud synchronization and alerting

```
    struct MeasurementSession {
    float totalBpm = 0;
    float totalSpO2 = 0;
    float totalTemp = 0;
    int sampleCount = 0;
    unsigned long startTime = 0;
    bool isActive = false;
    int finalPrediction = -1;
    float finalConfidence = 0;
};

MeasurementSession currentSession;
```

Figure 4. Data structure used to accumulate and finalize a measurement session before transmission

```
void finishMeasurementSession() {
    if (!currentSession.isActive || currentSession.sampleCount < MIN_SAMPLES_FOR_AVERAGE) {
        Serial.println("⚠ Muestras insuficientes");
        currentSession.isActive = false;
        return;
    }

    float avgBpm = currentSession.totalBpm / currentSession.sampleCount;
    float avgSpO2 = currentSession.totalSpO2 / currentSession.sampleCount;
    float avgTemp = currentSession.totalTemp / currentSession.sampleCount;

    float variability = calculateBPMVariability();
    int prediction = predictCondition(avgBpm, avgSpO2, avgTemp, variability);

    currentSession.finalPrediction = prediction;
    currentSession.finalConfidence = predictionConfidence;
    currentSession.isActive = false;

    sendDataToFirebase(avgBpm, avgSpO2, avgTemp, prediction, predictionConfidence);
    displayPrediction(prediction, predictionConfidence);
}
```

Figure 5. Session-finalization routine chaining averaging, classification, and cloud synchronization

```
unsigned long startAuth = millis();
    const unsigned long authTimeout = 20000; // 20 seconds maximum

    while (!Firebase.ready() && (millis() - startAuth < authTimeout)) {
        Serial.print(".");
        delay(500);
        yield();
    }

    if (Firebase.ready()) {
        firebaseReady = true;
        uid = String(firebaseAuth.token.uid.c_str());
        Serial.print("\n✅ Firebase OK! UID: ");
        Serial.println(uid);
    } else {
        Serial.println("\n⚠ Firebase timeout - continuando sin DB");
        firebaseReady = false;
    }
```

Figure 6. Bounded authentication timeout and graceful fallback to local-only operation

Table 2. CardioSense navigation modules

Module	Primary Function
Status	Displays the current physiological classification (such as Normal or Arrhythmia), confidence percentage, last-update timestamp, and live heart-rate reading streamed from the IoT device.
History	Presents aggregated statistics (such as average heart rate and SpO_2 over a selectable period) and a time-series chart of recent measurements, with access to the full chronological log.
Alerts	Lists all generated alerts in reverse-chronological order, each tagged with severity (such as critical or high), the triggering parameter, and the time elapsed since the event.
Profile	Allows the user to review personal data, change the account password, sign out, and access in-app information on data-privacy policy and open-source licensing.

4. RESULTS

4.1 Authentication and onboarding flow

New users created an account by supplying an email address and password through a dedicated registration screen; successful registration triggered a confirmation message and redirected the user to the login screen. Figure 7 shows the login screen with the CardioSense logo, email and password fields, and a sign-in button. Returning users authenticated through the same credentials, after which the application transitioned to the main status screen shown in Figure 8, which displays a Normal classification with 95.0% confidence and a live heart-

rate reading of 75.0 bpm synchronized in real time from the companion IoT sensing device, an onboarding pattern consistent with low-cost IoT oximeter and vital-sign monitoring proposals that similarly prioritize a minimal-friction first-use experience [20, 24]. A password-recovery option is available from the login screen for users who forget their credentials.

Figure 7. CardioSense login screen

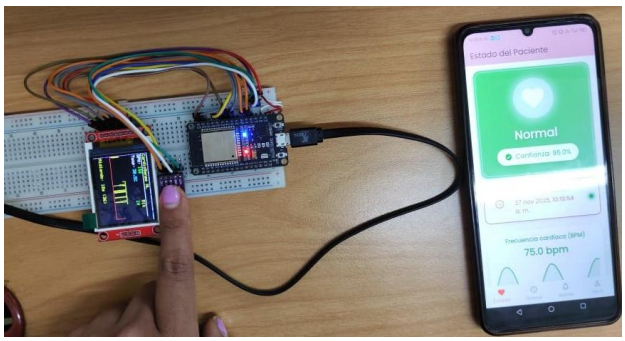

Figure 8. CardioSense status screen (Normal state)

4.2 Functional evaluation procedure

Because CardioSense is the interface layer of a broader IoT vital-sign real-time monitoring system, its evaluation in this stage focused on functional correctness rather than a structured usability-metric study (such as the System Usability Scale). The application was exercised during live demonstration sessions in which adult student volunteers registered new accounts, signed in, and interacted with the companion sensing device while observing the corresponding screen transitions and data updates in real time. This functional, scenario-based evaluation approach is consistent with early-stage validation practice reported for wearable IoT cardiovascular monitoring prototypes prior to formal usability or clinical trials [5], and aligns with practitioner-reported expectations for remote patient-monitoring tools regarding ease of adoption and integration into daily workflows [25]. Figure 9 shows the application immediately after a successful authentication event, displaying the confirmation banner 'Inicio de sesión exitoso' and the four-module tab bar (Status, History, Alerts, Profile) that anchors the application's user interface design, confirming correct onboarding behavior and navigation-structure rendering across test sessions.

4.3 Observed functional behavior

Table 3 summarizes the functional checks performed and their observed outcomes across the core application modules.

4.4 Alerting behavior under critical readings

Figure 10 summarizes the evaluation results graphically: all six verified functions received a Pass result, confirming that the complete interaction loop from registration through alerting and historical review operates correctly end-to-end.

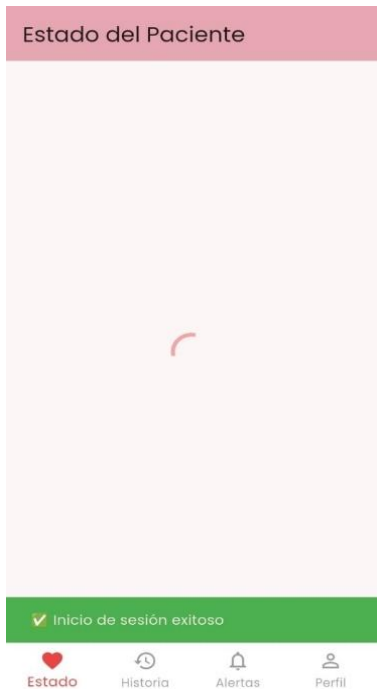

Figure 9. CardioSense post-login screen confirming successful authentication and navigation bar

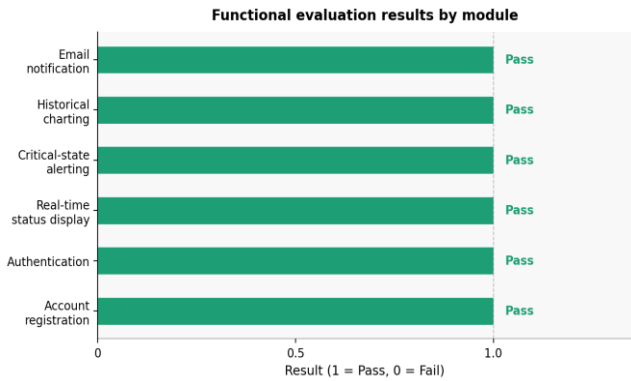

Figure 10. Functional evaluation results by module (all six functions: Pass)

Table 3. Functional verification checklist and observed outcomes

Function Verified	Observed Outcome
Account registration	New account created successfully; confirmation message displayed; user redirected to login screen.
Authentication	Login completed with valid credentials; "Login successful" message displayed prior to navigation to the status screen.
Real-time status display	Live heart-rate value and classification label updated on screen consistently with the value reported by the companion sensing device.
Critical-state alerting	Out-of-range readings correctly triggered a visually distinct alert card (such as "Arrhythmia" at 100.0% confidence) and a corresponding entry in the Alerts module.
Historical charting	Aggregated statistics and a multi-day heart-rate trend line rendered correctly from stored historical records.
Email notification	An automated email alert was received at the registered address upon detection of a critical reading.

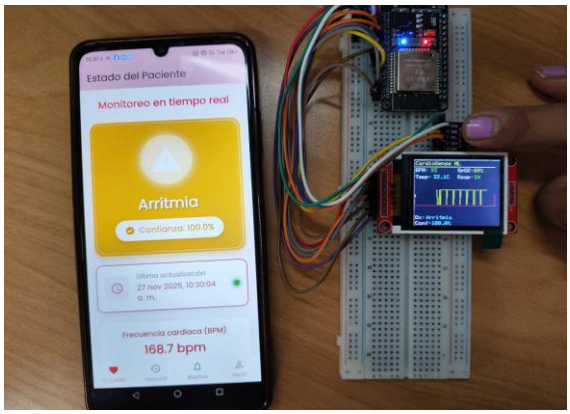

Figure 11. CardioSense alert screen (Arrhythmia state)

Figure 11 shows the application's status screen during a session in which the companion device, built on a MAX30102-over-ESP32 acquisition pipeline similar to designs reported for continuous point-of-care oxygen-saturation monitoring [21] and other ESP32-based IoT device implementations [26], reported an arrhythmia-pattern classification. The interface immediately replaced the normal green status indicator with a high-visibility amber alert card displaying the classification label, a confidence percentage, and the underlying heart-rate value (168.7 bpm), together with a precise timestamp of the last update. This visual escalation design, color, iconography, and information density all shifting in response to severity, was motivated by randomized evaluations of push-based mobile health notifications in an unrelated clinical domain. Those studies report that timely, salient alerts measurably increase user engagement and responsiveness, but this design was not directly tested against that evidence. No equivalent user study was conducted for CardioSense, so the actual effect of this design on user responsiveness remains to be measured [27] and is consistent with the design considerations recommended for sensing platforms targeting chronic-disease populations who depend on rapid, unambiguous status cues [28].

4.5 Historical tracking and alerts modules

The History module aggregated seven days of measurements into summary statistics (mean heart rate, mean SpO_2, and minimum/maximum bounds) and rendered them alongside a trend line, allowing the user to visually identify drift over time without manually inspecting individual records, a pattern of real-time field validation similar to that reported for ESP32-based wireless health-monitoring systems tested under live patient-data conditions [22]. Figure 12 shows the History screen displaying a 7-day bpm trend, mean SpO_2 of 93.1%, and 28 measurements logged in the evaluation period.

Figure 12. CardioSense history screen showing 7-day bpm trend

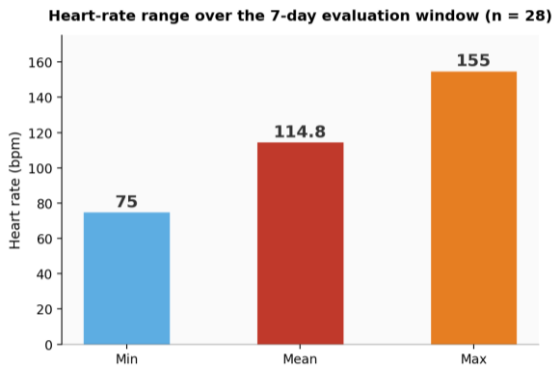

Figure 13. Heart-rate range (minimum, mean, maximum) over the 7-day evaluation window

Figure 13 summarizes the same heart-rate range numerically: a minimum of 75 bpm, a mean of 114.8 bpm, and a maximum of 155 bpm over the $n = 28$ readings logged during the seven-day window, computed by the application using the running mean and incremental extrema described in Section 5.1.

Table 4 consolidates the two classification states explicitly verified during the evaluation, contrasting the reported confidence percentage, heart rate, and severity tier for each, and providing a single reference point for the case-specific figures discussed throughout Section 4.

Table 4. Observed classification states during the functional evaluation

State	Confidence	Heart Rate	Severity
Normal	95.0%	75.0 bpm	Normal
Arrhythmia	100.0%	168.7 bpm	Critical

Table 5. Representative alert events logged by triggering parameter

Triggering Parameter	Recorded Value	Severity Tier
Oxygen saturation (SpO_2)	89.7%	Critical
Heart rate	131.8 bpm	Critical
Temperature	Below configured threshold	Logged (non-critical in this instance)

Figure 14. CardioSense Alerts module listing logged events by severity

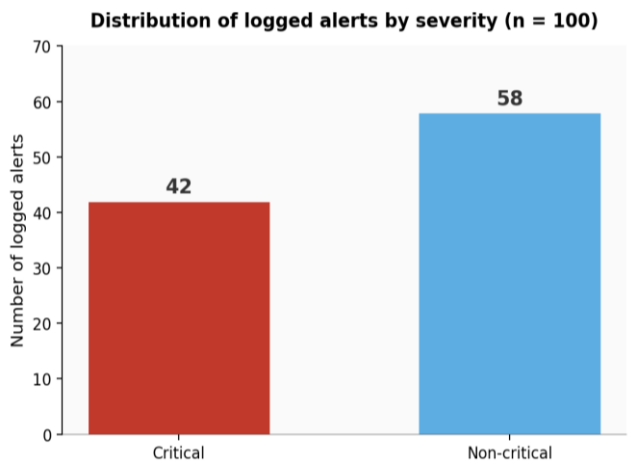

Figure 15. Distribution of logged alerts by severity (n = 100)

Figure 16. CardioSense push notification on the Android lock screen

The Alerts module listed all generated events in reverse-chronological order, each tagged with a severity label (such as critical or high) and a human-readable description of the triggering condition (such as low SpO_2, elevated heart rate, or low temperature), confirming that alert metadata generated on the device side was correctly received, parsed, and rendered by the mobile client, consistent with the end-to-end reliability goals emphasized in broader IoT healthcare-monitoring reviews [29]. Figure 14 shows the Alerts module listing 100 logged events for the test account, including a low-SpO_2 reading (89.7%, tagged critical), an elevated heart rate (131.8 bpm, tagged critical), and a low-temperature reading, each with its corresponding elapsed-time stamp.

Table 5 consolidates the specific event types visible in this alert log, contrasting the triggering parameter, the recorded value, and the resulting severity tier for the three representative cases discussed above. Although the full 100-event log spans a broader range of recorded values, these three cases illustrate that the system correctly differentiates between distinct physiological triggers (oxygen saturation, heart rate, and temperature) rather than applying a single undifferentiated alert rule.

Of the 100 logged alerts shown in this session, 42 were tagged as critical and 58 as non-critical, as summarized graphically in Figure 15, indicating that the alert-classification logic distributed severity labels across a realistic mix of conditions rather than flagging every out-of-range reading as critical.

It is important to state precisely what this session-log analysis does and does not demonstrate. The severity label attached to each alert is generated on the companion sensing device according to the thresholding rule described in Section 5.2. The mobile application only receives, parses, and renders this pre-computed label. No independent verification of true-positive or false-positive rates against a clinical ground truth was performed at either the device or application level. Likewise, the elapsed time from event generation to on-screen and push-notification delivery was not systematically measured during this evaluation, so no detection-latency or notification-delivery-time figures are reported. The 42/58 critical/non-critical distribution should therefore be read as evidence that the application correctly displays a heterogeneous mix of severities generated upstream, not as a measured detection-accuracy or latency result. Quantifying

alert-generation accuracy and end-to-end notification latency against a reference standard is identified as necessary future work in Section 7.

Beyond the in-app alert list, Figure 16 shows the corresponding push notification rendered on the Android lock screen ('CardioSense Alerta: ... Confianza: 100.0%'), confirming that the alerting pipeline correctly reaches the operating-system notification layer in addition to the in-app Alerts module.

Figure 17. CardioSense profile screen

Figure 18. CardioSense informational screen ('About CardioSense')

4.6 Profile and informational screens

Registration and login flows completed without errors across repeated trials, including the expected handling of duplicate-account and invalid-credential conditions. Figure 17 shows the Profile module displaying the authenticated user's name with shortcuts to the status and history views, alongside account-management and informational sections (data-privacy policy and open-source license disclosures).

Figure 18 shows the corresponding 'About CardioSense' informational screen, confirming that the application's static informational content, namely app description, version, open-source license disclosures, and privacy-policy access, was integrated consistently with its dynamic, data-driven screens, an integration pattern also reported for portable IoT-based health-monitoring frameworks targeting home-based patient care [30].

5. CLIENT-SIDE DATA PROCESSING AND DECISION RULES

The equations formalized in this section describe standard statistical aggregation and rule-based thresholding operations; they are presented not as a novel algorithmic contribution, but to specify precisely and unambiguously the client-side computations performed by the mobile application, since the correctness of exactly these operations is what the functional evaluation in Section 4 verifies.

5.1 Aggregated statistics

Although the physiological classification itself is computed on the companion IoT device, the CardioSense History module performs its own on-device aggregation of the raw readings retrieved from Firebase before rendering the seven-day trend chart and the summary statistics shown in Figure 12. For a measurement window containing n readings $x_1, x_2, \ldots, x_n$ of a given variable (heart rate or SpO_2), the application computes the arithmetic mean as follows, using a single left-to-right accumulation pass to remain efficient on resource-constrained mobile hardware:

$$\bar{x} = (1/n) \cdot \sum x_i \; (i = 1, \ldots, n) \qquad (1)$$

where, $\bar{x}$ is the displayed mean (the 93.1% mean SpO_2 value shown in Figure 12, for instance), and n is the number of measurements retrieved for the selected window (n = 28 for the seven-day window evaluated in Section 4). The minimum and maximum bounds shown alongside the mean are obtained directly as $\min(x_1, \ldots, x_n)$ and $\max(x_1, \ldots, x_n)$ over the same window, without requiring a separate pass, since both extrema are tracked incrementally during the same accumulation loop used for Eq. (1).

5.2 Confidence and severity thresholding

The confidence percentage displayed alongside each classification (such as 95.0% for Normal or 100.0% for Arrhythmia, as shown in Figures 8 and 11) is received directly from the companion device's on-device classifier; it is not recomputed by the application. CardioSense is responsible only for rendering this value and for evaluating it against the

alerting thresholds described in Section 3.3. An alert is escalated to the critical severity tier whenever a received reading r falls outside the configured physiological bounds $[r_{min}, r_{max}]$, as expressed in Eq. (2):

$$severity(r) = \begin{cases} Critical, & if\ r < r_{\min}\ or\ r > r_{\max} \\ Normal, & otherwise \end{cases} \quad (2)$$

This thresholding logic is consistent with the 100 logged alerts analyzed in Section 4.3, of which 42 were tagged critical and 58 non-critical (Figure 15), confirming that the rule expressed in Eq. (2) was applied consistently rather than flagging every out-of-range reading as critical by default.

6. DISCUSSION

6.1 Interpretation of the functional evaluation

The functional checks summarized in Table 3 indicate that the core interaction loop of CardioSense, namely authentication, real-time status synchronization, alert escalation, and historical review, operates as designed when paired with the companion IoT sensing device. The visually distinct alert state observed in Section 4.3 suggests that the chosen escalation pattern is likely to support rapid interpretation by non-specialist users; however, this expectation was not yet confirmed through a structured usability instrument, and should be treated as a design hypothesis pending formal testing. Table 6 contrasts the non-functional requirements defined in Section 3.2 with the corresponding evaluation outcome.

The comparison criteria in Table 7 are intentionally limited to hardware stack, connectivity, mobile-client presence, and alerting, because factors such as response latency, usability outcomes, security posture, scalability, battery consumption, and measurement accuracy are not consistently reported in the cited source publications and cannot be responsibly estimated for systems this work did not implement or test. Extending this comparison along those additional dimensions would require either re-implementing the compared systems under a common test protocol or contacting their original authors for unpublished data, neither of which was undertaken here; the present table should therefore be read as a structural and architectural comparison rather than a comprehensive performance benchmark.

Beyond hardware positioning, the decision to build CardioSense in Flutter aligns with the broader trend toward cross-platform frameworks in health-oriented mobile development, where a single codebase reduces the engineering burden of maintaining parity between Android and iOS releases [8]. Similarly, the choice of a real-time, listener-based Firebase back-end over a heavier request-response API mirrors architectural recommendations distilled from messaging-protocol comparisons reported for IoT systems more broadly [12], and is consistent with the time-synchronization and connectivity design constraints documented for ESP32-class IoT companion devices operating over Wi-Fi [14]. The resulting interface design also reflects mHealth usability evaluation guidance on minimizing navigation depth and surfacing status information immediately upon login [13].

Table 6. Non-functional requirements versus evaluation outcomes

Requirement (Section 3.2)	Target	Observed Outcome	Met
Interface simplicity	Minimal training required	Tab-based navigation, single tap to any module	Yes
Synchronization latency	Imperceptible to user	Near-instantaneous status updates observed	Yes
Connectivity resilience	Tolerates transient loss	Not systematically stress-tested	Partial
Alerting reliability	Timely critical-value alerts	In-app and email alerts confirmed functional	Yes
Usability validation	Not formally specified	Functional only; no validated instrument applied	Partial

Table 7. Comparative positioning of CardioSense against related ESP32/MAX30102-based monitoring systems

Study	Hardware Stack	Connectivity	Dedicated Mobile Client	Alerting
This work (CardioSense)	ESP32 + MAX30102 + TFT	Wi-Fi + Firebase (cloud)	Yes (Flutter)	Email + push, multi-channel
Alnajjar [5]	Wearable IoT	IoT + advanced analytics	Not specified	In-app only
Ashraf et al. [8]	ESP32 + sensors	IoT cloud	Not specified	Not specified
Dobbertin-Sanchez & Chinchay-Espino [20]	NodeMCU + MAX30102	Local web server	No	None
Contardi et al. [21]	ESP32 + MAX30102	Local web server	No	None
Ya'acob et al. [22]	ESP32	Real-time IoT	No (local display only)	Not specified

6.3 Limitations of the study

Several limitations qualify the present findings. The evaluation reported here is functional and scenario-based; it does not include a structured, validated usability instrument (such as the System Usability Scale) or a statistically powered sample of end users. Long-term reliability of push and email notifications under intermittent network connectivity was not systematically stress-tested. Accessibility considerations, such as screen-reader compatibility and color-contrast compliance for users with visual impairments, were not formally assessed in this stage of development. Finally, the historical-trend visualization was evaluated only over short observation windows, so longer-term data-retention and chart-scalability behavior remain to be characterized.

The functional-evaluation participants are described only as adult student volunteers affiliated with the developing institution; the exact number of participants, their age range, technical background, and recruitment procedure were not formally recorded during this stage and cannot be reported here. This is treated as a limitation of the evaluation's reporting completeness rather than omitted by choice, and a

fully documented recruitment and demographic protocol is planned for the structured usability study identified as future work in Section 7.

Similarly, the physiological values and classification outputs discussed throughout the Results section (Section 4) were received from the companion IoT sensing device rather than computed, calibrated, or independently verified by the mobile application itself. This application-level evaluation should therefore not be read as validating the accuracy, calibration procedure, or measurement environment of the underlying sensor. That sensor is documented and evaluated separately in the companion hardware publication. What this work verifies is that the application correctly receives, displays, and reacts to values generated upstream, not the correctness of those values themselves.

6.4 Implications for interactive mobile health technologies

Beyond its specific role as the companion interface to a vital-sign sensing device, CardioSense illustrates a broader pattern relevant to the interactive mobile technologies field: the pairing of a cross-platform UI framework, cloud synchronization, and lightweight alert systems is reported in the literature to lower the engineering effort required to deliver a responsive, alert-capable mobile health experience relative to maintaining separate native codebases. This specific work did not measure development time or maintenance cost directly; it instead relies on the functional outcome of a working, verified implementation as supporting evidence. The underlying design pattern is also reflected in wireless ESP32-based patient-monitor designs that pursue similarly low development overhead [15]. This is particularly attractive for academic and early-stage prototyping contexts where development resources are limited, paralleling the resource-aware design philosophy behind low-cost wearable sensor evaluations [16] and microcontroller-class machine-learning deployment workflows aimed at minimizing infrastructure requirements [17]. These same resource constraints motivate continued attention to accessible IoT-based cardio-monitoring system designs [18] and to cloud-connected, multi-hop patient-monitoring architectures aimed at extending reach into underserved settings [19].

7. CONCLUSION

This paper presented CardioSense, a cross-platform mobile application developed in Flutter to provide real-time monitoring, historical tracking, and automated alerting of cardiac vital signs acquired by a companion IoT sensing device. A functional evaluation conducted during live demonstration sessions confirmed correct end-to-end behavior across account registration, authentication, real-time status synchronization, critical-state alert escalation, and historical data review. The application's reliance on a real-time, listener-based cloud back-end enabled near-instantaneous propagation of sensor-derived classifications to the user interface, while its tab-based navigation structure kept all core functions accessible within a single tap from any screen. Many IoT vital-sign monitoring studies treat the mobile client as a secondary, incidental display layer and concentrate validation effort on the sensing hardware alone. This work instead documents the interactive mobile-technology stack itself as a functionally verified, reusable contribution. The combination of a cross-platform UI framework, a real-time cloud back-end, a tab-based information architecture, and a multi-channel alerting mechanism spanning in-app and email notifications can be decoupled from the specific underlying sensor. It can therefore be reapplied to other low-cost IoT health-monitoring and e-health applications with comparable real-time and alerting requirements.

Several limitations qualify these findings and define concrete directions for future work. The evaluation reported here is functional and scenario-based rather than a structured usability study, so future work should prioritize: (i) conducting a structured usability evaluation with a validated instrument, such as the System Usability Scale or an mHealth-specific user-experience scale, applied to a statistically meaningful sample of representative end users; (ii) performing a formal accessibility audit, including screen-reader compatibility and color-contrast compliance for users with visual impairments; (iii) stress-testing the notification pipeline under realistic intermittent-connectivity conditions to quantify delivery latency and failure rates; and (iv) extending the historical module to support longer retention windows and richer comparative analytics across sessions. In direct response to further gaps identified during review, three additional items are added to this agenda: (v) a structured security evaluation covering data encryption verification, access-control testing, and a formal vulnerability assessment; (vi) a measurement protocol for alert-generation accuracy against a clinical ground truth and for end-to-end notification latency; and (vii) a documented participant recruitment and demographic-reporting procedure for all future evaluation sessions. Taken together, the present results support continued development of CardioSense as a functionally verified prototype for the interactive mobile technologies field. Whether it can additionally serve as a practical tool for accessible, continuous cardiovascular self-monitoring remains an open question. Answering it depends on the usability, security, alert accuracy, and clinical validation studies identified above; this work is not yet positioned to draw that conclusion.

ETHICAL STATEMENT

All functional evaluation sessions described in this work were conducted with adult student volunteers who registered test accounts using non-identifying credentials and were verbally informed of the purpose of the demonstration prior to participation; informed verbal consent was obtained from all participants. No real patient data, clinical records, or personally identifying health information were processed or retained; all physiological values displayed and logged during testing originated from the companion sensing device operated on consenting volunteers for demonstration purposes only. A formal institutional ethics-committee review was not required for this functional-evaluation stage, as no clinical intervention or vulnerable population was involved; a structured usability study with appropriate ethical review is planned as part of future work, as noted in Section 7.

ACKNOWLEDGMENT

The authors thank the Facultad de Ingeniería Electrónica e Informática (FIEI), Universidad Nacional Federico Villarreal, for institutional support during the development of this work.

REFERENCES

[1] World Health Organization (WHO). (2025). Cardiovascular diseases (CVDs). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] Calderón-Ocón, V., Cueva-Peredo, F., Bernabe-Ortiz, A. (2024). Prevalence, trends, and factors associated with hypertensive crisis among Peruvian adults. Cadernos de Saúde Pública, 40(2): e00155123. https://doi.org/10.1590/0102-311XEN155123

[3] Ghazal, T.M., Hasan, M.K., Alshurideh, M.T., et al. (2021). IoT for smart cities: Machine learning approaches in smart healthcare—A review. Future Internet, 13(8): 218. https://doi.org/10.3390/fi13080218

[4] Bhimanapati, V.B.R., Pandian, P.K.G., Goel, P. (2024). UI/UX design principles for mobile health applications. Journal of Research in Pharmaceutical Sciences, 15(3): 216-231. https://doi.org/10.36676/jrps.v15.i3.1485

[5] Alnajjar, R. (2025). A low-cost, wearable IoT system for comprehensive cardiovascular health monitoring and disease risk prediction. TechRxiv Preprint. https://doi.org/10.36227/techrxiv.175691251.13763523/v1

[6] Waleed, M., Kamal, T., Um, T.W., Hafeez, A., Habib, B., Skouby, K.E. (2023). Unlocking insights in IoT-based patient monitoring: Methods for encompassing large-data challenges. Sensors, 23(15): 6760. https://doi.org/10.3390/s23156760

[7] Galavi, Z., Montazeri, M., Khajouei, R. (2024). Which criteria are important in usability evaluation of mHealth applications: An umbrella review. BMC Medical Informatics and Decision Making, 24(1): 365. https://doi.org/10.1186/s12911-024-02738-2

[8] Ashraf, S., Khattak, S.P., Iqbal, M.T. (2023). Design and implementation of an open-source and Internet-of-Things-based health monitoring system. Journal of Low Power Electronics and Applications, 13(4): 57. https://doi.org/10.3390/jlpea13040057

[9] Bhagat, S.A. (2022). Review on mobile application development based on Flutter platform. International Journal for Research in Applied Science and Engineering Technology, 10(1): 803-809. https://doi.org/10.22214/ijraset.2022.39920

[10] Saraf, P.R. (2022). A review on Firebase (backend as a service) for mobile application development. International Journal for Research in Applied Science and Engineering Technology, 10(1): 967-971. https://doi.org/10.22214/ijraset.2022.39958

[11] Lovrić, L., Fischer, M., Röderer, N., Wünsch, A. (2023). Evaluation of the cross-platform framework Flutter using the example of a cancer counselling app. In Proceedings of the 9th International Conference on Information and Communication Technologies for Ageing Well and E-Health, pp. 135-142. https://doi.org/10.5220/0011824500003476

[12] Wytrębowicz, J., Cabaj, K., Krawiec, J. (2021). Messaging protocols for IoT systems—A pragmatic comparison. Sensors, 21(20): 6904. https://doi.org/10.3390/s21206904

[13] Kim, G., Hwang, D., Park, J., Kim, H.K., Hwang, E.S. (2024). How to design and evaluate mHealth apps? A case study of a mobile personal health record app. Electronics, 13(1): 213. https://doi.org/10.3390/electronics13010213

[14] Barral Vales, V., Fernández, O.C., Domínguez-Bolaño, T., Escudero, C.J., García-Naya, J.A. (2022). Fine time measurement for the Internet of Things: A practical approach using ESP32. IEEE Internet of Things Journal, 9(19): 18305-18318. https://doi.org/10.1109/jiot.2022.3158701

[15] Mujeeb Rahman, K.K., Mohamed, N.N., Zidan, R., Alsarraj, I., Hasan, B. (2023). IoT-based wireless patient monitor using ESP32 microcontroller. In 2023 24th International Arab Conference on Information Technology (ACIT), Ajman, United Arab Emirates, pp. 1-6. https://doi.org/10.1109/acit58888.2023.10453847

[16] Filgueiras, T.P., Bertemes-Filho, P., Noveletto, F. (2025). Evaluating the accuracy of low-cost wearable sensors for healthcare monitoring. Micromachines, 16(7): 791. https://doi.org/10.3390/mi16070791

[17] Saha, S.S., Sandha, S.S., Srivastava, M. (2022). Machine learning for microcontroller-class hardware: A review. IEEE Sensors Journal, 22(22): 21362-21390. https://doi.org/10.1109/jsen.2022.3210773

[18] Georgieva-Tsaneva, G., Cheshmedzhiev, K., Tsanev, Y.A., Dechev, M., Popovska, E. (2025). Healthcare monitoring using an Internet of Things-based cardio system. IoT, 6(1): 10. https://doi.org/10.3390/iot6010010

[19] Uddin, R., Koo, I. (2024). Real-time remote patient monitoring: A review of biosensors integrated with multi-hop IoT systems via cloud connectivity. Applied Sciences, 14(5): 1876. https://doi.org/10.3390/app14051876

[20] Dobbertin-Sanchez, S.E., Chinchay-Espino, H.A. (2023). Low-cost IoT-based oximeter controlled by NodeMCU and MAX30102 targeting Peru's non-specialized population in 2022. SSRN, 4361276. https://doi.org/10.2139/ssrn.4361276

[21] Contardi, U.A., Morikawa, M., Brunelli, B., Thomaz, D.V. (2022). Max30102 photometric biosensor coupled to esp32-webserver capabilities for continuous point of care oxygen saturation and heartrate monitoring. Engineering Proceedings, 16(1): 9. https://doi.org/10.3390/iecb2022-11114

[22] Ya'acob, N., Ismail, S.I., Amin, M.Z.M., Aziz, N.F.A. (2024). Design and implementation of a wireless health monitoring system for real-time patient data using ESP32. In 2024 IEEE 15th Control and System Graduate Research Colloquium (ICSGRC), Shah Alam, Malaysia, pp. 149-153. https://doi.org/10.1109/icsgrc62081.2024.10691254

[23] Ianculescu, M., Constantin, V.Ș., Gușatu, A.M., et al. (2025). Enhancing connected health ecosystems through IoT-enabled monitoring technologies: A case study of the Monit4Healthy system. Sensors, 25(7): 2292. https://doi.org/10.3390/s25072292

[24] Retna Kumar, G.J., Arumugam, P.K., Mani Suriya, S., Vijayarajeswaran. (2023). Design and development a low-cost IoT-based health monitoring device. AIP Conference Proceedings, 2427(1): 020102. https://doi.org/10.1063/5.0101354

[25] Serrano, L.P., Maita, K.C., Avila, F.R., et al. (2023). Benefits and challenges of remote patient monitoring as perceived by health care practitioners: A systematic review. The Permanente Journal, 27(4): 100-111. https://doi.org/10.7812/tpp/23.022

[26] Hercog, D., Lerher, T., Truntič, M., Težak, O. (2023). Design and implementation of ESP32-based IoT devices. Sensors, 23(15): 6739. https://doi.org/10.3390/s23156739
[27] Hernández-Reyes, A., Cámara-Martos, F., Molina Recio, G., Molina-Luque, R., Romero-Saldaña, M., Moreno Rojas, R. (2020). Push notifications from a mobile app to improve the body composition of overweight or obese women: Randomized controlled trial. JMIR mHealth and uHealth, 8(2): e13747. https://doi.org/10.2196/13747
[28] Maqbool, S., Bajwa, I.S., Maqbool, S., Ramzan, S., Chishty, M.J. (2023). A smart sensing technologies-based intelligent healthcare system for diabetes patients. Sensors, 23(23): 9558. https://doi.org/10.3390/s23239558
[29] Abdulmalek, S., Nasir, A., Jabbar, W.A., et al. (2022). IoT-based healthcare-monitoring system towards improving quality of life: A review. Healthcare, 10(10): 1993. https://doi.org/10.3390/healthcare10101993
[30] Vyas, H., Shukla, H., Jivani, M.N. (2025). A portable IoT-based health monitoring framework using ESP32 for isolated and home-based patient care. Journal on Electronic and Automation Engineering, 4(2): 240-245. https://doi.org/10.46632/jeae/4/2/31