International Journal of Basic and Applied Sciences, 14 (2) (2025) 520-534

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/wf3p9b62
Research paper



Development of A Comprehensive IoT-Based Monitoring Application for High-Risk Pregnancies Enhancing Maternal and Fetal Health Outcomes

P. Bavithra Matharasi [1] *, Prabu Selvam [2], Himanshu Sharma [3], D. Sugumar [4], Midathada Vinay Kumar [5], S. Rosaline [6], Addanki Mounika [7], Tatiraju V. Rajani Kanth [8]

[1] *Department of Computer Science (MCA), Mount Carmel College, Autonomous, Bengaluru, Karnataka 560052, India*
[2] *School of Computing, SRM Institute of Science & Technology, Tiruchirapalli Campus, Tiruchirapalli, Tamil Nadu 621105, India*
[3] *Department of Computer Science & Engineering, IILM University, Greater Noida, Uttar Pradesh 201306, India*
[4] *Department of Electronics and Communication Engineering, Karunya Institute of Technology and Sciences (Deemed to be University), Coimbatore, Tamil Nadu 641114, India*
[5] *Department of Electronics and Communication Engineering, Avanthi Institute of Engineering and Technology (Autonomous), Cherukupally, Vizianagaram, Andhra Pradesh 531162, India*
[6] *Department of Electronics Engineering (VLSI Design and Technology), R.M.K. Engineering College, Kavaraipettai, Tamil Nadu 601206, India*
[7] *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh 522302, India*
[8] *Senior Manager, TVR Consulting Services Private Limited, Hyderabad, Telangana 500055, India*
**Corresponding author E-mail: p.bavithra.matharasi@mccblr.edu.in*

Received: May 10, 2025, Accepted: June 18, 2025, Published: June 30, 2025

Abstract

This paper presents the design, development, and testing of MAVATI, a mobile health application focused on perinatal monitoring for ex-expectant mothers and healthcare professionals. The system includes real-time monitoring of biomedical parameters, alert generation (both manual and automatic), and a personalized recommendation module. Developed as a minimum viable product (MVP) for both patient and physician interfaces, the application was evaluated through usability testing and load testing to validate its performance and user-friendliness. Usability tests demonstrated intuitive navigation and effective interaction with key functionalities, while load testing using Apache JMeter confirmed the system's scalability and reliability under concurrent access. The results indicate that MAVATI is a responsive and user-centered solution capable of supporting maternal healthcare monitoring. Further clinical validation is recommended to assess its real-world effectiveness and potential for data-driven research.

Keywords: *Perinatal Monitoring; Maternal Healthcare; Mobile Application; Real-Time Alerts.*

1. Introduction

Maternal health is a fundamental pillar of public health, particularly during the prenatal and perinatal periods when both the mother and the developing fetus are vulnerable to a wide range of risks. Effective prenatal and perinatal care is essential to protect and enhance the well-being of both the expectant mother and her unborn child. Numerous studies have shown that a mother's health during pregnancy significantly affects fetal development and can have lasting impacts on the child's future health. Furthermore, complications such as hypertensive disorders, gestational diabetes, and other maternal health problems that arise during pregnancy can often be predictive of similar chronic conditions later in the mother's life (Shafqat et al., 2021).

In this context, pregnancies deemed to be high-risk—defined by Khaire & Dhanalakshmi (2022) as those with potentially suboptimal maternal and/or fetal prognoses compared to a normal pregnancy—require special attention. These pregnancies account for approximately 20% of all cases but are responsible for over 80% of negative perinatal outcomes. These statistics emphasize the critical need for early detection, continuous monitoring, and timely medical intervention to mitigate complications and reduce maternal and fetal morbidity and mortality.

Maternal Mortality (MM), as defined by the World Health Organization (WHO, 2024), refers to the death of a woman during pregnancy, childbirth, or within 42 days following the end of pregnancy due to pregnancy-related causes. The global community, through the Sustainable Development Goals (SDGs), has committed to reducing the global maternal mortality ratio to less than 70 per 100,000 live births

Copyright © P. Bavithra Matharasi et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

by the year 2030. Achieving this target necessitates the implementation of effective strategies and systems that facilitate the timely identification of risks and appropriate medical care, particularly in resource-limited or socially vulnerable populations.

In India, maternal mortality remains a pressing public health issue. Socioeconomic inequality, barriers to healthcare access, and systemic shortcomings contribute to a high incidence of complications during pregnancy that often go undetected or untreated. According to data from the National Department of Statistics (DANE, 2021), the leading causes of maternal death include obstetric conditions not elsewhere classified (O95-O99), hypertensive disorders in pregnancy (O10-O16), and complications during labor and delivery (O60-O75). Alarmingly, a large proportion of maternal deaths are attributed to unspecified obstetric causes, highlighting a critical gap in the classification and understanding of maternal mortality, an issue that could be improved with more robust and continuous health monitoring systems.

To address these challenges, it is essential to implement a comprehensive, continuous, and accessible maternal health monitoring system that enables healthcare professionals to detect and respond to risk indicators in a timely and effective manner. As suggested by (Rostami et al., 2021; Choi et al., 2020; Alam et al., 2020; Allahem & Sampalli, 2020), such systems should integrate real-time biomedical monitoring, clinical data analysis, and communication tools to support proactive maternal care.

This proposal introduces a software-based solution aimed at improving the monitoring and care of high-risk pregnancies. The solution consists of a mobile application for healthcare professionals and patients, integrated with a smartwatch-based Internet of Things (IoT) device for the real-time collection of biomedical data. This system facilitates seamless, two-way communication between patients and providers and supports data-driven decision-making for prenatal care (Saarikko et al., 2020; Machorro-Cano et al., 2024; Al Khatib et al., 2024; Gong et al., 2023).

The proposed solution also aligns with national healthcare strategies such as the Maternal-Perinatal RIAS (Integral Health Care Routes), reinforcing existing frameworks through enhanced monitoring and patient engagement. It promotes early detection of complications, adherence to health programs, increased access to medical guidance, and comprehensive monitoring throughout pregnancy and postnatal care (Pilosof et al., 2021).

Ultimately, the implementation of this mobile monitoring system as a healthcare innovation can significantly contribute to reducing maternal mortality in India. It empowers healthcare professionals with timely data and strengthens the healthcare system's capacity to deliver targeted, individualized care to those most in need.

2. Literature review

Recent progress in wearable technology has greatly aided the creation of non-invasive systems for detecting fetal movement, designed to enhance prenatal supervision. Xu et al. (2022) developed a dual-accelerometer wearable device utilizing machine learning techniques like Support Vector Machines and Random Forests to effectively differentiate fetal movements from maternal actions, achieving high detection precision and showcasing the system's feasibility for a real-time, home-based application. In a similar vein, Qin et al. (2023) developed a compact and energy-efficient wearable gadget intended for expectant mothers, incorporating sophisticated signal processing methods to minimize noise from breathing and maternal movement, and utilizing a cloud-based system for remote observation. Somathilake et al. (2022) broadened their research by employing passive inertial sensors, like accelerometers and gyroscopes, to concurrently evaluate the well-being of both the fetus and the mother. Their system showed the ability to gather long-term data with little interference, employing statistical and machine learning models for precise classification.

In a wider scope of physiological tracking, Georgieva-Tsaneva et al. (2025) introduced an IoT-driven cardio monitoring system that, although not solely focused on fetal movement detection, is pertinent because it combines various biosignals, edge computing for instant analysis, and secure remote access—elements that are increasingly vital in maternal-fetal health systems. Simultaneously, acoustic-based detection has become more significant, as shown by Ouypornkochagorn et al. (2023), who created a wearable acoustic device utilizing signal processing methods such as spectral subtraction to separate fetal sounds from abdominal noise, resulting in enhanced detection sensitivity. Ghosh et al. (2024) advanced this method by integrating various sensor types—acoustic, inertial, and gyroscopic—into one wearable device, employing data fusion algorithms to enhance reliability and minimize false alarms, particularly in intricate settings with maternal movement.

Ultimately, advancing their research on acoustics, Ouypornkochagorn et al. (2025) introduced a convenient and portable acoustic system tailored for residential use. The system included enhanced sensor sensitivity and wireless communication features for transmitting data remotely to clinicians, emphasizing the significance of accessibility and usability for users without technical expertise.

Wang et al. (2024) used machine learning models, including random forest and gradient boosting, to predict pregnancy complications in women undergoing assisted reproduction. Their results showed that ML-based approaches outperformed traditional statistical methods in identifying risk factors such as gestational hypertension and diabetes, increasing the ability of data-driven risk prediction in obstetrical contexts.

Rahman et al. (2024) used Support Vector Machine (SVM) to classify pregnant women into risk categories based on vital signs and clinical indicators. Their work has shown that even simple models can effectively support maternal health screening, particularly in rural or resource-limited settings.

Khadidos et al. (2024) proposed an ensemble learning framework that combines multiple classifiers (e.g., XGBoost, Random Forest) to improve the prediction of maternal health risks. Their model achieved high accuracy and showed strong potential for real-time clinical decision support when integrated with wearable or cellular systems.

Dimitrov (2016) discussed the role of the Medical Internet of Things (MIoT) in healthcare with an emphasis on the technologies used and real-time monitoring. He highlighted issues such as data privacy, infrastructure, and collaboration – critical considerations for large-scale deployment of smart maternal health systems.

According to the WHO 2023 report, approximately 287,000 women died globally due to pregnancy and childbirth, with 94% of these deaths occurring in low- and middle-income countries. This highlights the need for AI-based maternal health solutions, such as those described in these studies, especially in line with Sustainable Development Goal 3 (SDG 3).

3. Solution design

3.1. Solution modeling

The solution modeling for the high-risk pregnancy monitoring application was carried out using a UML class diagram, which illustrates the core structure and interactions among the system's main components. At the heart of the design is the abstract class User, which defines common attributes shared by all system participants. This class is inherited by two key roles: Patient, representing pregnant individuals registered in the system, and Doctor, representing healthcare professionals providing prenatal care. Each Patient is associated with a PregnancyProfile, which captures relevant clinical and biomedical information throughout the pregnancy. To define care relationships, the Care association class links Doctors to Patients, allowing the system to manage which doctors are responsible for which patients.

The system incorporates an Alert mechanism through another abstract class, which branches into ManualAlert—created by users when they identify symptoms of concern—and AutomaticAlert, which is generated automatically when biometric measurements exceed defined thresholds. In response to alerts, a PatientRecommendation may be issued by the doctor, offering specific advice or instructions. The system also includes a FetalMovement class, enabling patients to log instances of perceived fetal activity. Additionally, enumerations (shown in Figure 1) are used to model specific data types such as user roles, alert severity, and monitored parameters. The complete structure and relationships among these classes are illustrated in the class diagram in Figure 2, forming a solid foundation for a scalable, responsive, and user-centered application architecture.

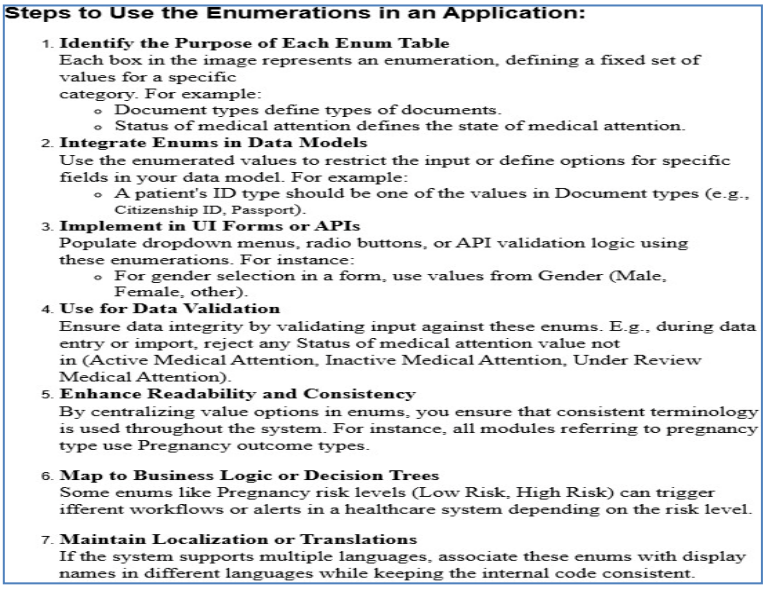


Fig. 1: Enumeration Steps for Core Application Types.

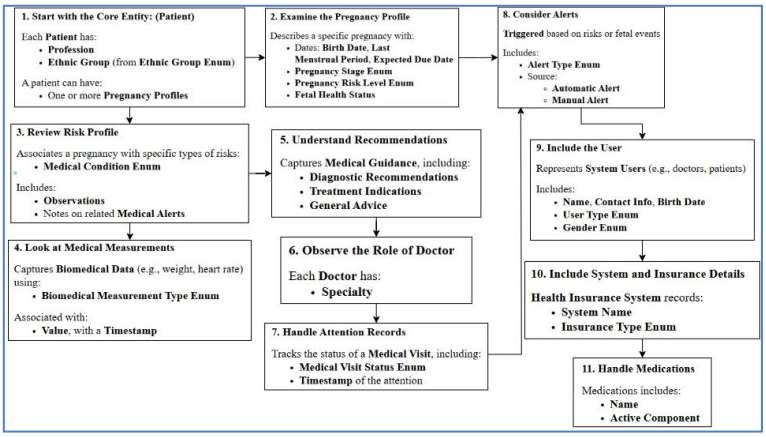


Fig. 2: Class Diagram for Maternal Health Monitoring System.

Since non-relational databases (NoSQL) were also used to store the patients' biomedical measurement data, it was necessary to model the documentary data. This model can be seen in Figure 3:

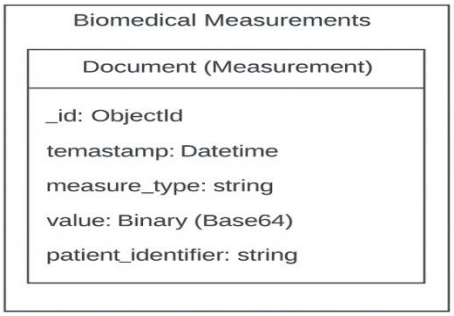


Fig. 3: Modeling the Documentary Data Structure.

As can be seen, the measurement value is binary because it is stored in an encrypted form. The date and time describe when the measurement was taken by the IoT device, or when it was added manually, the type of measurement stored (heart rate, blood oxygen percentage, etc.), and the patient's ID number.

3.1.1. Component model

The application components were modeled based on the functionalities required and the user stories gathered during development. The design emphasized identifying core architectural connectors and assigning responsibilities to each component. Six primary components were identified: User Manager, Monitoring Manager, Health Integration Manager, Persistence Manager, Biomedical Measurement Manager, and Graphical Interface (UI). Some of these components encapsulate smaller subcomponents, and their exposed and consumed interfaces were also modeled. The Health Integration Manager plays a vital role by enabling the application to interact with standard mobile health platforms, such as Apple Health, Google Fit, or Google Health Connect, to retrieve data from IoT devices like smartwatches. The Biomedical Measurement Manager oversees the creation and management of biomedical measurements and provides an interface used by the Alert Manager to generate automatic alerts based on abnormal health data. The User Manager handles user account operations, including registration and authentication, and exposes an interface utilized by the Biomedical Measurement Manager and the Monitoring Manager to verify user credentials. The Monitoring Manager coordinates all tasks related to monitoring, such as alert generation, issuing recommendations, and managing patient pregnancy profiles and related risk factors. Finally, the Persistence Manager oversees handling database operations, ensuring that other components can reliably access and store the data required for their functions.

3.2. Selected technologies

3.2.1. Data layer technologies

For the data layer, two complementary data management strategies were chosen to meet the intended purposes.

1) First, for data that required ACID structure and transactionality, such as user profiles, alerts, recommendations, pregnancy risks, and medications, the open-source relational database PostgreSQL was chosen. Since a high number of requests to the database is expected, the "Read Committed" isolation strategy was chosen for the PostgreSQL database configuration, considering its ability to offer an adequate balance between consistency and performance. According to Saarikko et al. (2020), this isolation level prevents dirty reads, meaning that a transaction will never read data written by an uncommitted transaction, which is crucial for maintaining data integrity without incurring a significant performance penalty.
2) Second, for data requiring high volume and continuous use, such as patient biomedical measurements, a NoSQL database called MongoDB was chosen. MongoDB is a document database that allows flexible and scalable data storage, which is essential for handling large volumes of information generated in real time by health monitoring devices.

Specifically, a data cluster was chosen on MongoDB's database-as-a-service (DBaaS) platform, called Atlas. MongoDB Atlas provides a managed solution that simplifies the configuration, operation, and scalability of MongoDB. Furthermore, this cluster is optimized to handle data received from IoT devices efficiently and with low latency in the capture and processing of biomedical data. IoT devices, such as the smartwatches used in the solution, generate data continuously and in large quantities. MongoDB Atlas can handle these demands thanks to its ability to distribute data across multiple servers and its horizontal scaling architecture.

Therefore, MongoDB Atlas was chosen to store patients' biomedical measurements due to its ability to scale horizontally, its flexibility in handling unstructured data, and the advanced security and management features it offers, all optimized for the specific needs of IoT devices in the healthcare environment.

3.2.2. Backend technologies

The key technologies used to develop the application's backend were:

1) FastAPI: The FastAPI library was used to implement the application's backend API in Python. FastAPI is known for its superior performance and support for creating modern, efficient APIs. This library implemented backend security features, which were managed using OAuth 2.0 and JSON Web Tokens (JWT) to secure API endpoints and manage user sessions.
2) SQL-Alchemy: An extremely popular Python library that functions as a programming model manager, allowing the mapping of relational database structures to Python objects. This functionality, known as Object Relational Mapping (ORM), significantly simplifies database operations.

3.2.3. Frontend technologies

Based on the decisions made regarding the development strategy, the decision was made to create a mobile application for both doctors and patients. The most relevant technologies and libraries used to achieve this goal were:

1) Flutter: A software development kit (SDK) maintained and developed by Google to facilitate the development of cross-platform mobile applications. This technology was chosen because it is very versatile when it comes to developing mobile applications that can be used on most current devices, as it works on both iOS and Android operating systems.

2) background_fetch: This Flutter library (background_fetch | Flutter package (pub.dev)) allows actions to be performed in the background. It works for both iOS and Android devices. Using Isolates in Dart, it is possible to create this concurrent task through multithreading. This library was chosen to allow reading of patients' biomedical measurements, even when the application is not actively being used. However, it has several limitations that will be explained in more detail later.
3) flutter_secure_storage: This Flutter library (flutter_secure_storage | Flutter package (pub.dev)) allows you to store sensitive data, such as biomedical measurements or session tokens, in encrypted form. It can be used for both Android and iOS. For Android devices, the AES algorithm is used, where the AES secret key is encrypted with RSA, and the RSA key is stored in KeyStore. Similarly, for iOS, Keychain is used to securely store data.
4) get: With this popular Flutter package (get | Flutter package (pub.dev)), you can optimize your application, make it more productive, and implement your chosen MVC model more efficiently, as a context is not required to navigate between routes or access the application's controllers.

3.2.4. Integration technologies

Among the integration technologies used for the implementation of the application, the following stand out:
1) fastapi-sqlalchemy: This is a Python library (FastAPI-SQLAlchemy • PyPI) that acts as a middleware, providing simple integration between FastAPI and SQLAlchemy in the application's backend. It offers useful features to facilitate common integration tasks between these two libraries.
2) Health: This is a Flutter package (health | Flutter package (pub.dev)) that allows connection to the main health application on mobile devices. In the case of Android, it connects to Google Fit, and in the case of Apple, to Apple Health. This library was chosen for its versatility in reading data on both platforms, allowing it to do so in real time, on demand, and efficiently. However, the choice of this library entails some limitations in the development of the application, which will be explained in more depth later.

3.3. Application design

3.3.1. Description of the architecture pattern

A service-oriented architecture (SOA) was implemented, as defined by Qin et al. (2023), which is defined as a set of distributed components that provide and/or consume services. In an SOA architecture, the components that offer services and those that consume them can use different programming languages and platforms. Services are largely autonomous: service providers and service consumers are typically deployed independently.
Therefore, this architecture was chosen as the appropriate one for developing the monitoring application. Development was focused on this approach, and the following independent services were developed:

- Backend: Implements services that encapsulate the logic and operations of the monitoring application based on the database. Each service is autonomous and can be deployed, updated, and scaled independently.
- API: Serves as the service exposure layer, allowing other components (such as the mobile application) to interact with the backend services through HTTPS calls. The API is designed following REST principles.
- Mobile Application (Client): Consumes the services exposed by the API to obtain, manipulate, and display data. The mobile application communicates with the API for all necessary operations.

Regarding the architectural pattern chosen for the development of the mobile application, a Model-View-Controller (MVC) architecture was chosen, as this pattern fits well with the chosen technology (Flutter SDK).
Regarding the architecture chosen for the deployment of the application and the backend, a microservices architecture was chosen using a serverless infrastructure focused on the "compute-as-back-end" architecture, which is described as an approach where a serverless computing service such as AWS Lambda and third-party services are used to build a backend for web, mobile, and desktop applications (Shafqat et al., 2021).

3.3.2. Quality attributes

To meet the implementation of the non-functional requirements explained in Chapter 2, several design strategies were addressed.
1) Performance: A microservices architecture was chosen because it allows individual services to be scaled and optimized. This is because more resources can be allocated to critical microservices, or those that require more resources, to meet latency and performance requirements.

In addition, an application load balancer was implemented to balance incoming traffic to each backend microservice and thus effectively manage concurrency. Therefore, the AWS Application Load Balancer service was used to balance traffic.
Additionally, it is important to consider that caching is a computing technique used to temporarily store data in a cache, allowing for faster retrieval for future requests for the same data. This technique can be implemented directly by hardware components or programmatically within a software system's architecture. The benefits of caching include improved application performance in terms of latency, response time, and IOPS (input/output operations per second), reduced power consumption, and reduced backend load. The key to effective caching lies in having a temporary cache that is significantly faster than recalculating the data.
Thus, for the mobile application, several forms of optimization and performance improvement were implemented, such as the use of caching strategies to reduce the computational cost of accessing data. In this regard, a caching strategy was implemented using the SQLite database, where user data, alert data, recommendation data, and assurance system data are stored. This reduces the number of requests required to the backend and the database, lowering computational costs and improving application performance.

2) Security: Data security and privacy are very important to comply with legislation; therefore, robust encryption strategies were used to encrypt sensitive data. However, it is also important to consider the impact of encryption on application performance, making a balanced trade-off between security and performance. Thus, for this case, information security was considered in its three possible states:

- In transit: When data is in transit, for example, from the application to the backend, or in transit from one microservice to another, the HTTPS protocol is used to encrypt this communication. Additionally, TLS mutual authentication is used, where both the client and the server must authenticate each other.

- At rest: Since data travels encrypted via the HTTPS protocol from the application to the backend, it was decided to encrypt the average value in the backend. Therefore, it was decided to use a library that provided encryption algorithms, as these play a vital role in protecting data from unauthorized access, and there are many available.

To choose the best option, the robustness of the encryption and the performance, understood as the time it takes the algorithm to encrypt the information, were considered. Thus, considering the arguments given by (Shafqat et al., 2021), the Advanced Encryption Standard (AES) 256-bit algorithm was chosen, as it is one of the most efficient algorithms and is widely supported and adopted in hardware and software. Likewise, AES's shorter encryption time is notable compared to other popular algorithms, as evidenced in Figure 4.

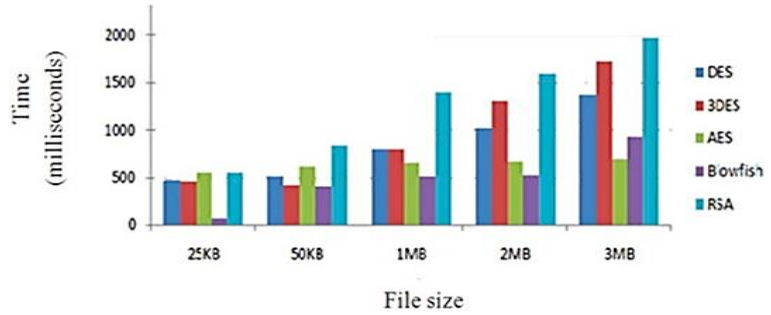


Fig. 4: Performance Comparison – Encryption Algorithms.

As can be seen, compared to other encryption algorithms, AES has outstanding performance for different input data sizes. The Python cryptography library (cryptography • PyPI) was used to implement this encryption.

- In use: For biomedical data used in mobile applications for patients and doctors, a cache was used to temporarily store data while it is used by different functionalities. This cache is securely stored using the Flutter Secure Storage library and is automatically purged after being sent to the database.

To manage user sessions, the OAuth 2.0 strategy is used, in which user sessions are managed through JSON Web Tokens (JWT) session tokens. These tokens are also stored securely, as are biomedical measurements, and this cache is also cleared after the tokens expire.

3) Scalability: To ensure the application's scalability, the AWS Fargate service was used, as explained above, as it is a serverless service that allows scaling on demand with an automatic auto-scaling group. This is how AWS documentation explains it: "AWS Fargate, the serverless computing engine for Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Services (EKS), enables customers to scale applications faster, improve performance, and reduce wait times. We've made several improvements over the past year to scale applications up to 16 times faster, making it easier to build and run larger-scale applications on Fargate" (Amazon Web Services Inc., n.d.).
4) Availability: To meet this quality attribute, we aim to minimize technological downtime as much as possible. When analyzing the deployment view, elements were identified that could represent single points of failure, such as registering microservice containers in Amazon ECS and not using any self-managed serverless services like EC2. Therefore, a better and more resilient solution was proposed: using fault-tolerant, highly available AWS cloud services, also ensuring replication to ensure high availability and self-management. For these reasons, Amazon Fargate was also chosen as the appropriate service.

In addition, leveraging the features provided by the AWS cloud, the concept of availability zones was used. These zones are data centers distributed within the same AWS region, physically separated from each other but connected by highly redundant networks. This concept was used to deploy containers in different availability zones and ensure greater availability.

5) Maintainability: A development strategy was chosen to ensure loose coupling and high cohesion. The model-view-controller (MVC) architecture style also implements a logical and modular separation of the system. Additionally, serverless services in the AWS cloud, such as the Fargate cluster, were used, eliminating the need to worry about the infrastructure layer or server maintenance.
6) Usability: By implementing an intuitive user interface, it was possible to ensure that users could easily manage the applications and complete the required tasks in a fluid and consistent manner. To this end, the applications for doctors and patients were designed following a general framework, but adapting the graphical interfaces to the specific needs of each role. This enabled a personalized and efficient user experience, suitable for both medical professionals and patients.
7) Portability: To meet this quality attribute, it was decided to use Docker containers to run the application backend, as explained above. These containers are deployed using AWS Fargate serverless clusters. However, if migration to another cloud provider or different infrastructure is necessary, this type of container deployment offers the advantage of minimizing platform dependencies on the software, as the dependencies are isolated and containerized in well-identified locations. This allows for easy migration to another system without significant hassle.

3.3.3. Deployment

The deployment was carried out considering the chosen architectural pattern, the design and architectural decisions made, as well as the quality attributes considered. Therefore, it was decided to deploy the application using the cloud services offered by Amazon Web Services. As can be seen in the deployment diagram, there is an API Gateway that acts as the gateway to the application infrastructure.

This approach was implemented following a Legacy API Proxy architecture, which, as described by Shafqat et al. (2021), consists of "an innovative use case of the Amazon API Gateway and Lambda, which is what we call the Legacy API Proxy. Here, developers use API Gateway and Lambda to create a new API layer on top of legacy APIs and services to make them easier to use. The API Gateway is used to create a RESTful interface, and Lambda functions are used to translate requests and responses and convert data into formats that legacy services understand."

However, instead of using Lambda as a compute-as-a-back-end strategy, each of the developed microservices—users, monitoring, and media—are deployed in Docker containers managed by Amazon ECS on a serverless cluster of the AWS Fargate serverless service. This also ensures the maintainability and portability of the system. The proposed deployment architecture and its different nodes and services can be seen in Figure 5.

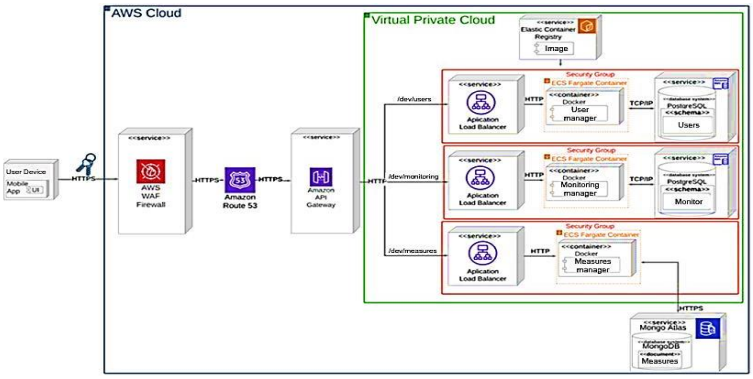


Fig. 5: Deployment Diagram – MAVATI.

Thus, these two approaches were combined to design the architecture. Therefore, client requests are routed to each service through the API Gateway, using a server-side discovery strategy. Or, in other words, each request is routed to the requested service from the server side. Each microservice runs in containers using Docker and is deployed serverlessly on an AWS Fargate cluster through AWS Elastic Container Service.

Regarding the architectural decisions made regarding security, malicious requests are filtered through the API Gateway using the Amazon WAF service, an application firewall managed by Amazon. All services are deployed in the same Amazon VPC, and for each microservice, the implemented infrastructure is in the same security group, ensuring secure and federated data isolation.

On the other hand, the MongoDB data node is outside these federated security groups, as it is self-managed by Mongo Atlas. However, the connection to this data source is made through the Mongo API (not through any driver), which ensures that information traffic is always encrypted and handled through a secure TSL socket and the HTTPS protocol.

Finally, it is important to mention that the choice was made to communicate with the microservices using the HTTP protocol. This approach is secure because requests never leave the established VPC environment and are only accessed through a policy associated with an AWS Identity and Access Management (IAM) role. This allows monitoring and measurement microservices, for example, to securely access information from the microservice that handles authentication (users).

4. Solution implementation and results

4.1. General approach

Based on the requirements gathered, the development and architectural decisions made, and the proposed design, a minimum viable product (MVP) was developed for both the physician and patient applications.

The same design approach was followed for both applications, using a bottom toolbar where users can access the content of each feature with expressive buttons and a minimalist yet intuitive design.

Usability tests were conducted to evaluate how intuitive a group of expectant mothers found the application. Additionally, a usability test was performed, and load tests were conducted on the designed architecture to verify its alignment with the quality attributes and non-functional requirements collected.

4.2. Prototype implementation

4.2.1. Prototype description

Patient Application: For patients, the focus was on monitoring, alerts, and recommendation features.

- Monitoring: In this regard, the monitoring feature first asks the user for permission to access health data. Once access is granted, the app's main view displays a beating heart animation to indicate that monitoring is active and working correctly. Figure 6 shows the flow of actions in the app: logging in, granting health data permissions, and the app's home screen, where the animation can be viewed.

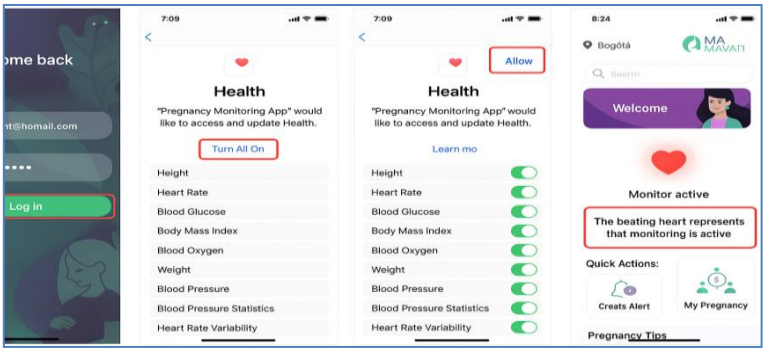


Fig. 6: Login and Monitoring Flow – MAVATI.

- Manual Alerts: Regarding the alert generation functionality, as shown in Figure 7, the list of alerts is first displayed, which includes both automatically generated and manual alerts.

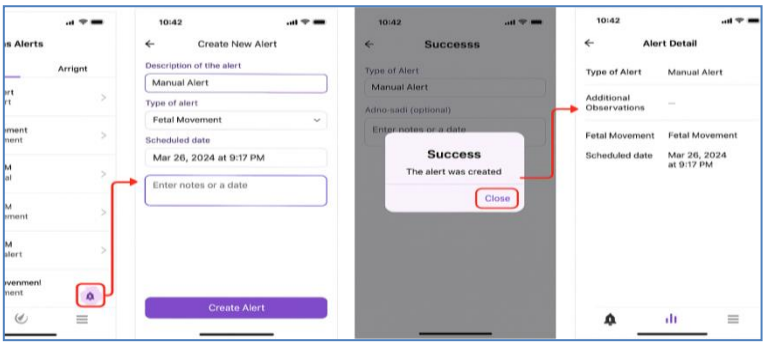


Fig. 7: Manual Alert Generation Flow – MAVATI.

Next, the floating button is pressed to create a new alert. The necessary data to generate the alert is then entered, and the alert is created. Finally, you can see how the new alert is added to the list.

- Automatic Alerts: To generate automatic alerts, first, the health app on the user's device, either Apple Health or Google Fit, which is connected to the smartwatch, records an atypical value for a biomedical measurement, as shown in Figure 8. For example, if the smartwatch detects an abnormally high heart rate, this data is recorded in the health app.

Once this outlier is detected, the device's health app automatically generates a new alert. This alert is added to the app's alert list, where all previously generated alerts can be viewed. The list displays both automatic alerts and those generated manually by the user.

Finally, selecting the new alert from the list allows you to access the alert details. These details describe the type of biomedical measurement that caused the alert, in this case, heart rate. The specific atypical value recorded is also displayed, for example, 200 beats per minute, and the date the alert was generated. This level of detail allows the patient to understand the reason for the alert and expect a consistent recommendation from their healthcare professional.

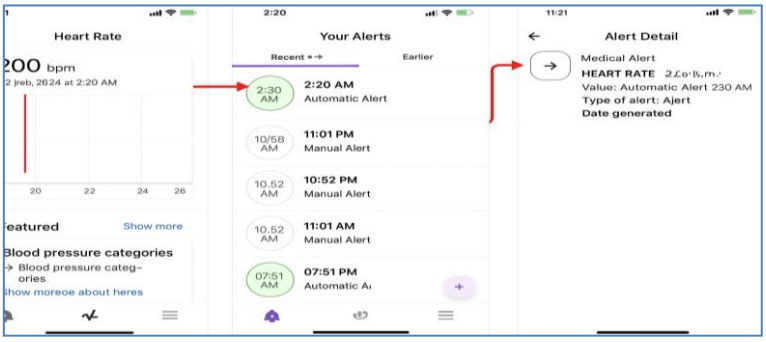


Fig. 8: Automatic Alert Generation Flow – MAVATI.

- Recommendations: Based on an alert generated by the patient, either manually or automatically, the physician can create a new recommendation. The new recommendation can be viewed in the list of recommendations, as shown in Figure 9. The patient can also view the recommendation details by selecting it from the list.

In the example shown in Figure 9, it is assumed that the physician has already created a recommendation. The user is informed that it is a new recommendation, and the recommendation details are displayed.

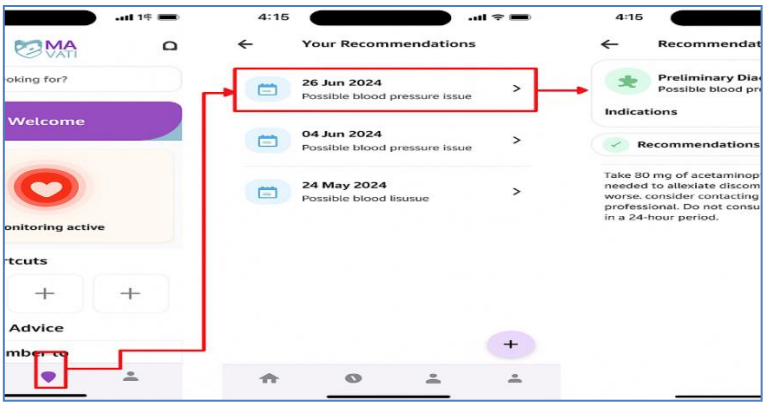


Fig. 9: Recommendation Access Flow – MAVATI.

Application for healthcare professionals: For physicians, the emphasis was placed on the monitoring functionality for each registered patient. The mobile application allows healthcare professionals to view their patients' biomedical measurements in real time, such as heart rate, blood pressure, and other key indicators previously explained.

In addition, the application facilitates the creation of personalized recommendations based on alerts generated by patients. These recommendations can respond to either automatic alerts, derived from atypical values detected by connected monitoring devices, or manual alerts, reported directly by patients.

- Patient Directory: To view the patients seen by the physician and available for monitoring, the physician first logs into the monitoring application, as shown in Figure 10. In the application's main menu, at the bottom, all the patients available for monitoring are listed.

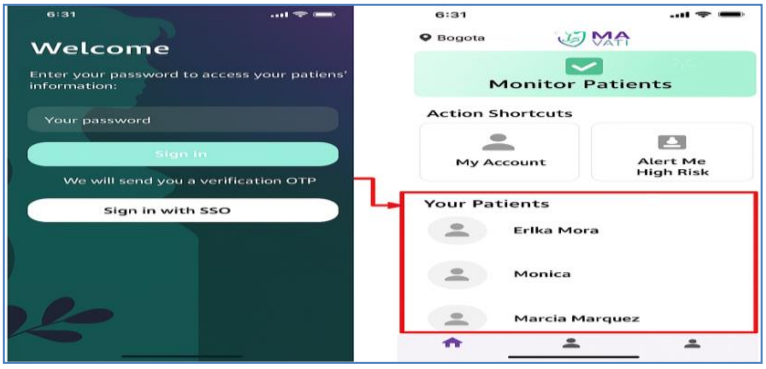


Fig. 10: Patient Directory Access Flow – MAVATI.

- Viewing Biomedical Measurements: To access a patient's biomedical variables, the healthcare professional enters the patient directory, selects the patient's name, and then accesses the biomedical variables view. The variability of the variable for the current day can then be viewed. Additionally, clicking on any of the dots allows you to view the variable's details.
- Generating alert-based recommendations: To create a new recommendation, the physician first goes to the general alert list, where they can view the alerts generated by all patients. Next, in the details of the selected alert, they select the option to create a recommendation. After reviewing the reasons behind the alert generation, the physician completes the necessary fields with relevant information and proceeds to generate a new recommendation based on the alert.

4.2.2. Usability testing

The objective of the usability testing was to verify the functionalities developed for physicians and patients, with special emphasis on the continuous monitoring functionality. To conduct these tests, biomedical data were collected continuously from a study subject for one week. This subject was fitted with an Apple Watch smartwatch, and the monitoring app was installed on their mobile phone, an iPhone 14 Pro Max, to simulate the continuous monitoring provided by the app. A commercial glucometer and a pulse oximeter were used to record blood glucose and blood oxygen levels, respectively. Since these values are not read from the smartwatch, they were manually entered into Apple Health, the mobile device's default health app.

4.2.3. Test results

After the subject's biomedical data was collected, it was viewed daily from the physicians' app, and various measurements were taken at different times of the day.
Figure 11 shows the graphical interface of the MAVATI mobile application for healthcare professionals. This image displays the daily evolution of measurements taken by a patient's smartwatch for a biomedical measurement. As can be seen in the image, the ordinate axis of the graph shows the value of the measurements taken, while the abscissa axis shows the time at which the measurement was taken.



Fig. 11: Graphical Display of Biomedical Measurements (Heart Rate) – MAVATI.

In this case, the biomedical measurement in question is heart rate. Furthermore, two peaks show measurements outside the normal range determined for heart rate: one of 200 bpm and the other of 170 bpm. These peaks of measurements outside the normal range are graphed with red dots in the UI, indicating the abnormality of the measurements. Measurements within the normal range are graphed with blue dots for each measurement taken. Each point on the graph can be selected by the user to view the measurement details.
Similarly, in Figure 12, the graph of blood sugar level measurements can be viewed from the doctor's application. To perform these measurements, the test subject used a commercial glucometer to measure their blood sugar level at different times of the day.

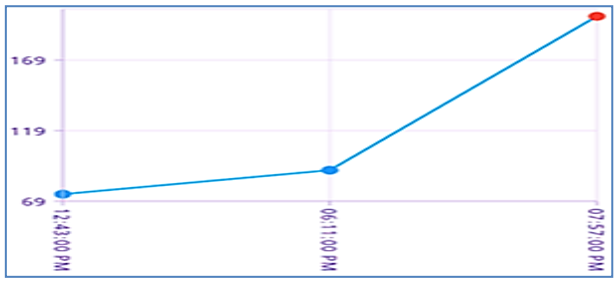


Fig. 12: Graphical Display of Biomedical Measurements (Blood Sugar Level) – MAVATI

These measurements were then manually entered into the patient's MAVATI application. Thus, as can be seen in the image, there is only one measurement outside the normal range, determined by a value of 200 mg/dL. This atypical value is highlighted in red at the measurement point.

During these tests, the patient's blood pressure and blood oxygen percentage were also recorded, as shown in Figures 13 and 14. The blood pressure recording graph presents a bar graph showing the systolic and diastolic blood pressure values. The red bars highlight atypical values in the measurements. Similarly, the graph representing the blood oxygen percentage identifies several red dots that indicate measurements outside of normal parameters. Additionally, as with the other biomedical measurement display options, selecting any of the indicative bars or dots allows the user to access the measurement details, including the exact time it was taken.

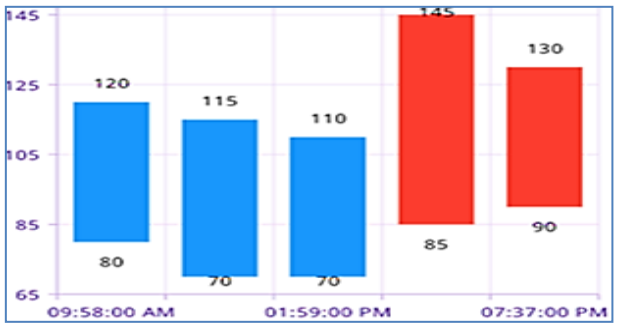


Fig. 13: Graphical Display of Biomedical Measurements (Blood Pressure) – MAVATI.

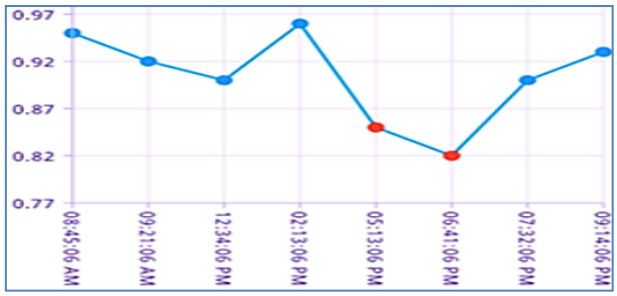


Fig. 14: Graphical Display of Biomedical Measurements (Blood Oxygen Percentage) – MAVATI.

4.3. Load testing implementation

4.3.1. Test description

To perform load testing on the previously designed architecture, the Apache JMeter tool was used. This tool allows load testing to analyze and measure the performance of a variety of web services. In the context of the monitoring application, the measurement microservice was chosen to test, retrieving measurements for doctors (GET) and creating patient measurements (POST), as this service is the one that should scale the most, as it handles the most HTTP requests from users and doctors continuously and concurrently. Furthermore, the measurement microservice communicates with the user microservice to manage user authentication. Therefore, this test also verifies the scalability of this other microservice.

Therefore, a test was designed in Apache JMeter, implementing a thread pool test for the GET and POST of the measurements. Thus, for each of these tests, the following were established:

- Number of threads: This represents the total number of virtual users (doctors/patients) simulating HTTP GET/POST requests for the measurements. A total of 500 threads were established.
- Rise period: This represents the time in seconds it will take to execute the full number of threads. A time of 500 seconds was established. So, for this case, the 500 threads will be created, and each request will be executed every 500 seconds. Therefore, each request will be created and sent every second:

Ramp-up period = 500 sec = 1 sec

Number of threads: 500

- Loop counter: Indicates the number of iterations of the test scenario. A value of 5 times was set.

Figure 15 shows the setup for the load tests:

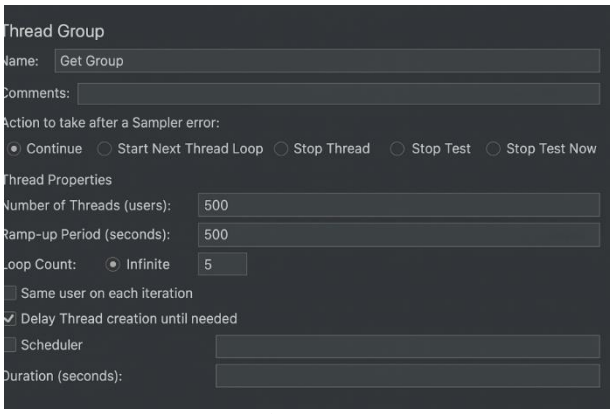


Fig. 15: Wire Group Structure.

4.3.2. Results

- GET method for biomedical measurements:

Figure 16 shows that a total of 1,043 samples (requests) were executed for this test. Additionally, the mean and median requests are 897 milliseconds and 940 milliseconds, respectively, suggesting that most requests have a response time close to these values. Similarly, the 90th percentile is 1,172 ms, the 95th percentile is 1,231 ms, and the 99th percentile is 1,373 ms. This indicates that 90% of requests have a response time less than 1,172 ms, 95% have a time less than 1,231 ms, and 99% have a time less than 1,373 ms.

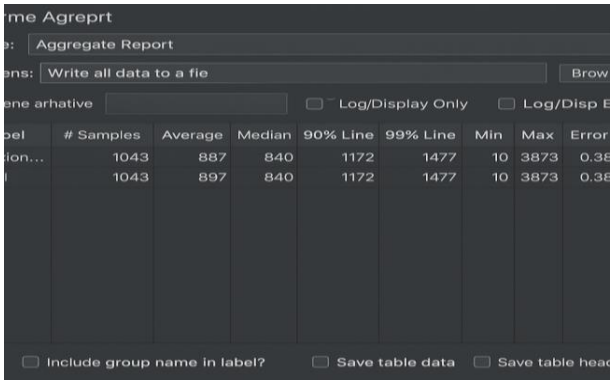


Fig. 16: Aggregate Report - GET Requests Measures.

In addition to this, it can be concluded from the image that the error rate is 0.38%, indicating that a small fraction of requests resulted in errors. This is a relatively low value and suggests that most requests were successful. Finally, the throughput is 5.0 requests per second. The data transfer rate is 584.80 KB/second, and the total amount of data sent per request is 3.87 KB.

From this image, it can also be concluded that the mean and median response times indicate moderate efficiency, with most requests responding in less than one second. The percentile values show that, although most requests are fast, some may have longer response times. Likewise, an error rate of 0.38% is low, suggesting that the system is quite reliable, with very few failed requests.

Finally, the throughput of 5 requests per second, along with a high data transfer rate, suggests that the system can handle a moderate request load with good performance, as in Figure 17.

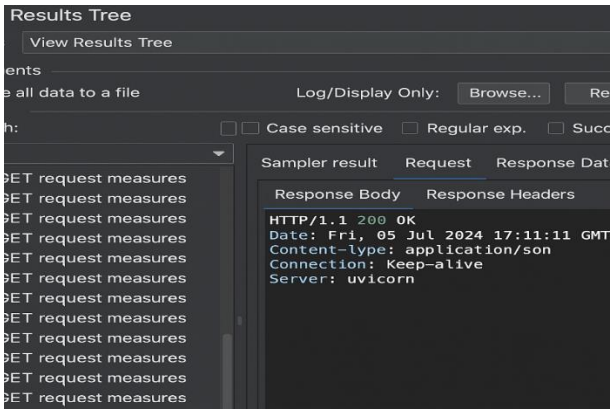


Fig. 17: Results Tree – GET Requests Measures.

Regarding response times, a graph with this information can be seen in Figure 18. As can be seen, response times decrease from an average of approximately 1000 ms to 600 ms after the system scales with the autoscaling strategy of the AWS Fargate serverless cluster. This demonstrates that the system can scale efficiently as the workload and response latency increase.

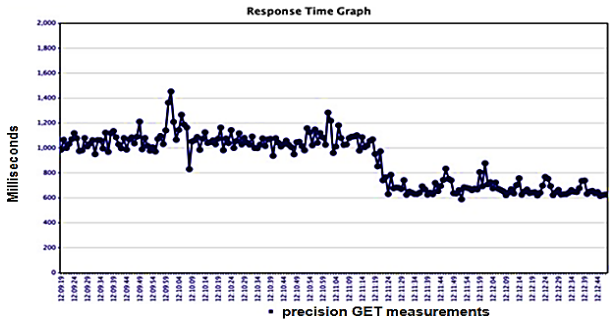


Fig. 18: Response Time Graph – GET Requests Measures.

Figure 19 shows the load test results graph for GET requests to biomedical measurements. In this image, each of the black dots represents a request. As can be seen, the mean response time is displayed on the blue line, the median on the purple line, the request throughput on the green line, and the standard deviation of request time on the red line.

At the beginning of the test, there is a large variation in the data and, therefore, in the measures of central tendency. This is because the system has not yet scaled and is suddenly being significantly stressed. However, as the system scales and begins to handle concurrent requests better, the mean, median, and throughput begin to converge toward specific values. This indicates that requests are being responded to uniformly, with some variation, but without random dispersion or a disproportionate increase in response times. The mean response time is 199 ms, indicating fast and consistent performance overall. Similarly, the median is 178 ms, suggesting that at least half of the requests have a response time below this value, reinforcing the observed consistency. Additionally, the standard deviation of 52 ms indicates some variability in response times, although not extremely high.

At the start of the test, response times appear to be higher, but they quickly decrease and stabilize. This may indicate an initial warm-up period for the system before reaching optimal performance. Once stabilized, response times are consistent and low, which is a positive indication that the system can handle the load efficiently.

The graph shows that after the initial phase, there is consistency in response times, with few noticeable deviations. This indicates that the system can maintain stable performance under the applied load. The standard deviation indicates that the data dispersion increases at the beginning of the test, as the data dispersion relative to its mean also increases. However, this increase is asymptotic, or decreasing, indicating that as the system scales, requests are handled similarly and the system responds similarly, stabilizing the standard deviation at a specific value. This means that these data are not disproportionately dispersed, indicating continued system performance.

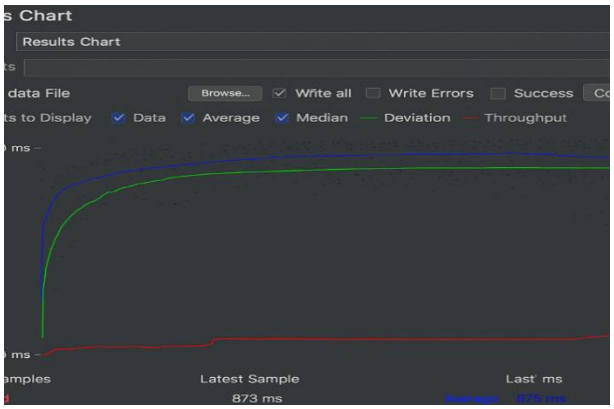


Fig. 19: Results Graph - GET Requests Measures.

- POST method for biomedical requests:

To perform load testing for the POST requests for the biomedical measurements, the biomedical measurement model shown in Figure 20 was used.

```
[
  {
    "value": "NumericHealthValue - numericValue: 0.93",
    "patient_id": "193748204",
    "measurement_type": "BLOOD_OXYGEN_PERCENTAGE",
    "timestamp": "2024-07-05T21:14:06.616"
  }
]
```

Fig. 20: Biomedical Measurement Structure - POST Requests Measurements.

The aggregate report of the load tests performed on the application architecture can be seen in Table 1. As can be seen, 263 ms, 286 ms, and 407 ms respectively, indicate the response times below which 90%, 95%, and 99% of the requests were completed. Additionally, a very low error rate of 0.07% can be seen, suggesting that virtually all requests were processed correctly, except for a small fraction that failed.

Table 1: Performance Comparison Before and After SMOTE Application (Test 2 - Precision)

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	% Error	Throughput	KB/sec	Sent KB/sec
Request	1525	199	178	263	286	407	32	725	0.07%	5.0/sec	1.59	4.17
Total	1525	199	178	263	286	407	32					

This can also be seen in Figure 21, which shows the results tree of the requests made in the tests. The vast majority of them were successful. This indicates that the system is capable of scaling effectively without affecting the success of most requests or the response time.

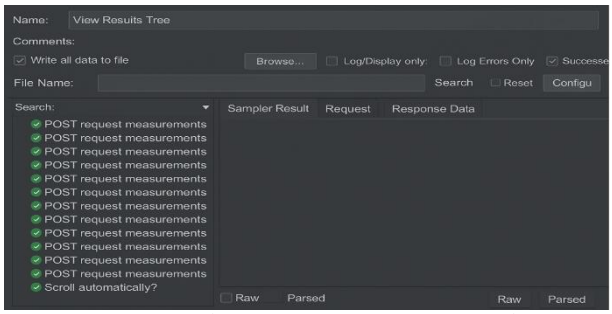


Fig. 21: Results Tree - POST Requests Measured.

Similarly, Figure 22 presents the response time graph, which illustrates how the response time of POST requests varied during the test execution. Most of the measurements are concentrated around 200 ms, reflecting consistent performance. However, two pronounced latency peaks are identified: one around 12:55:30 and another around 12:56:00, with values reaching or exceeding 350 ms. These peaks are associated with the time required for AWS Fargate to scale the service, deploy the necessary infrastructure.
Thus, aside from the peaks, the graph shows stable behavior, suggesting that the system handled requests with a predictable response time.

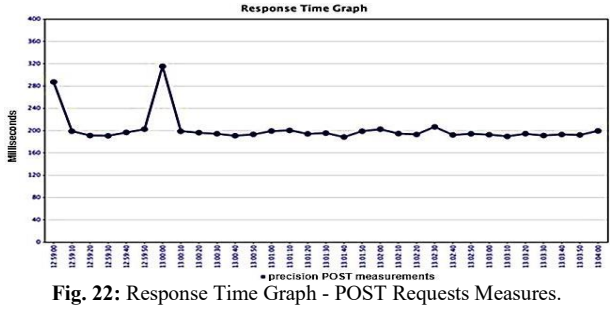


Fig. 22: Response Time Graph - POST Requests Measures.

Similarly to the load test for GET requests for biomedical measurements, in the load tests for POST requests, Figure 23 shows that, as in the previous test, the data and the measures of central tendency—mean, median, and standard deviation—have a high degree of variation at the start of the test. However, when the system scales, the data show much less dispersion and more uniform variability; therefore, the values of these measures converge to specific values. In the case of the mean, the mean is 199 ms, and the mean is 178 ms. Furthermore, the deviation value hovers around 52 ms, which is not a very high value, indicating that most of the data do not disperse significantly from the mean.

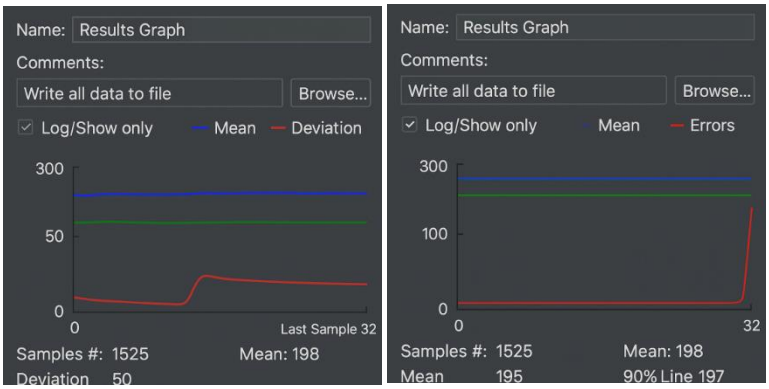


Fig. 23: Results Graph - POST Requests Measures.

In summary, the graph shows that, like the previous test, after an initial adjustment period, the system can efficiently handle the workload, providing fast and consistent response times. The measures of central tendency and dispersion converge to specific values, demonstrating the system's ability to scale and maintain stable performance under load.
Clinical trials are planned as the next step to validate the efficacy of MAVATI in healthcare settings. These trials involve a variety of patients and healthcare providers in a variety of clinical settings. The aim is to evaluate the accuracy of real-time monitoring, the reliability of alerts (manual and automated), and the effectiveness of doctor-patient communication within the program. Key metrics include clinical response times to alerts, adherence to user recommendations, and patient outcomes over time. These trials provide key evidence to assess the practical utility, scalability, and integration potential of MAWATI into standard medical practice (Shafqat et al., 2021).

4.4. Potential limitations

MAWATI faces smartwatch compatibility constraints, which limit its use on multiple devices; lack of internet connectivity in rural areas can lead to productivity declines; and the cost of the required equipment may limit accessibility to low-income users, impacting overall scalability and inclusivity.

4.5. Future developments

To enhance the capabilities of MAWATI, future developments could include AI-based risk prediction models that analyze biomedical data and predict health outcomes. Improved cross-platform compatibility enables wider adoption of wearables and mobile devices to expand their use. Additionally, MAVATI's integration with India's RIAS (Real-time Intelligent Analytics System) framework enables nationwide healthcare data collection, early detection, and resource allocation, especially in healthcare settings.

5. Ethical considerations

The implementation of MAVATI raises critical ethical concerns related to patient consent, data privacy, and biases inherent in IoT-based healthcare systems. Drawing on the principles outlined by Dimitrov (2016) in their discussion of data ethics, MAVATI must ensure informed consent is explicitly obtained from all users before any data is collected or processed. Furthermore, stringent safeguards must be in place to protect sensitive health information, aligning with ethical frameworks and regulatory standards such as GDPR and HIPAA. Another key issue is algorithmic and IoT bias, where the accuracy of data and subsequent clinical decisions may be skewed due to device limitations or demographic underrepresentation. Addressing these concerns requires ongoing audits, transparent data governance, and inclusive testing across diverse populations to ensure fairness and accountability in MAVATI's deployment (Somathilake et al., 2022).

5.1. Social impact and scalability

MAWATI plays a critical role in achieving Sustainable Development Goal 3 (Good Health and Well-Being) by enabling early detection and ongoing monitoring of maternal health risks, thereby helping to reduce maternal mortality. This alert and counseling system allows patients and healthcare providers to take timely action, especially in high-risk pregnancies. Designed with scalability in mind, the MAWATI architecture supports deployment in resource-constrained environments using ubiquitous mobile technologies and expensive monitoring devices. It aligns with the journal's mission to promote innovative, inclusive, and impactful health technologies that address global health disparities (Somathilake et al., 2022).

6. Conclusions and recommendations

It was possible to identify the necessary functional and non-functional requirements through the specialized medical advice received. Based on this, an architecture was designed that met these requirements, and a minimum viable product was developed for both healthcare professionals and patients. This product provides the necessary functionalities to enable close monitoring of patients by their physicians to improve their perinatal health. The designed and developed architecture was subjected to load testing to verify its scalability, confirming that it met the quality attributes for the application's availability and performance. Intensive clinical trials of the monitoring app, involving physicians and patients, are needed in real-life settings to evaluate the app's effectiveness and confirm the extent to which it improves maternal health and reduces risks for pregnant women. Furthermore, data collected from real patients in these trials, or during the app's production phase, could be used at a later stage to design data analysis and machine learning algorithms for conducting medical studies focused on scientific and/or medical research.

References

[1] Shafqat, W.; Malik, S.; Lee, K.-T.; Kim, D.-H. PSO-based optimized ensemble learning and feature selection approach for efficient energy forecast. Electronics 2021, 10, 2188. https://doi.org/10.3390/electronics10182188.

[2] Khaire, U.M.; Dhanalakshmi, R. Stability of Feature Selection Algorithm: A Review. J. King Saud Univ. Comput. Inf. Sci. 2022, 34, 1060–1073. https://doi.org/10.1016/j.jksuci.2019.06.012.

[3] Rostami, M.; Berahmand, K.; Nasiri, E.; Forouzandeh, S. Review of swarm intelligence-based feature selection methods. Eng. Appl. Artif. Intell. 2021, 100, 104210. https://doi.org/10.1016/j.engappai.2021.104210.

[4] Choi, R.Y.; Coyner, A.S.; Kalpathy-Cramer, J.; Chiang, M.F.; Campbell, J.P. Introduction to machine learning, neural networks, and deep learning. Transl. Vis. Sci. Technol. 2020, 9, 14.

[5] Alam, T.M.; Shaukat, K.; Hameed, I.A.; Luo, S.; Sarwar, M.U.; Shabbir, S.; Li, J.; Khushi, M. An Investigation of Credit Card Default Prediction in the Imbalanced Datasets. IEEE Access 2020, 8, 201173–201198. https://doi.org/10.1109/ACCESS.2020.3033784.

[6] Allahem, H.; Sampalli, S. Automated uterine contractions pattern detection framework to monitor pregnant women with a high risk of premature labour. Inform. Med. Unlocked 2020, 20, 100404. https://doi.org/10.1016/j.imu.2020.100404.

[7] Saarikko, J.; Niela-Vilén, H.; Ekholm, E.; Hamari, L.; Azimi, I.; Liljeberg, P.; Rahmani, A.M.; Löyttyniemi, E.; Axelin, A. Continuous 7-month internet of things-based monitoring of health parameters of pregnant and postpartum women: Prospective observational feasibility study. JMIR Form. Res. 2020, 4, e12417. https://doi.org/10.2196/12417.

[8] Machorro-Cano, I.; Olmedo-Aguirre, J.O.; Alor-Hernández, G.; Rodríguez-Mazahua, L.; Sánchez-Morales, L.N.; Pérez-Castro, N. Cloud-Based Platforms for Health Monitoring: A Review. Informatics 2024, 11, 2. https://doi.org/10.3390/informatics11010002.

[9] Al Khatib, I.; Shamayleh, A.; Ndiaye, M. Healthcare and the internet of medical things: Applications, trends, key challenges, and proposed resolutions. Informatics 2024, 11, 47. https://doi.org/10.3390/informatics11030047.

[10] Gong, L.; Xiao, Z.; Xu, L.; Ding, Y.; Zou, Z.; Zheng, L. An IoT-based wearable labor progress monitoring system for remote evaluation of admission time to hospital. IEEE J. Biomed. Health Inform. 2023, 27, 3037–3048. https://doi.org/10.1109/JBHI.2023.3264251.

[11] Pilosof, N.P.; Barrett, M.; Oborn, E.; Barkai, G.; Pessach, I.M.; Zimlichman, E. Inpatient telemedicine and new models of care during COVID-19: Hospital design strategies to enhance patient and staff safety. Int. J. Environ. Res. Public Health 2021, 18, 8391. https://doi.org/10.3390/ijerph18168391.

[12] Xu, J.; Zhang, Y.; Wang, L.; Chen, X. Fetal movement detection by wearable accelerometer duo based on machine learning. IEEE Sens. J. 2022, 22, 11526–11534. https://doi.org/10.1109/JSEN.2022.3172451.

[13] Qin, M.; Xu, Y.; Liang, Y.; Sun, T. A wearable fetal movement detection system for pregnant women. Front. Med. 2023, 10, 1160373. https://doi.org/10.3389/fmed.2023.1160373.

[14] Somathilake, E.; Delay, U.H.; Senanayaka, J.B.; Gunarathne, S.L.; Godaliyadda, R.I.; Ekanayake, M.P.; Wijayakulasooriya, J.; Rathnayake, C. Assessment of fetal and maternal well-being during pregnancy using passive wearable inertial sensor. IEEE Trans. Instrum. Meas. 2022, 71, 4005111. https://doi.org/10.1109/TIM.2022.3175041.

[15] Georgieva-Tsaneva, G.; Cheshmedzhiev, K.; Tsanev, Y.-A.; Dechev, M.; Popovska, E. Healthcare Monitoring Using an Internet of Things-Based Cardio System. IoT 2025, 6, 10. https://doi.org/10.3390/iot6010010.

[16] Ouypornkochagorn, T.; Dankul, W.; Ratanasathien, L. Fetal movement detection with a wearable acoustic device. IEEE Sens. J. 2023, 23, 29357–29365. https://doi.org/10.1109/JSEN.2023.3326479.
[17] Ghosh, A.K.; Catelli, D.S.; Wilson, S.; Nowlan, N.C.; Vaidyanathan, R. Multi-modal detection of fetal movements using a wearable monitor. Inf. Fusion 2024, 103, 102124. https://doi.org/10.1016/j.inffus.2023.102124.
[18] Ouypornkochagorn, T.; Ratanasathien, L.; Dankul, W. A Portable Acoustic System for Fetal Movement Detection at Home. IEEE Sens. J. 2025, 25, 1478–1486. https://doi.org/10.1109/JSEN.2024.3486710.
[19] L. Floridi and M. Taddeo, "What is data ethics?" Philos. Trans. R. Soc. A Math. Phys. Eng. Sci., vol. 376, no. 2133, p. 20180081, 2018. https://doi.org/10.1098/rsta.2018.0081.
[20] C. Wang, A.L.V. Johansson, C. Nyberg, A. Pareek, C. Almqvist, S. Hernandez-Diaz, and A.S. Oberg, "Prediction of Pregnancy-Related Complications in Women Undergoing Assisted Reproduction, Using Machine Learning Methods," Fertil. Steril., vol. 122, pp. 95–105, 2024. https://doi.org/10.1016/j.fertnstert.2024.02.024.
[21] M.A. Rahman, R.M. Noor, S. Mallik, N.K. Santa, S. Deb, and A. Pathak, "Classification of Health Risk Levels for Pregnant Women Using Support Vector Machine (SVM) Algorithm," IOSR J. Comput. Eng., vol. 26, pp. 7–17, 2024.
[22] A.O. Khadidos, F. Saleem, S. Selvarajan, Z. Ullah, and A.O. Khadidos, "Ensemble Machine Learning Framework for Predicting Maternal Health Risk During Pregnancy," Sci. Rep., vol. 14, p. 21483, 2024. https://doi.org/10.1038/s41598-024-71934-x.
[23] D.V. Dimitrov, "Medical Internet of Things and Big Data in Healthcare," Healthc. Inform. Res., vol. 22, no. 3, pp. 156–163, Jul. 2016, https://doi.org/10.4258/hir.2016.22.3.156.
[24] Saarikko, J.; Niela-Vilen, H.; Ekholm, E.; Hamari, L.; Azimi, I.; Liljeberg, P.; Rahmani, A.M.; Löyttyniemi, E.; Axelin, A. Continuous 7-month Internet of Things–based monitoring of health parameters of pregnant and postpartum women: Prospective observational feasibility study. JMIR Form. Res. 2020, 4, e12417. https://doi.org/10.2196/12417.