PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 01/2023

Vital Sign Project: Remote Monitoring of Patients for Healthcare Workers

Andrew Diniz da Costa
Carlos José Pereira de Lucena
Antonio Carlos Bechimol Barbosa
Carlos André Pereira de Lucena
Polyana Sampaio Ramos Barboza

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 01/2023 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena January, 2023

Vital Sign Project: Remote Monitoring of Patients for Healthcare Workers *

Andrew Diniz da Costa, Carlos José Pereira de Lucena,
Antonio Carlos Bechimol Barbosa, Carlos André Pereira de Lucena,
Polyana Sampaio Ramos Barboza

andrew@les.inf.puc-rio.br, lucena@inf.puc-rio.br, antonio.benchimol@les.inf.puc-rio.br,
andre.lucena@les.inf.puc-rio.br, pbarboza@inf.puc-rio.br

Abstract. The healthcare area has presented great opportunities for scientific research in Information Technology. One of the reasons that highlighted the relevance of healthcare area was the pandemic reality we have dealt since the beginning of 2020. As impact, the leading app stores registered significant growth in the number of available eHealth apps. Aiming to design and develop innovative technological solutions to help leverage the healthcare sector to a technological development level compatible with its importance and criticality levels, as initial result the Software Engineering Lab, this whitepaper presents a solution developed to monitor remotely vital sign data from patients who are at hospitals. Details of the architecture proposed and how the iOS app developed to healthcare workers present vital sign data of patients are explained in detail.

Keywords: Healthcare, health, mobile, iOS development, and vital signs.

Resumo: A área da saúde tem apresentado grandes oportunidades de pesquisa científica em Tecnologia da Informação. Um dos motivos que destacaram a relevância da área de saúde foi a realidade pandêmica que enfrentamos desde o início de 2020. Como impacto, as principais lojas de aplicativos registraram crescimento significativo no número de aplicativos de eHealth disponíveis. Com o objetivo de projetar e desenvolver soluções tecnológicas inovadoras para ajudar a alavancar o setor de saúde a um nível de desenvolvimento tecnológico compatível com seus níveis de importância e criticidade, como resultados iniciais, o Laboratório de Engenharia de Software, apresenta neste whitepaper uma solução desenvolvida para monitorar remotamente dados de sinais vitais de pacientes que estão em hospitais. Detalhes da arquitetura proposta e como o aplicativo iOS desenvolvido para profissionais de saúde apresenta dados de sinais vitais de pacientes são explicados em detalhes.

Palavras-chave: Cuidados de saúde, saúde, dispositivos móveis, desenvolvimento iOS, siinais vitais.

In charge of publications:

PUC-Rio Departamento de Informática - Publicações
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: publicacoes@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

The healthcare area is emerging as a fertile ground for scientific research in Information Technology. Research activities in the area allow us to address several issues to promote technological development. Regarding mobile device apps, the leading app stores registered significant growth in the number of available eHealth apps. Suring the last measured period, there were 54,546 healthcare and medical apps available on the Google Play [Statistica a, 2022] and 41,517 iOS healthcare ands medical apps available on the Apple Store [Statistica b, 2021]. In 2020, the number of mHealth apps available to iOS users kept growing, and in the first quarter of 2021 reached its peak of almost 54,000 apps. These growths find one of its causes in the pandemic reality we face since the beginning of 2020. We managed to design and develop innovative technological solutions to help leverage the healthcare sector to a technological development level compatible with its importance and criticality levels.

Several current operational problems of medical care and hospitals can be solved with technological support. The application of computational solutions to hospital activities has the capacity to transform the present reality, trough, for instance, the improvement of the work processes. As examples of improvements brought by the use of innovative technological solutions, we can name: i) Change in the way the physician-patient relationship occurs, due to remote patient monitoring possibilities [Fernandes and Lucena, 2015]; ii) Ease of information access and sharing among the medical team and the patient relatives [Fernandes et al, 2016]; iii) More mobility for patients, whose health status can be monitored from home or work, without being physically restricted to hospital facilities; iv) Possibility of collaborative work between the local team and external professionals; It allows a second opinion about patients' diagnoses and treatments, since patient information is already in a distributed database; v) Possibility of automatic processes, such as vital patient data collection, by using sensors; vi) Remote and real-time monitoring of patient health conditions; vii) Alerts to healthcare professionals in emergency situations; viii) Decrease in elapsed time for detection of anomalies in the vital signs of monitored patients, by using software agents; in this context, software agents consist of computational entities that perform activities in response to emergency situations. Hence, investments in technology can permit reaching results that extrapolate the operational level. Since operational problems related to medical routines can be solved by solutions such as the ones shown above it may allow hospital managers to focus on more relevant, strategic level, questions.

In the literature there are a set of solutions related to remote monitoring of patients offered, such as, My Vitals [My Vitals, 2021], VIVA Link [VIVA Link, 2021], and Binah.ai [Binah.ai, 2021]. Although being interesting proposals, they do not lead with some of following points: (i) applying correctly native resources offered by platform used to provide better experiences to the users, (ii) using software agents that contributes to the inclusion of intelligence, (iii) allowing the personalization of important features to monitoring, such as patient data visualisation, and (iv) allowing the easy addition of new features. Aiming to contribute to the development of eHealth solutions, we propose a solution that looks for dealing the points mentioned allowing healthcare workers to monitor remotely vital sign data from patients.

This whitepaper presents a solution of remote monitoring that has been developed. It is structured as follows. Section 2 shows a theoretical background is presented. Section 3 presents an overview of the solution that we have developed. Section 4 explains details how vital sign data are collected from monitors used. Section 5 shows which

web services has been developed to the solution. Section 5 explains which web services were created to the solution, and shows some screens of the app created. Section 6 presents screens of the app developed, and Section 7 presents conclusions and future works.

2 Theoretical Background

In this section important concepts considered in the work are presented.

2.1 Internet of Things (IoT)

IoT is a new field within Computer Science that has grown quickly in recent years. Kevin Ashton introduced the term Internet of Things in 1999 [Ashton, 2009]. One can define IoT as a global network of smart devices that can sense and interact with their environment for communication with users and other things and systems alike.

2.2 Software Agents

An agent [Wooldridge, 1999] is an element of a computational system that is situated in some environment where it can perform autonomous actions, in order to reach the goals that are delegated to it. Agents contain properties such as autonomy and learning. The learning process is related to its capability of learning from its experience. Autonomy has been acknowledged as a key characteristic for its actuation that it uses to decide as to act to satisfy its goals [Wooldridge, 2009]. In this context, autonomy means the possibility of actuation without human or other systems intervention, although the set of possible actions should be previously defined.

Although agents control its behavior and internal state, they do not have entire control of the environment. Agents contain a set of actions that can carry out tasks, whose execution can result in changes in the environment. For this reason, one can consider that an agent can have partial control over its environment so that it can influence it, depending on the action it decides perform [Norvig, 2013].

2.3 General Concepts about Software Frameworks

Frameworks are tools used to generate applications related to a specific domain; that is, to handle a family of related problems [Markiewicz and Lucena, 2001]. The choice of using existing frameworks or developing new generators of applications is justified by the ability of this kind of tool to offer design and code reuse. This fact allows a boost in software development productivity and shorter time-to-market, compared to traditional approaches.

Frameworks contains fixed and flexible points known as frozen spots and hot spots, respectively. Hot spots are extension points that allow developers to create an application from the framework instantiation process. In this case, developers should create specific application code to each hot spot, through the implementation of abstract classes and methods defined in the framework. Frozen spots, in turn, consist of the framework's kernel, corresponding to its fixed parts, previously implemented and hard to change, which will call one or more of the application's hot spots and will be present in each framework's instance [Norvig, 2013]..

Frameworks' building activities comprise three main steps: 1-Domain analysis; 2-Design; 3-Instantiation process. The domain analysis step includes requisites elicitation activities, along with hot and frozen spots' definitions. The design step is responsible for drawing the hot and frozen spots through a modeling language, by using UML [UML, 2016] diagrams, for example, to show the framework's extensible and flexible points. Design patterns [Gamma et al, 1995] are also used in this phase. The instantiation phase corresponds to the application generation phase, through hot spot implementation [Markiewicz and Lucena, 2001].

3 Architecture Overview

The approach, which has been developed, considers two user types: healthcare workers and patients. As initial scope we decided to develop a multilanguage solution to the iOS platform. Currently, we have in English and Portuguese.

Presently, we are collecting patients data from vital signal monitors, Apple watch, and more sensors are being added. Considering that health data are being handled, we have planned in our scope the application of data security to all information used in the solution.

Aiming to illustrate an overview of the solution, Figure 1 shows vital sign monitors used: CM100 and CM120 connected to a computer. Both monitors are Philips and they are produced in Brazi using the HL7 standard. Besides, there is a hub connecting the monitors and the computer, being possible to include more monitors if desired.

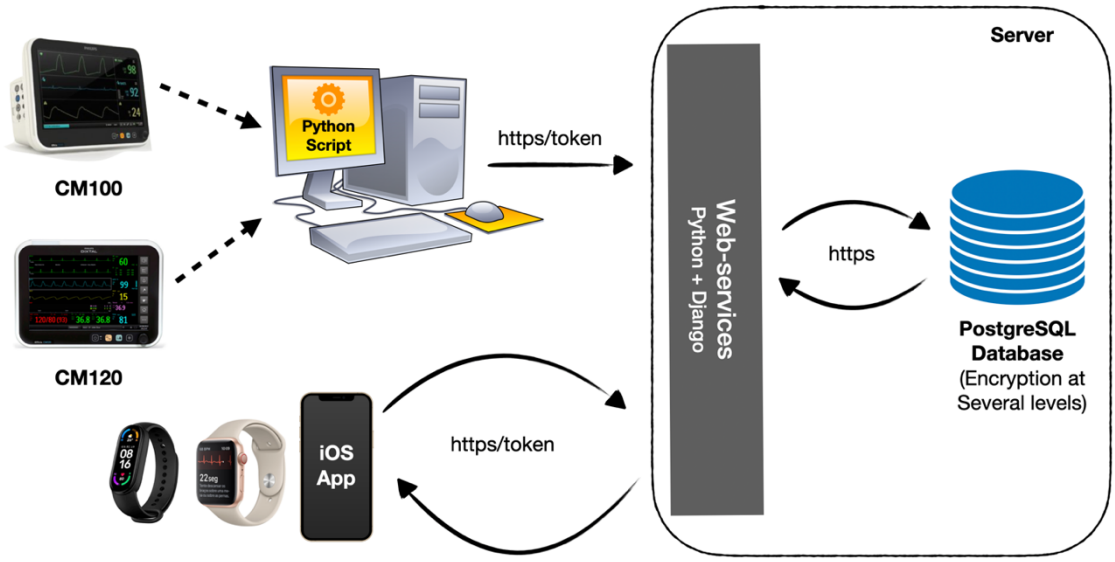

Figure 1. Architecture

In the computer connected to the monitors, there is a Python script that verifies per minute if data was sent from them. It uses socket TCP/IP to make that verification. When there are new data collected, they are saved into a PostgreSQL database from web services developed. PostgreSQL [PostgreSQL, 2021] offers encryption at several levels, and provides flexibility in protecting data. This tool is secure sensitive data such as, medical records or financial transactions. Thus, it is possible, for example, to encrypt all tables of a database, a subset of tables or even some columns.

The web service layer used to connect the Python script to the database (see Figure 1) is also used by iOS app to connect with the same database. Such layer was developed using the Python language and the web framework Django [Django, 2021].

Aiming to deal with data security, Django already provides a flexible password storage system and uses a specific algorithm (PBKDF2), which is a password stretching mechanism recommended by NIST [NIST, 2021]. Besides, Django also uses token to allow access to webservices, supports cryptography integrated with PostgreSQL, and we are considering in our scope to use https communication.

Currently, the webservices and database developed are at a remote server. Such server can be easily migrated to another computer. The basic configuration of that is the following.

1. **Operating system:** Linux
2. **RAM Memory:** 1 Gigabyte
3. **SSD Memory:** 25 Gigabytes
4. **Server:** Apache 2.4.4.1
5. **Database:** PostgreSQL 12.7
6. **Service Provider:** Digital Ocean

Figure 2, presents a class diagram with the main classes of the solution that has been developed. To represent the possible users of the app, a User class was created. Besides, to represent Healthcare workers (Doctors or Nurses) and Patients, one class per type of user was offered.

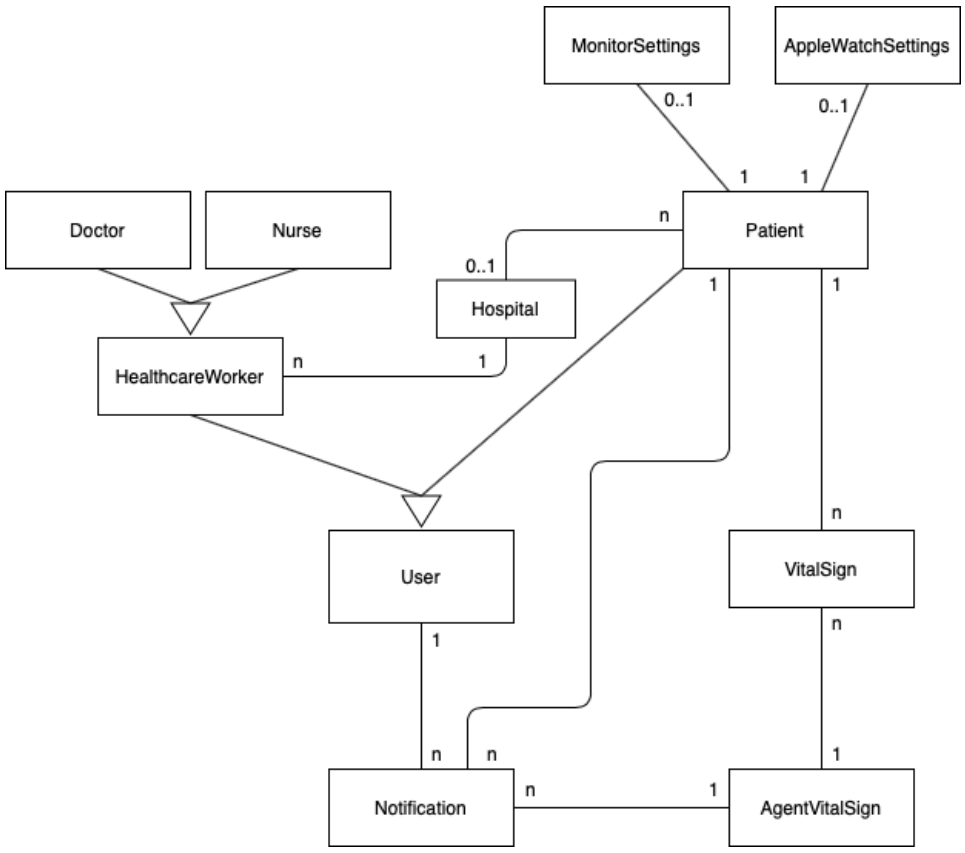

Figure 2. Class diagram of the proposed solution.

Each patient has a set of vital sign data being monitored. When some anomaly happens from the data collected of a patient, a software agent (AgentVitalSign class) sends a notification (Notification class) to users that can deal with that issue.

All healthcare workers of the system are related to some hospital, while a patient not necessarily. In the current version of the solution, patients can be monitored from monitors, and from other devices (e.g. apple watch), which do not require them to be at a hospital. In addition, each patient monitored can have a set of visible data in the app to some healthcare worker to analyze. The indication which data types of patients are visible per healthcare are represented by MonitorSettings and AppleWatchSettings classes. The first class represents the data types of a vital signs monitor, while the second class represents data related to an apple watch from a patient.

4 Python Script to Collect Data from Monitors

In this section details on how a Python script is able to collect vital signs data from Philips monitors are presented. In order to make that collection, a script developed from the Python language is responsible for receiving data exported in HL7 protocol by monitors every 60 seconds through TCP/IP socket on RJ45 network interface. If the monitor has a wireless network interface, the script can also be used as long as the monitor and the server machine are properly identified. The source code is available in a (private) repository on GitHub.

The script has handled data exported by monitors using the HL7apy library [HL7apy, 2021]. It is important to mention that data exported via RJ45 only occurs with minimal patient identification. The most common identifying attribute is the MRN (also known as the medical record number). However, it is possible to define another identifier from the monitor admin panel. Besides, other attributes referring to a patient, such as name, date of birth, can be configured as mandatory or optional to be exported in HL7 messages.

Below, in subsection 4.1, a step by step explaining how to configure a Philips monitor to collect data via Python script is presented. Next, subsection 4.2 details how to execute the Python script is explained. For last, subsection 4.3 mentions which methods were created in that script developed.

4.1 Setting up data export

Below there is a step by step of what it is necessary to collect data exported from a monitor via Python script. More information on exporting data from Philips monitors can be found at "Documentation for the Efficia Monitor" manual offered with the CD that came with each monitor.

Step1: Select the gear icon to access the settings menu as illustrated in Figure 4.

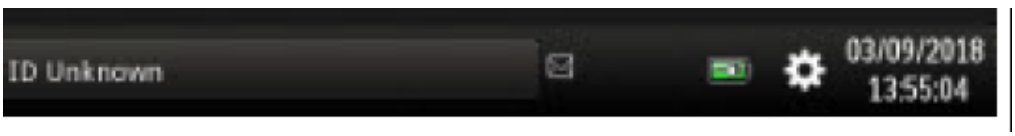

Figure 4. Accessing settings menu.

Step 2: Select the Admin tab (see Figure 5).

Figure 5. Adm tab of a Philips monitor.

Step 3: Select the Data Export menu. Next, activate the channel to be used (WLAN, serial output or LAN).

Step 4: If the Data Export of item is "LAN" (it is used in the current scenario), go back to the Admin tab, select LAN, next enter the IP and port data of the server machine in the network.

4.2 Executing script

To execute the Python script developed in a computer (server), it is necessary to install their main dependencies. They can be installed using Python's pip dependency manager from the command below.

pip install requests hl7apy

Next, see how the server that will run the script is identified on the network to start receiving data and configure it in the data export menu illustrated previously. By default, the listening port is '22222'. This can be modified in the 'SERVER_PORT' variable in the config.py file.

After receiving the raw data in HL7/bytes, the data is parsed to create and populate a json/dict. Next, the data are sent to a web service defined from the variable 'resource_post', located in the config.py file (it is necessary to request a valid token for requests). Thus, data are sent via HTTP POST method using Python's requests library as soon as json/dict is populated.

After that, execute the listener_and_parser.py file with the Python interpreter from the command line below to start listening and sending data.

python <project_folder >/listener_and_parser.py

4.3 Methods Created

Below the signatures of the created methods in the listener_and_parser.py file are presented with a brief explanation.

1. **build_ack_response(ack_code, msg_ctrl_id):**
   It creates ACK message (ACKnowledgment) for the messages received from the monitor.

2. **listener_socket(host, port):**
   It makes a TCP/IP socket listen in infinite loop mode. Thus, this method receives the data, and sends the appropriate ACK message to the monitor.

3. **format_date(date):**
   It formats (readable format) the exported dates in the HL7 message.

4. **replace_all(text, dic):**
   It uses a dictionary to replace characters that can make parsing the HL7 message difficult.

5. **check_if_alarm(obr_segment):**
   It checks whether the HL7 message sent by the monitor is an ORU alarm message. If it is, a special handling is required for the monitor to go on sending observation ORU messages normally.

6. **parser_hl7_message(message):**
   It uses HL7apy lib to parse and extract data from HL7 message. Only patient data and data related to sensors connected to the monitor (and the patient) are exported. For instance, if only an oximetry sensor is connected, then only perfusion, pulse and spO2 is available in the analysis result. There is a loop to go through each HL7 segment that has the monitored attributes and builds a list of dictionaries.

7. **send_request_values_dbservice(resource, data):**
   It sends a JSON data to a service via HTTP POST as soon as parsing is complete.

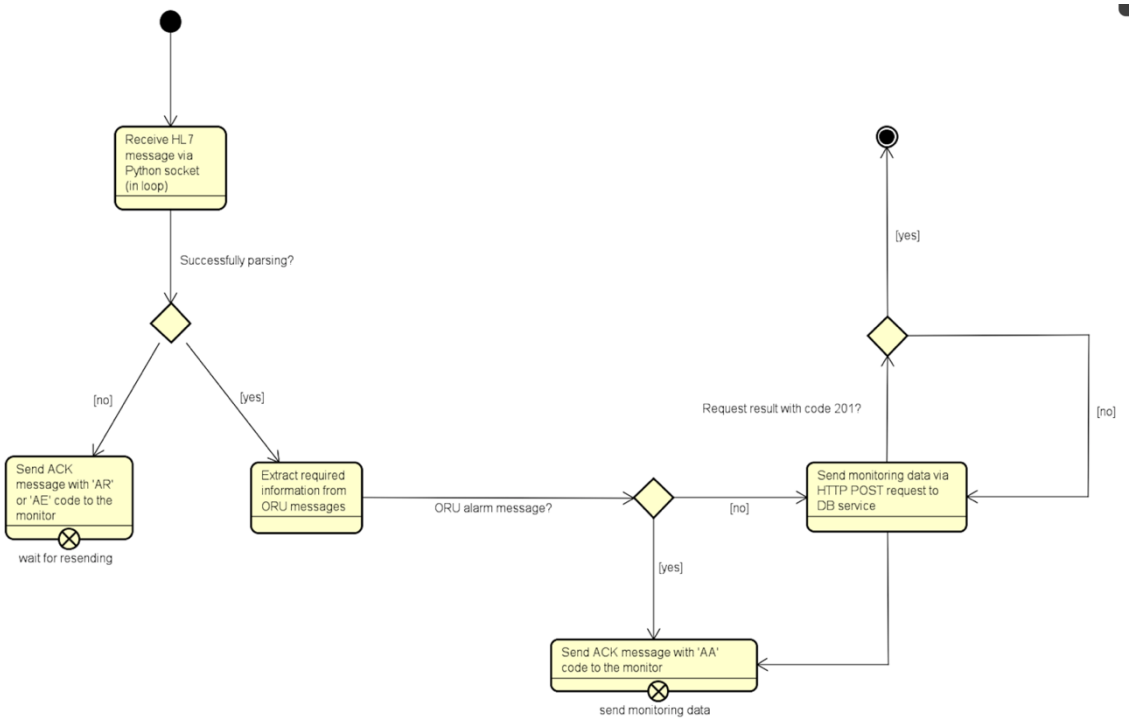

Figure 6. Activity diagram of the proposed solution.

Figure 6, illustrates an activity diagram that presents a flow processing an HL7 message in an instance of the loop implemented from the listener_socket(host, port) method.

5 Web Services Developed

Below the web services developed in Python and using the Django framework to the solution proposed are mentioned. The source code of these web services is in a private Github repository.

1. **POST /coreapp/user/**

   It creates a new user.

2. **POST /coreapp/auth/**

   It makes the validation from a login and password of a user provided.

3. **POST /coreapp/user/forgot-password/**

   It requests the change of a password to access the app.

4. **POST /coreapp/user/change-forgotten-password/**

   It allows the change of a password if the user forgets their current password.

5. **PATCH /coreapp/user/{id}/**

   It updates information of users.

6. **POST /socialnetworkapp/apple-auth/**

   It makes the validation from a login Apple using an apple id.

7. **GET /coreapp/user/**

   It gets user information from an access token.

8. **GET /coreapp/patient/search-patient/**

   It filters patients from a name, returning the patient desired.

9. **GET /coreapp/patient/**

   It returns a list with all patients.

10. **GET /coreapp/patient/{id}/monitoring-options/?type_monitoring={type}**

    It gets information of monitoring from some user with the following possible types:

    - monitor = results of some monitor.
    - apple_watch = results from some Apple Watch
    - all = all results considering all devices (e.g. monitor and apple watch).

11. **PUT /coreapp/patient/{id}/monitoring-options/?type_monitoring={tipo}**

    It updates the information of monitoring from some user with the following possible types:

    - monitor = it updates data of some monitor.
    - apple_watch = it updates data from some Apple Watch
    - all = it updates all data of devices considered (e.g., monitor and apple watch).

12. **POST /coreapp/patient/{id}/start-monitoring/?type_monitoring={tipo}**

    It generates information of monitoring to some user with the possible following types:

    - monitor = It generates the monitoring of a monitor.
    - apple_watch = It generates the monitoring of an Apple Watch.
    - all = It generates both monitoring

13. **POST /coreapp/patient/{id}/quit-monitoring/?type_monitoring={tipo}**

    It finishes the monitoring of some user with the following possible types:

    - monitor = It finishes the monitoring of a monitor.
    - apple_watch = It finishes the monitoring of an Apple Watch.

- all = It finishes both monitoring.

14. **POST /coreapp/patient/**

    It performs the validation of a login and password from a patient's access.

15. **POST /coreapp/monitoring/**

    It creates a new monitoring from a patient ID.

16. **PATCH /coreapp/monitoring/{id}**

    It updates information of some patient from an ID informed.

17. **GET /coreapp/monitoring/?patient={id}&issuer={tipo}**

    It gets information of monitoring from some user with the following possible types:

    - monitor = It shows information from some monitor.
    - apple_watch= It shows information from Apple Watch.

18. **GET /coreapp/monitoring/{id}/**

    It gets information of monitoring from some ID provided.

19. **POST /coreapp/monitoring/{id}/**

    It deletes some monitoring from some ID provided.

6 iOS App

In this section, an overview of the main screens of the app being developed are explained. To offer that explanation, we will consider that a healthcare worker desires to use the app. Thus, from Figure 7.A, two options to access the app are presented to them. The first option uses the Sign in with Apple feature with two-factor authentication. If the user will have an Apple device, they can sign in and re-authenticate with Face ID or Touch ID.

The second option was a way to show that we can offer other approaches to access the app. In that case, the healthcare worker will be able to request an authorization of access informing some data from a form presented in Figure 7.B. When the user receives an email informing that he/she is authorized to access the app, he/she can return to the app to make the login from the screen presented in Figure 7.C. In that access, a two-factor authentication can be also included and defined considering the preferences desired.

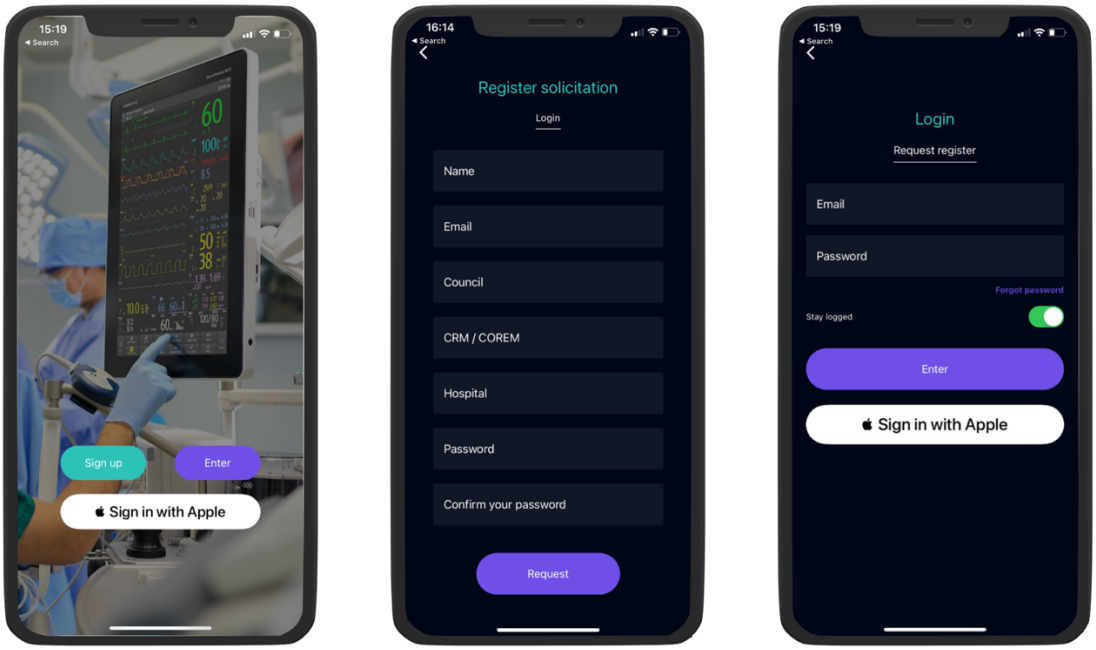

Figure 7. Screens to access the app.

Figure 8.A shows the home screen of the app with a list of patients that the healthcare worker is monitoring. If they wants to add more patients in their list, the user can tap the green button on the top right of the screen. That control is to authorize the access of other patients' data and, it can be defined by considering preferences requested. For instance, we can add a two-factor authentication exclusive to add more patients per healthcare worker.

Now, in a use-case in which the healthcare worker chooses one patient presented in the screen(Figure 8.A).The screen from Figure 8.B is shown to the user. It brings a set of vital sign data of the patient chosen, such as ECG and SPO2. These data are the last information collected per vital signal type.

If the healthcare worker desires to see only the last data collected from the Apple Watch or from the monitor connected, they can choose the option desired. Thus, only data collected per device chosen will be presented.

The user also can choose a different way to see data of the same patient by pressing the green button, located in the top screen of the Figure 8.B. Next, the screen of the Figure 8.C is presented. It allows to access more information per vital sign collected. For instance, if the healthcare worker chooses one data type, such as SPO2, they can see the data history from the patient considered. In addition, the app offers a set of fields that allows informing a period desired to filter data.

We decided to create that approach of listing data, because the Philips monitors did not offer all necessary information to reproduce charts presented on the their screens. Thus, to be consistent with the information collected, we followed that approach.

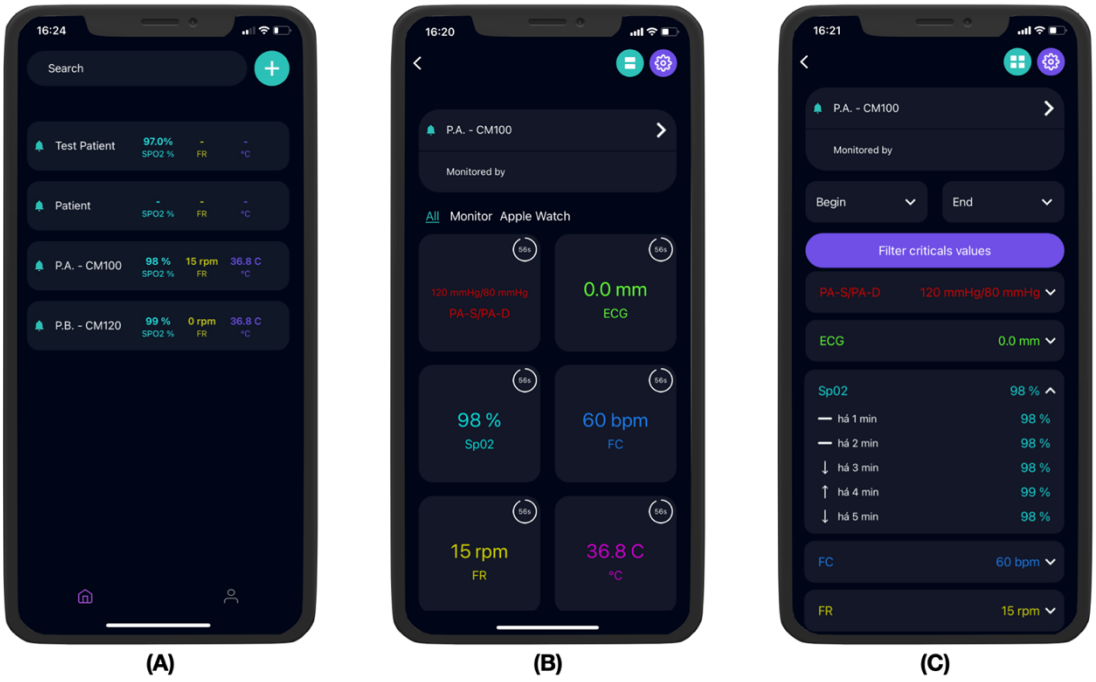

Figure 8. Screens to see data of patients.

In Figure 9 a set of screens related to settings of the app is presented. Figure 9.A allows healthcare workers choose which vital sign data they desire to see per device and per patient. Thus, for some patients, they will be able to choose to see all data, and for other patients only a subset. For example, if the healthcare worker unchecks the ECG option, as presented in Figure 9.B, when they return to the previous screen (Figure 9.C), the ECG data is not more visible.

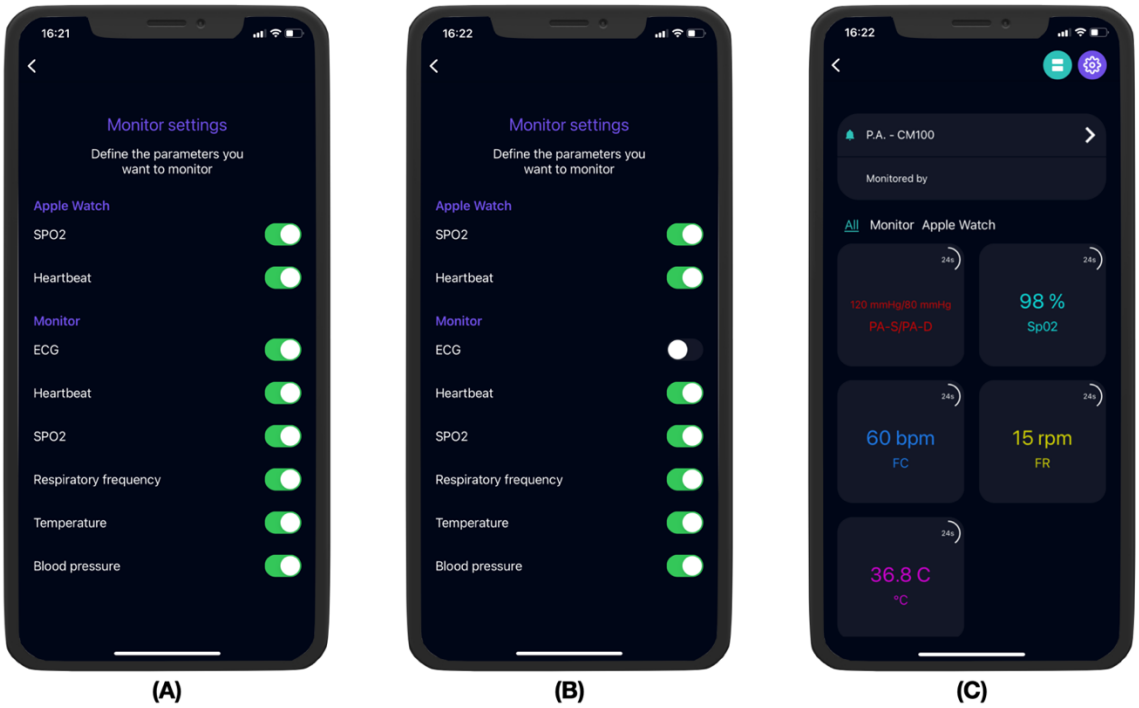

Figure 9. Screens to settings.

Finally, it is important to mention that the app allows patients to access only their own data. Thus, when they are users of the solution, they do not get to see data of other patients, as presented in Figure 8.A. Besides, when patients first access the app, they decide if they are providing informed consent to authorizes the access of health data collected from other devices (e.g. Apple watch) to be monitored by healthcare workers. To collect these data, the HealthKit framework [5] is used. Figure 10 shows currently how the grant request of health data is presented to the users.

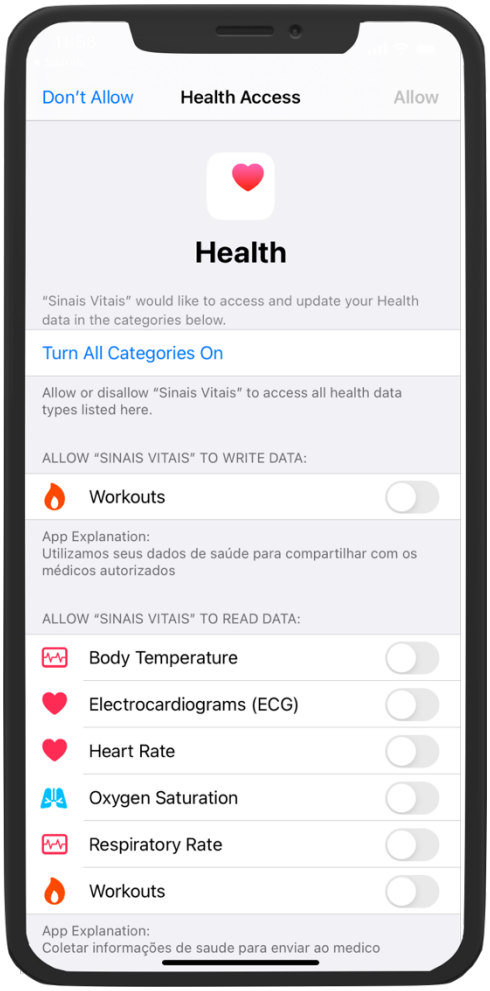

Figure 10. Requesting grant access to health data.

7 Conclusions and Future Works

This document presented a proposal of solution to monitor remotely vital sign data of patients. From the iOS app developed, healthcare workers can access data from different patients, who can use different devices to monitor them. Besides, each patient also can see their own vital sign data. This development project has been an effort to offer a mobile, secure, and reliable solution that will be able to help healthcare workers for the monitoring of patients.

1. Aiming to evolve this project, a set of next steps was defined, as follows.
2. Validating and improving the solution created with the Grand River Hospital.

3. Using other monitor brands. Having access to other monitors will contribute with the project evolution.
4. Extending the current approach to ensure the security of the data used.
5. Applying big data analysis to identify information that can contribute with the hospital and the research that has been performed.
6. Implementing alerts to healthcare workers to contribute with their monitoring.
7. Including software agents that can contribute with intelligent executions to the solution.

8 Acknowledgments

This work was supported by the PPI Softex program, Agreement No. 0200-10/2021/SOFTEX/PUCRio/Residence at TIC, financed by the Ministry of Science, Technology and Innovation with resources from Law 8.248 of October 23, 1991. In addition, this work was also supported by State Institute of Engineering and Architecture (IEEA) linked to the State Department of Infrastructure of the State of Rio de Janeiro, through Contract 001/2021 with funds from the Government of the Rio de Janeiro State.ISBN:978-1-4503-0000-0/18/06Year:2018

References

ASHTON, K. That 'Internet of Things' Thing - In the real world, things matter more than ideas. URL: http://www.rfidjournal.com/articles/view?4986. Archived at: http://www.webcitation.org/6jxDkcPuS. 2009. Last access: August 22, 2016.

BINAH.AI. Available at https://www.binah.ai. Last access: Nov 16, 2022.

DJANGO: The web framework for perfectionists with deadlines. Available at https://www.djangoproject.com. Last access: Nov 16, 2022.

FERNANDES, C. O.; LUCENA, C. J. P. An Internet of Things Application with an Accessible Interface for Remote Monitoring Patients. In: DESIGN, USER EXPERIENCE, AND USABILITY: INTERACTIVE EXPERIENCE DDESIGN- 4th INTERNATIONAL CONFERENCE, DUXU 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, 2015, Proceedings Part III, p. 651–661.

FERNANDES, C. O.; LUCENA, C. J. P.; LUCENA, C. A. P.; AZEVEDO, B. A. Enabling a Smart and Distributed Communication Infrastructure in Healthcare. In: Y.-W. Chen, C. Torro, S. Tanaka, J. R. Howlett, & L. C. Jain (Orgs.), INNOVATION IN MEDICINE AND HEALTHCARE 2015. Cham: Springer International Publishing. 2016, p. 435–446.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 1995.

HealthKit. Available at: https://developer.apple.com/documentation/healthkit. Last access Nov 16, 2022.

HL7APY: a lightweight Python library to parse, create and handle HL7 v2.x messages. Available at https://github.com/crs4/hl7apy. Last access: Nov 16, 2022.

MARKIEWICZ, M. E., & LUCENA, C. J. P.. Object Oriented Framework Development. Crossroads, 2001, 3–9.

MY VITALS. Available at https://play.google.com/store/apps/details?id=com.MyVitals&hl=en&gl=US. Last access: Nov 16, 2022.

NIST: National Institute of Standards and Technology. Available at https://www.nist.gov. Last access: Nov 16, 2022.

NORVIG, P.; RUSSELL, STUART; Artificial Intelligence (in Portuguese). Elsevier, 2013.

POSTGRESQL: The World's Most Advanced Open Source Relational Database. Available at https://www.postgresql.org. Last access: Nov 16, 2022.

UML. Available at: http://www.uml.org/. Last access: Nov 16, 2022.

VIVA LINK. Available at https://www.vivalink.com. Last access: Nov 16, 2022.

STATISTA A. Number of mHealth apps available in the Google Play Store from 1st quarter 2015 to 3rd quarter 2022. Available at https://www.statista.com/statistics/779919/health-apps-available-google-play-worldwide/. Last access: Nov 16, 2022.

STATISTA B. Number of mHealth apps available in the Apple App Store from 1st quarter 2015 to 3rd quarter 2022. Available at https://www.statista.com/statistics/779910/health-apps-available-ios-worldwide/. Last access: Nov 16, 2022.

WOOLDRIDGE, M. Multiagent Systems. In G. Weiss (Org.), Cambridge, MA, USA: MIT Press, 1999, p. 27–77.

WOOLDRIDGE, M. An Introduction to MultiAgent Systems. Wiley, 2009.