ContinuousCare:
Personal Monitoring System

André Pedrosa [85098], Filipe Pires [85122], João Alegria [85048]
Coordinator Professor: Carlos Costa

Department of Electronics, Telecommunications and Informatics,
University of Aveiro

- 2018/19 -

ContinuousCare:
Personal Monitoring System

- 2018/19 -

Written report for the Informatics Project of the Degree in Informatics Engineering of the University of Aveiro, accomplished by André Pedrosa, Filipe Pires and João Alegria under the coordination of Carlos Costa, assistant professor at the Department of Electronics, Telecommunications and Informatics and researcher at the Institute of Electronics and Informatics Engineering of the University of Aveiro.

Abstract **–** The ever-growing use of information and communication technologies in the past decades for several ends and the proliferation of mobile devices for monitoring vital signs and physical activity is enhancing the emergence of a new healthcare paradigm.
This report suggests the integration of smart devices with air quality monitors in a personal healthcare system to answer some existing limitations of similar solutions. The system achieves this by automatically gathering data from personal devices and offering a powerful online visualization tool that considers the two information sources for patients, medical personnel and ordinary users.
ContinuousCare is all about helping citizens better understand their health and body activity, aiding professional doctors with analysis tools and making available valuable data for external systems.

Index Terms **–** Health Monitoring, Environment Analysis with Geolocation, Life Tracking, Medical Information System, Smart Healthcare, High-End Technologies

Table of Contents

Abbreviations

API	Application Programming Interface
CC	ContinuousCare
DBMS	Database Management System
DETI	Department of Electronics, Telecommunications and Informatics
EEG	Electroencephalography
EMG	Electromyography
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
IEETA	Institute of Electronics and Informatics Engineering
PHPN	Public Health Personal Number
PHR	Personal Health Record
PHSS	Pervasive Health Service System
REST	REpresentational State Transfer
SPA	Single Page Application
SSR	Server-Side Rendering
UA	University of Aveiro
UI	User Interface
UPHSM	Ubiquitous Personal Health Surveillance and Management
WSGI	Web Server Gateway Interface

Introduction

The use of science and technology in medicine has been explored by man-kind throughout the past decades more than ever. Information and communication technologies brought a proliferation of mobile devices for monitoring vital signs and physical activities of citizens and, consequently, the emergence of a new paradigm of medical monitoring is growing in the research communities.

On one hand, software has been developed to integrate these sort of devices and make available useful tools for injury detection and alert, amongst other purposes, and the target audiences are mostly specific groups like elderly people or athletes.
On the other hand, the environmental conditions surrounding human life is often neglected by these healthcare technologies. Headaches and diseases with possible connections to poor air quality are not covered at all, limiting the prevention and detection capabilities of such promising tools.

In this report we propose a personal monitoring solution called ContinuousCare. Our system approaches the problem considering both the human activity and the environment it is exposed to and proposes powerful forms of data aggregation and visualization. The work described in this document was proposed by our coordinator, professor Carlos Costa, for the discipline of Informatics Project of the degree in Informatics Engineering of the University of Aveiro and is ment to present itself as a proof of concept covering technologies and paradigms learned throughout our bachelor's degree and taking advantage of smart devices with public APIs. It is assumed that the reader has knowledge about the proposal (1), though we make sure all aspects are explained in detail.

The document is divided into three chapters and covers the development process of ContinuousCare:
Chapter 1 covers the planning phase, where we dedicated time to do research around existing software in the same topic and to learn about smart devices that we could use for our system; it is also in this chapter that we present the requirements for the system as well as the target audience.
Chapter 2 is dedicated to the technical aspects of the solution; we expain the architecture of the system, the technologies used and the strategies implemented;
Chapter 3 describes the solution as a product - meaning the applications developed that consume our public API, the process of deploying the entire system and the developers guide for future improvement or integration with other systems.
Conclusions and final remarks are presented after these chapters, and more useful, though secondary, information is provided in the appendix section.

1. Requirements Elicitation

This chapter is the result of a specification process led by our team in order to strain all the requirements for the system. Here, the main goals were evaluated and their scope defined. Moreover, a study around the state of the art was conducted by us in order to learn from research strategies with good results and the technologies being used in similar scenarios.

1.1. State of the Art

All research efforts were dedicated to finding projects, studies and technologies with similar purposes. This improved the system requirements firstly by discovering the latest and most innovative works around this area, and secondly by understanding what went right and wrong in those works. An overview of these works is present next.

The two last articles are not as up-to-date, nevertheless we dediced to maintain them as part of our state-of-the-art as they brought valuable ideas and were very much connected to our own concept.

1.1.1. Health Monitoring Systems using IoT and RaspberrPi

This report (2) from four students at SSPACE college, India, is similiar to ContinuousCare in the area of interest, but they focus on different implementation aspects. Their project consisted in developing an IoT system responsible for: collecting metrics from a user in a non-invasive way and through multiple sensors; aggregating the data collected; processing it and creating some notifications in case of the occurrence of dangerous situations.

Their main focus, however, was mainly on the data collection aspect, principally in the hardware - opting for a IoT approach, a system architecture that is on the rise nowadays and basically translates to a scalable network of little smart devices and sensors, each one capable of collecting, making some minor processing and sending the data collected to a central machine responsible for aggregating the information from all the nodes. The fact that each node is very small (no bigger than a smartphone) and that the network can be scalled globally, makes IoT the perfect candidate to the base structure for a medical monitoring system, being each node assigned to a patient.

This was the idea of the authors and they implemented this with the help of a Raspberry PI, various sensors such as a temperature sensor, a blood pressure sensor, a heartbeat sensor and a ECG sensor - all connected to the Raspberry.

Even though there are quite a bit of conceptual similarities with our system, the entire architecture is quite different from our intents and therefore not much could be retrieved from their study. We used their analysis as a sort of guideline of what should be considered in terms of metrics. We also took in consideration the idea that it could be a possibility for us in a future more distant than the scope of our project to adapt our data collection devices to IoT networks or make our own data collection modules for better adaptability and scalability.

1.1.2. Angel-Echo: a Personalized Health Care Application

Anger-Echo (3) is a combination of technologies proposed by a partnership from the american universities that offers a health monitoring system with a voice interface. Their two main data sources are the Angel Sensor, a wristband that monitors heart rate, blood oxygen, step count, sleep quality and

so on, and the Amazon Echo, a wireless speaker and voice command device used for the interaction between user and system. Angel was a very appealing device to be introduced as one of our supported devices as it even had an API and an SDK, but it seemed to no longer be available on the market, so we had to discard that option. The interaction with Echo was very interesting too, but not exactly the kind of interface we were looking for, so we kept the idea as one possible new module to be added to ContinuousCare in a future beyond the scope of our project.

With regards to the implementation itself, the article did not go into much detail and focused mainly on the voice interface itself. We mention this article as it is a good example of the type of tools that are currently being developed to increase the value of smart healthcare systems. Their tests proved to be a very promising solution and much is to gain from interfaces that overcome the inexperience of users such as eldery people when it comes to smart devices.

1.1.3. Geolocation Based Health Monitoring, Consultancy And Alarm System

This article (4) , written by Emre O. Tartan, from the Baskent University, and Cebrail Ciflikli, from the Erciyes University, both in Turkey, presents an Android Application for health monitoring, consultancy and alarms based on geolocation. The product focuses on heart rate tracking, although mentioning that it is capable of integrating other metrics, and offers a system of detecting abnormal values and notifying the so-called caretakers and doctors. Unhealthy behaviours are detected by their app according to the physical activity in practise and to the age of the user, and informs the respective accounts through email, sms and notification.

Their analysis on heart rate and ECG patterns and limit values were valuable to our system, as they dedicated much of their time in defining what was dangerous for each thresholds (according to age and heart conditions). In terms of practical usefulness of their implementation, not much could be learned from their article, as they did not go too deep on the subject. However, this is one of the few studies we came across that actually took advantage of geolocation to bring value to the product (although in their case was simply to indicate the nearest hospital in case of an emergency).

1.1.4. Beyond Health Tracking

The article about Beyond Health Tracking (5) from University of Trento and Hospital of Milan is about a personal health and lifestyle platform. Basing themselves on the idea of personal health records (PHR), this team of researchers created a personal health and lifestyle recorder which makes easier for an individual to keep track of and maintain a healthy lifestyle and helps him with personalized advice through a platform that monitored and assessed all data collected. Here we saw the first uses of interaction between system and user that we intended to explore ourselves.

Their GPS based monitoring application brought useful contributions to our research since it tracks a cyclist's position in real time, computes his/her power output and overall energy consumption, shares that data with others, and provides nutrition advices. In our case, the supported devices did not have GPS embedded, so we had to come up with a different solution.

The sharing feature for performances and result presented itself a very interesting characteristic of their solution, since it proved that people and professionals follow advices given by others with same goals. However, the application of this feature in CC seemed to be not as useful.

1.1.5. Ubiquitous Personal Health Surveillance and Management

The UPHSM (6) system is in many ways similar to ContinuousCare. This project integrates health-related sensory devices to aid senior citizens and patients who need long-term attention to their ilness, with the help of ubiquitous surveillance and remote management of the recorded data. It differs on the target audience, since it focuses on elderly patients and messages sent to family members and doctors, and it ignores the benefits of an environmental data analysis.

The research team used several sensors that could be interesting to be integrated with ContinuousCare, such as: electromyography (EMG) sensor for muscle activity, electroencephalography (EEG) sensor for brain activity, blood pressure sensor, blood glucose sensor, breathing sensor for respiration, etc. Even though we focused on wearables already common in regular citizens, these sensors would enrich the diversity of datatypes and improve the diagnoses of doctors connected to our system. Nevertheless, the need for aditional hardware presented a big challenge to the UPHSM team. Their modularity and use of wireless communication technologies seemed useful to us, taking in consideration the success of the software, but was not directly applicable.

1.2. System Goals

In short words, our goal was to develop a system of web information that allows the automatic gather of clinical and environmental data and an intuitive form of visualization of the daily information regarding a citizen. The solution, what we later called ContinuousCare (CC), collects data of diferent types and natures and allows a simplified form of registering basic physical and mental health states of a person.

The usage of our system is potentially visible anywhere a person is. Since the clinical and environmental data collected helps analyse several aspects of a person's life, detect bad health habits and understand symptoms and complaints, the aplicability of the system extends to anyone concerned with their lifestyle quality.

When called for a regular check-up by the family doctor, a CC user no longer needs to try to remember relevant episodes to mention. Instead, the doctor simply requests access to the user's information collected and makes the analysis with the tools offered by the system itself. This makes the usage of the software by clinical offices very appealing.

The idea, however, is to increase the depth of applicability and build the system with great scalability and mechanisms to serve not only our daily routines, but also other softwares and machine learning algorithms and make available rich data to more elaborate studies on long term diseases. The way we prepared the system for this is explained further ahead.

1.3. Requirements

Once the first part of the specification process was completed, it became possible to define all of the requirements needed to be met for the system to work as intended. These requirements are listed in this section and the target audience is described with greater detail along with the actors' use cases in the next.

1.3.1. Functional

There are three main components in the system. These are: the devices used by the actors, the user interface (UI) from where they can interact with the system, and the online server, where most of the activity happens. In a typical scenario, an account will have associated two devices, 1 non-portable for air quality measurements, and 1 smart bracelet for the remaining features. The following list covers all functional requirements that must be fulfilled. It served as a description of what the software should do.

1. The home device must send data to the server whenever an actor is indoors.
2. The personal device (bracelet) must send data to the server whenever an actor uses it.
3. The server must store all data collected from the devices in a database.
4. The system must keep track of the actor's geolocation at all times.
5. The server must collect environment data based on the actor's geolocation.
6. The system must still collect data even when the devices are not being currently used (worn).
7. The UI must make available to the actor all data collected.
8. The UI must present the latest data and allow the access to the progress data (past periods of time).
9. The system must support multiple accounts.
10. The system must have 2 separate user profile types.
11. The system must be prepared to allow access to the information to other actors (e.g. medical personnel) with the owner's permission.
12. The system must be prepared to receive and store health states according to the actor.
13. The system must support accounts with several home devices.
14. The system must support several accounts with the same home device.
15. The system must not allow the use of several personal devices by the same account.
16. The system must allow the association and deassociation of devices (e.g. device update).
17. The UI must inform the actor which devices are associated.

1.3.2. Nonfunctional

Since usability and data processing were key aspects for the success of the software, a list of constraints that it must respect was created for future refference and quality testing. This list also includes the properties/characteristics it must exhibit.

1. The system must be available at all times (24/7).
2. The devices must not stop producing data (unless if they run out of battery or stop functioning).
3. All information must be accessible by the actor without exceptions.
4. The system must always persist all data of all accounts.
5. The system must be prepared to expand in case of a large increase of number of users.
6. All system functionalities must be well documented.
7. The system's database must store all data in the least amount of space possible.
8. All actors' personal data must be confidential to the owners alone.
9. The user interface must be supported by the most common web browsers.
10. All requests must be responded within a timespace no longer than 1 second.

1.4. Target Audience

1.4.1. Actors

The target user for the services provided by the system belongs to one of 2 groups (with the possible existance of a third group): either he/she is a common user (a client) or a medical user (a doctor). The third group reffers to a developer that integrates the system with his/her own application.

- **Client**: Has access to most features of the web interface; He is able to see historical data, register relevant episodes and symptoms, give permission to doctors to see it and manage the associated account devices.
- **Doctor**: also refered as medic; Only has features available when a permission is given by a client; Assuming it is, he can analyse client's historical data as a helping tool to make a better diagnostic of the clients health state; The system grants visualization tools dedicated to doctor accounts.

It is expected to see medical personnel using the system with high frequency. The levels of experience with technology needed for both these actors to be confortable with the system are relatively low. An actor only needs to be used to work with the basic tools of a computer and a web browser in order to properly use our tools.

Regarding the set up of the external devices and their maintenance, these are tasks that may make actors encounter difficulties that require some more experience. However, it is a matter taken in consideration and we intend to make available everything necessary for any kind of person to be able to use the system without much entropy.

Both actors need to be authenticated before any communication with the server, since sensitive and personal data is stored and transfered.

1.4.2. Use Cases

Figure 1 represents the use case model of the system, having only one main package: the web application. As said before, both the client and doctor use the web application but with different purposes. For this reason, we grouped their use cases in different packages within the web app package.

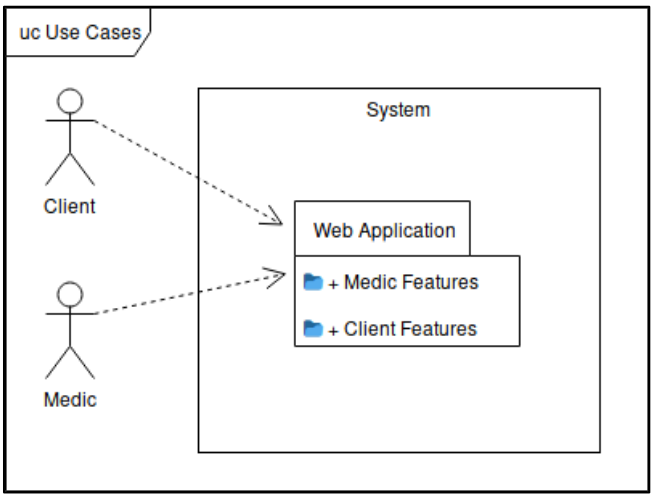


Figure 1: Use case model in UML format.

The diagram in figure 2 shows the activities that both the client and the doctor can do, with each package linked to the respective use cases. The doctor package is mostly characterized by its dependency upon the client package.

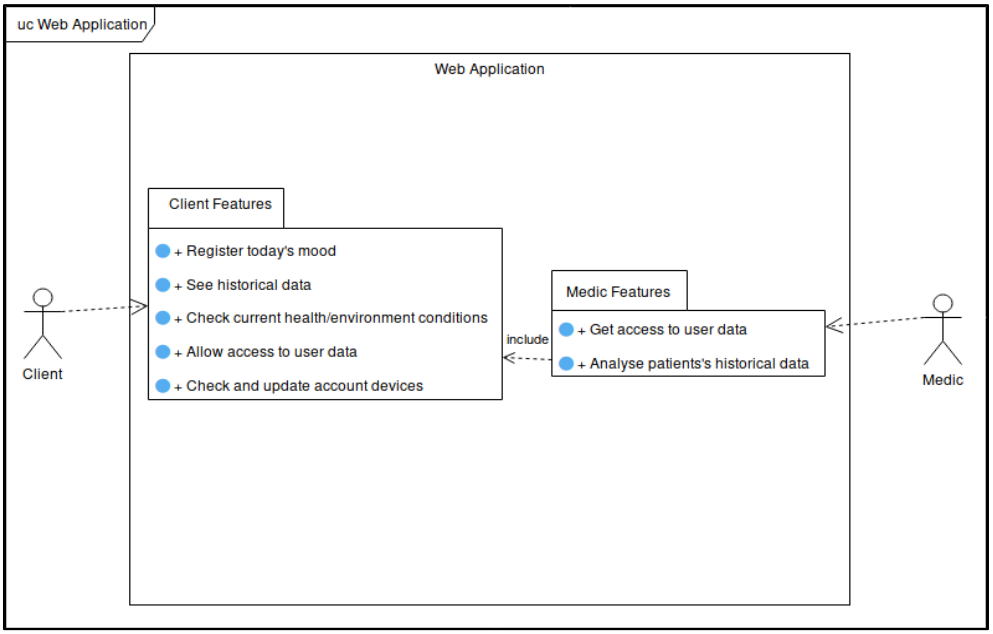


Figure 2: Web application use case diagram.

All operations that a client can do on the web application are presented and described bellow. All use cases on figure 3's diagram assume that the client is already authenticated.

- **Register today's mood**: Allow the client to register his health status during the current day by selecting options from a predefined set. This makes possible to, later on, use the data retrieved from the external devices and already labelled by the system to be fed to machine learning algorithms applicable according to the labels. You will find in *appendix A* a table with the default possible daily emotions and health issues made available as options for the user.
- **See historical data**: Allow the user to browse through the personal historical data so that he can keep track of his health and progress.

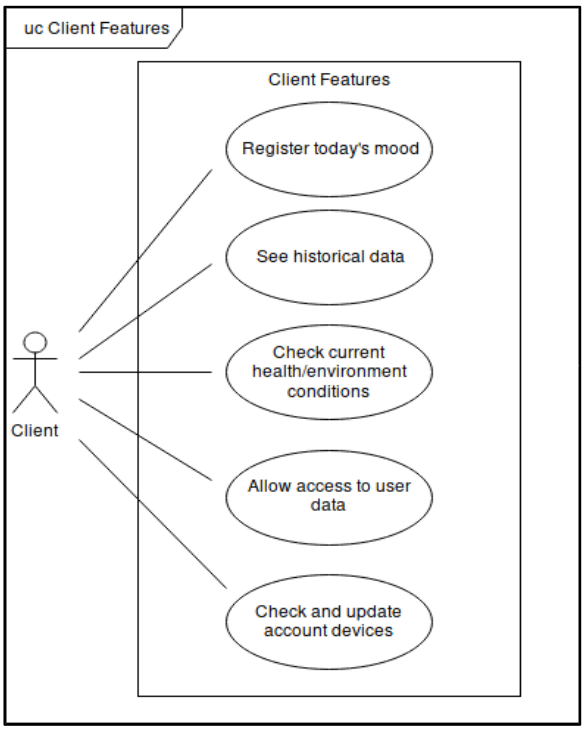


Figure 3: Client features.

- **Check current health/environment conditions**: Although the user can see this type of data on the external devices, a second option is given to the client where he can access the web application to consult the latest values appropriately pointed out when in unhealty levels.
- **Allow access to user data**: Allow a client to grant temporary access to a medic for professional analysis purposes. This activity is aimed to be used during medical consultations.
- **Check and update account devices**: Make possible the association and dissociation of new/old external devices to the user's account.

All operations that a doctor can do on the web application are presented and described next. All use cases on figure 4's diagram assume that the medic is already authenticated.

- **Get access to user data**: The same way a client can grant access to doctor accounts, a doctor is also able to ask permission himself. The acceptance process must envolve the presence of both entities in the same space.
- **Analyse patient's historical data**: Allows the medic to analyse the historical data during a consultation with a client. This activity depends on the medic having the permission to access the data of the patient that he is currently attending.

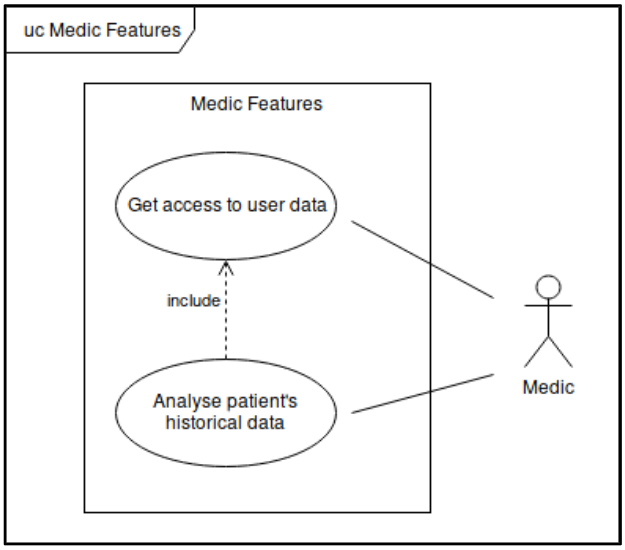


Figure 4: Doctor Features.

2. System Architecture & Implementation

2.1. Domain Model

The domain model, depicted in Figure 5, describes the entities, their attributes and relationships and also the constraints that govern the problem domain. This model helped us keep an abstraction of the system in a more readable format.
The first entity is the user, divided into the two actors specified in section 2.2.1. The client can have external devices associated that can be of two types:
1. Bracelet (FitBit)
2. Home evice (Foobot), or Public API (In case the client isn't home)

Regarding the permissions and data sharing feature, both the medic and the client can ask for or grant them, respectively, more than once. This implies the existance of a one-to-many relationship. Each permission can cover one or more records. Hence the presence of a many-to-many connection between record and permissions.
Each client has a set of external devices associated, each external devices can be associated with more than one client. For this, the relation between the client and the external devices is many-to-many. At last, a record is associated with only one client and only one external device so for both connections one-to-many relationships were used.

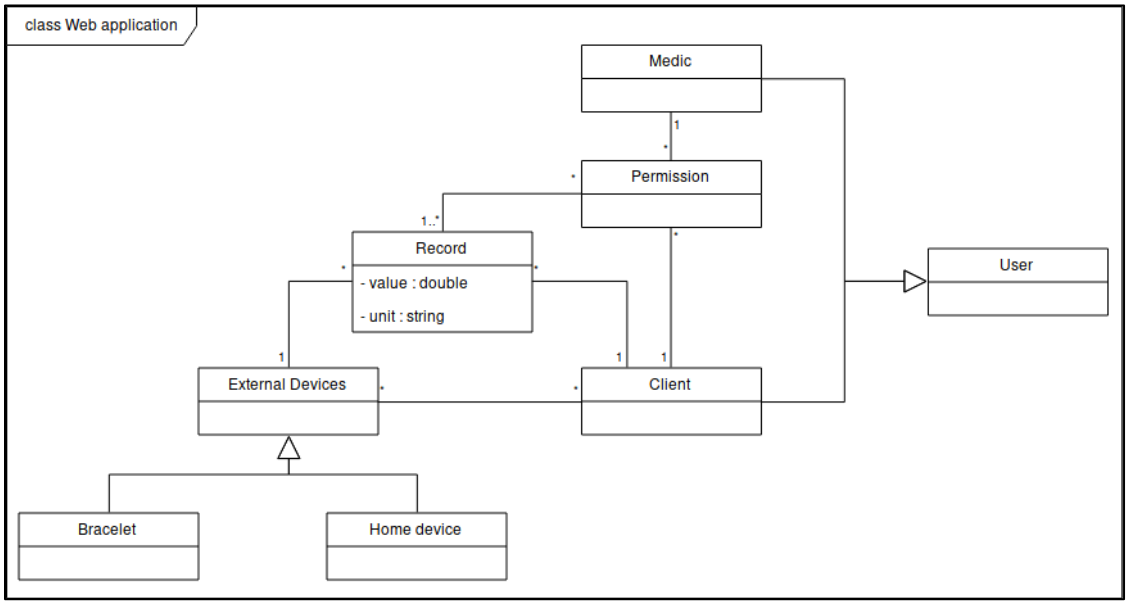


Figure 5: Domain model.

2.2. Architecture Diagram

Many design decisions had to be made carefully for the success of the solution. The key decisions aimed to ensure the robustness and scalability of ContinuousCare and regarded the collecting of data from different sources, the storage strategy appropriate for our needs and the database load and access-time optimizations.
In this section we explain our design choices and present the architecture diagram of the final solution, describing the tasks of each component.

ContinuousCare follows a modular architecture, where each module is replaceable and updates to one do not imply refactoring of others. This not only allowed us to maintain a reduced interdependency within the system, but also helped us divide work amongst the team during the development phase.
The Monitoring Server displayed in figure 6 is our main server and is divided into 3 different units: processing unit, persistence unit and REST API unit. The processing unit is responsible for gathering the information from all the devices and other external sources like public APIs; This operation is scheduled, which means that it is executed many times along the day; The unit is responsible for doing this scheduling for processing the data collected, like filtering it, assigning labels, etc. The persistence unit is the component responsible for storing all the data after processing; It is this component that interacts with the database interface and makes sure every single field is persisted correctly. The REST API unit is the component responsible for supporting a REST interface for the entire system, serving as the point where every client application connects to.

The implementation of the GPS service was not in our initial plan, as it is the result of a solution to the unexpected fact that none of the available smart devices that could be used by us provided access to its location. As geolocation was a crucial aspect of CC, we had to implement from scratch a service consisting of a simple server responsible for receiving the users' locations and a mobile application that tracks them. This application is mentioned further in the report.

The UI server is our main interface with the end-users. We found that the most appropriate solution would be a web application running on this server that communicates with the Monitoring Server API and presents in a simple and intuitive way the information gathered along with other useful features. More about this will also be described in the next chapter.

But what about the smart devices that have been continuously mentioned in this report?
ContinuousCare supports two monitoring devices: the Fitbit Charge 3 smart bracelet and the Foobot home device. The bracelet is a health and fitness tracker (7) that measures daily calorie burn, pace and distance, and tracks heart rate, sleep hours and stages, physical activity and even female health for ovulation calendar prediction. Our focus is on collecting, treating and presenting the data regarding the calories, heart rate and sleep stages, as these present more interesting information for analysis purposes. Foobot, on the other hand, is an indoor air quality monitor (8) capable of detecting invisible, odorless pollutants that fall into three categories (volatile compounds, particulate matter and carbon dioxide), calculating temperature and humidity levels and alerting users about activities that affect the air conditions. We collect the data from all of its sensors instead of the categories they fall into for treating our own way.
As foobot only measures local air quality, in order to still keep track of the surroundings of the user, we collect data from a public weather API called Waqi (9) when he/she is +50 meters away from foobot.

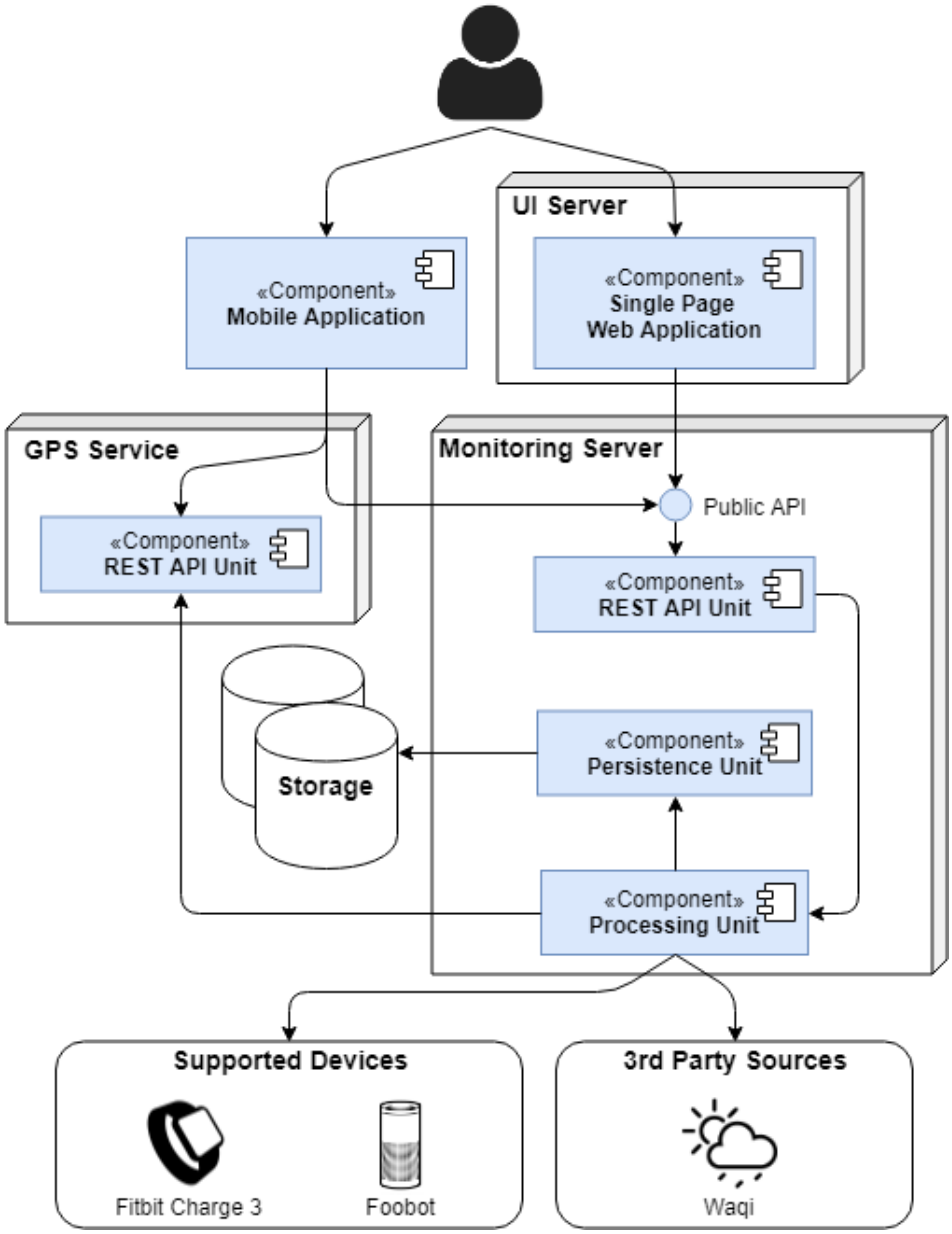


Figure 6: Architecture diagram.

Moving on to the first mentioned major decision (in the beggining of this section), we wanted to make sure the system was prepared to increase the number of supported devices throughout time, as well as to replace or update those already supported. We did this by creating 2 abstract classes to work as templates to all data sources that may be added to the system, one with the name *DataSource*, representing the data source itself, and another called *Metric*, that represents any metric that can be retrieved from a *DataSource* instance:

```python
class Metric(ABC):

    def __init__(self, dataSource):
        super().__init__()
        self.dataSource = dataSource

    @property
    def url(self):
        uuid = self.dataSource._authentication_fields.get("uuid")
        if uuid:
            return self.URLTemplate.replace("UUID", uuid)
        return self.URLTemplate

    @abstractproperty
    def URLTemplate(self):
        return ""

    # remaining methods not shown here
```

```python
class DataSource(ABC):

    def __init__(self, authentication_fields, user, id, location):
        super().__init__()
        self._authentication_fields = authentication_fields
        self._user=user
        self._id=id
        self._location=location

    @property
    def user(self):
        return self._user

    @property
    def id(self):
        return self._id

    @abstractproperty
    def header(self):
        return ""

    # remaining methods not shown here
```

Figures 7 and 8: code fragments from the abstract classes of the Monitoring Server.

Being that a major part of our healthcare system is the ability to collect and store all the medical and environment data of the users, the storage strategy to be adopted had to be well measured and appropriate for our specific needs.
A hybrid database was our approach to this concern. We designed a relational database in MySQL (10) for the structured and interconnected data: accounts information, permissions between users, and devices. For the bulk data received from the devices and external sources, we chose a time-series driven database, implemented with InfluxDB (11). This second one was found the best technology for our case as it is dedicated to continuous data streams and queries with time intervals. It is this database that feeds most of our web application's features.
For the time-series database we had to read the documentation available and learn how to use it properly as no member had any previous experience with such type of storage.

Yet in the persistance domain, after some tests we concluded that the need for optimization was present, since the server was taking too long to answer when executing methods that required writing to the database. The source of this slow behaviour was that transactions were being processed one at a time, when they could be executed once with all values being written by the same procedure. We took this in consideration and made an effort of, when possible, writing all the values at once, like the case when we fetch various metrics in a given timestamp and all those values and metrics could be stored by a single transaction.
However, this did not solve the bottleneck problem in the long run, so a few more mechanisms were implemented, such as keeping in cache the values for a given time interval and periodically writing all the values stored with a batch write.

Lastly we needed to guarantee that when increasing the number of accounts and devices and the frequency of usage of the system, we would manage to handle the increased number of requests to our endpoints, and so we decided to deploy our server into a Web Server Gateway Interface (WSGI) using the Gevent Python library (12) and use NginX (13) as a reverse proxy to help us with issues like load balancing and caching. This proved to be valuable when testing the overall performance.

2.3. Technological Model

We have already talked about some of the technologies used, as they play a big part in the quality of the final product. In this section, we list all of them pointing the reader to where they were useful to us.

The database management systems (DBMS) chosen, as previously explained, were MySQL and InfluxDB. In addition to the advantages of InfluxDB due to its time driven nature, this system provided features such as the ability to, after a defined amount of time, crunch the stored values (or even deleting them if necessary), reducing the storage space occupied.
For the main server we choose Python as the primary language, the reasons being its flexibility, efficiency and being fast enough to meet our requirements. To integrate every component with each other we needed to use multiple libraries, being the most important ones:

- Flask, a microframework that help us construct our REST API;
- WebSockets, a library used for creating a websocket server (14);
- GEvent, a library that provides a high-level synchronous API, that in this case was used for creating the server execution environment (12);
- Influxdb, a library that provides a client to the InfluxDB database (15);
- Mysql-connector-python, a library that provides a client to the MySQL database (16).

All these libraries and technologies were fairly new to us, being necessary to first learn how to use them and then implement the necessary features.

In relation to the GPS server, we adopted a similar strategy from the main server, but as it was a lot simpler we ended up only using Flask to create the endpoints. And for the mobile application that interacts with this server, we developed it for Android (17) smartphones only, with the use of some native libraries such as Volley to make the HTTP requests and Location to gain access to the host's GPS coordinates.
For the main user interface, the Web Application, we decided to use the HTML5, CSS3 and JavaScript based frontend developing framework VueJS (18), with the abstraction of NuxtJS (19). Nuxt was chosen because the framework offers a project structure very intuitive by default and makes the transition between the setup of Single-Page Application and Universal Application (with server-side rendering) very easy. Our final product deploys a SPA, though small changes would be needed to fully support SSR.
Vue already deals with scalability, robustness, modularity and performance, so the efforts focused only on finding libraries and plugins that materialized the desired features:

- Axios, a library that aided in making the necessary HTTP request;
- Bootstrap, a toolkit used for creating a responsive website that adapts to any device;
- Leaflet, a library that enabled us the usage of interactive maps for a better visualization of te information;
- Apexcharts, a library that supports the creation of several types of charts and graphs for data visualization.

These front-end frameworks and libraries were totally new to all members of the team and showed to have a difficult learning curve that all the members had to overcome. More about them will be mentioned when describing ContinuousCare as a final product.

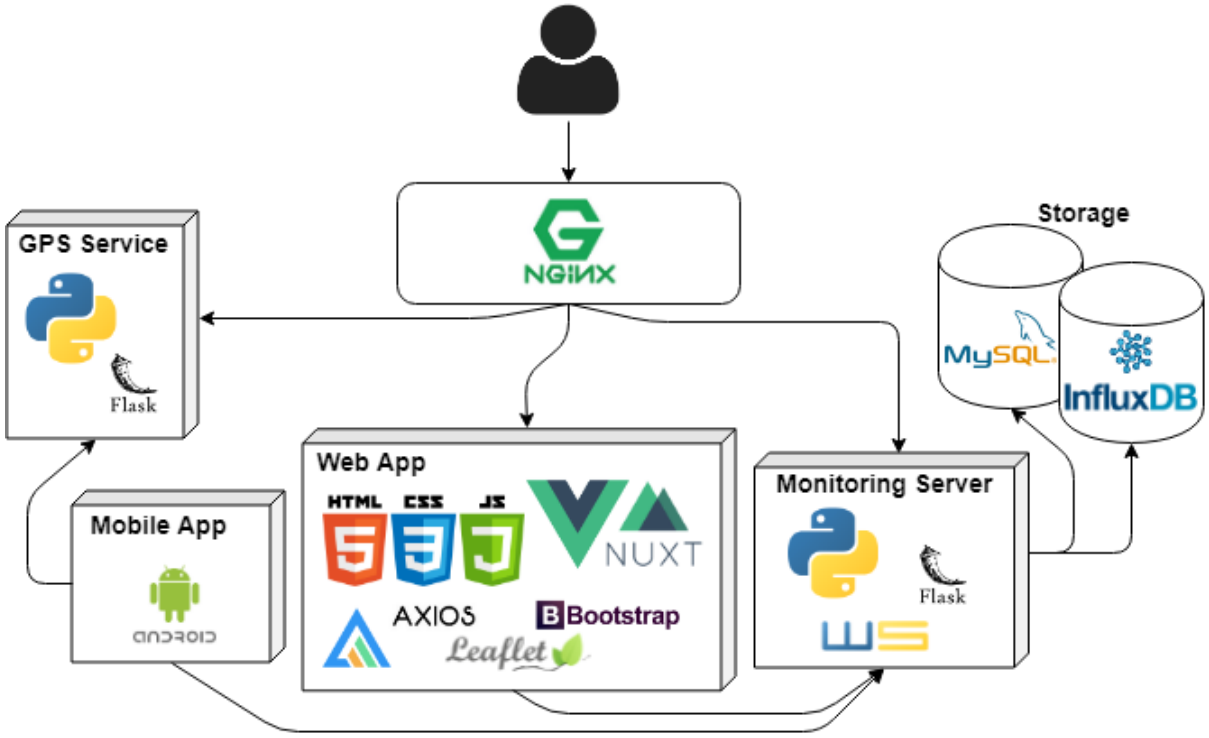


Figure 9: Implementation diagram listing the technologies used in each module.

2.3. Data Collection and Aggregation

As mentioned on a previous section, we get the personal data from devices external to our platform. To get that data we depend on their public APIs which require a set of keywords called “secrets” requested from the user by CC when he wants to associate a new device to his account. The device’s information is stored on the relational database where a specific supported device is linked to the user. Also, a device has several authentication fields that are used to interact with its public API (and its location in case of the home device – Foobot). In appendix C, we make available a highly descriptive diagram of our storage implementation in case the reader has interest in knowing more about it.
The responsibility to retrieve the personal data relies on the processing unit of our Monitoring Server. Here, for each client, a thread is in charge of making the necessary API calls with the respective authentication fields provided by the client.

One important feature when we are retrieving data from sensors is to get real time data; on this area we are limited by how the device stores and exposes what it collects; for that we assume that the devices are synced with their latest retrieved data. This proved to create limitations, as no guarantees exist that all users keep their devices synchronized and online.
All environment and health personal data is composed by a set of values of specific metrics with a timestamp associated. This type of data is stored on InfluxDB.
Instead of creating a single measurement for each metric we divided into several measurements:

- Environment: values for a specific location where the user was present
- Event: moments where some health or environment metric went out of the normal standards
- HealthStatus: health metric retrieved from bracelet devices (Fitbit)
- Path: array of latitude and longitude values
- PersonalStatus: a list of moods (appendix A)
- Sleep: sleep patterns and their durations

The only measurement that has also data on the relational database is sleep, to auxiliate the retrieval from the time series database.
To retrieve health status and sleep we use our Fitbit bracelet, our android application for GPS location and for environment we have two sources: home devices, if the user is within a 50 meters radius from it or the third-party public API Waqi where we retrieve information according to the user’s location.

In the long run, we will be holding a big load of data and some of it starts to be irrelevant such as data from five or six month from the current time. Our system maintains all data as we agree that it is important to allow the user and the medic to access all historic data from any point of time. However, we support the application of an aggregation operation over the data after a defined time.

Having all this data can come in handy for research purposes in the field of machine learning. The desired outcome for this integration would be to offer models rich data so that they in return give predictions regarding diseases or harmful feelings such as headaches, and generate warnings regarding dangerous behaviours and hazardous environments. One real life example would be to detect that for high levels of CO2 the clients has headaches; so, before happening one event as such, the system warned the user that the levels were rising and that could lead to the occurrence of headaches.

2.4. Public REST API

Our public API uses HTTP requests to retrieve formatted data from ContinuousCare. Its main purpose is to feed the user interface with information for the end-users, although its use is not limited to that end. We make available the hyperlink to the documentation of the REST API for interested readers and/or developers looking to integrate our services with another system. The tool we used to create the documentation was Swagger, a development platform with dedicated tools for the purpose (20):

https://app.swaggerhub.com/apis/joaoalegria/ContinuousCare/1.0.0

POST	/signup	SCHEMA	InternalErrorMessage
POST	/signin	SCHEMA	LogicalErrorMessage
GET	/logout	SCHEMA	InvalidErrorMessage
GET	/devices	SCHEMA	AuthenticationErrorMessage
POST	/devices	SCHEMA	UnauthorizedErrorMessage
PUT	/devices	SCHEMA	SuccessfulMessage
DELETE	/devices	SCHEMA	NewDevice
POST	/mood	SCHEMA	FitBitAuthFields
DELETE	/mood	SCHEMA	FoobotAuthFields
GET	/environment	SCHEMA	User
GET	/healthstatus	SCHEMA	UpdateUser
GET	/personalstatus	SCHEMA	Medic
GET	/sleep	SCHEMA	UpdateMedic
GET	/event	SCHEMA	GrantPermission
GET	/path	SCHEMA	NamePermission
GET	/download	SCHEMA	NumberPermission
GET	/profile		
PUT	/profile		
DELETE	/profile		
GET	/supportedDevices		
GET	/permission		
POST	/permission		
GET	/permission/{medic}/accept		
GET	/permission/{medic}/reject		
DELETE	/permission/{client}/pending		
DELETE	/permission/{medic}/accepted		

Figure 10: Summary of the ContinuousCare API's end-points.

3. Final Product

3.1. Applications

In this last chapter we present the ContinuousCare applications that interact with our REST API and make use of the devices supported by our servers. These are the product ment for the end-users and the climax of the project.

3.1.1. Web Mockup

The mockup attached in *appendix B* is the result of a useful planning of the browser-based UI. Although this basic design suffered several changes throughout its implementation phase, the general structure of the web application remained faithful to our initial idea. This mockup was developed using an online tool called *Proto.io* (21).

3.1.2. Web Application

Most of the time spent by users is expected to be on the ContinuousCare web application. It is through the website that a user directly interacts with the system and from this UI both clients and doctors can analyse clinical data once processed. In this section we describe the list of features supported by the application's final version, following the order in which the use cases were mentioned and described.

Accessing the website is done like in any other destination address. Once in it, the application displays the *welcome* page, with the name, logo, and a brief description of what CC offers. The four main services provided are also presented to the visitor, along with a chance to create an account or to find out more about ContinuousCare.

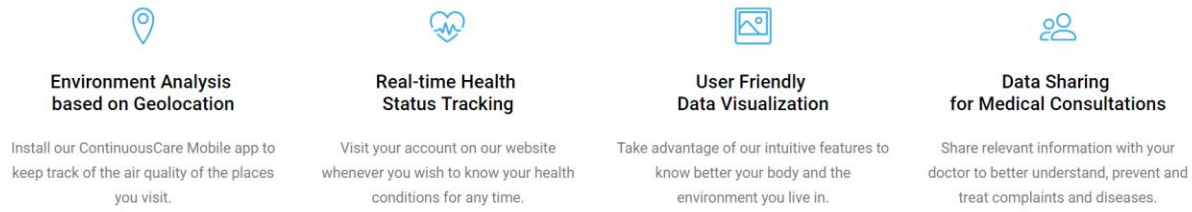
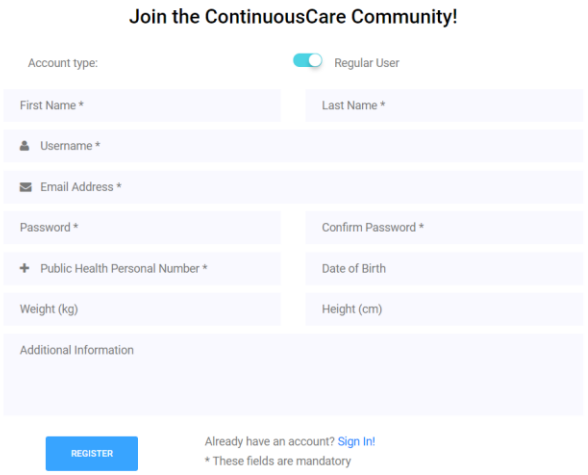


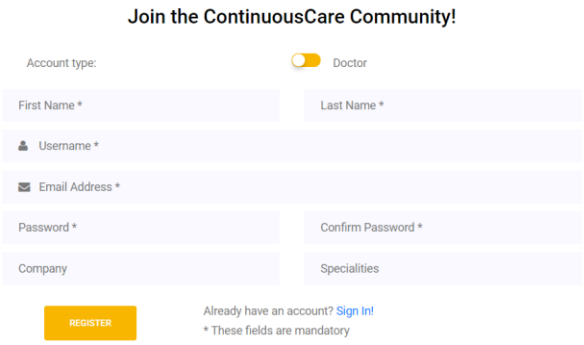
Figure 11: Four main services provided by the CC system and presented on the website's *home* page.

The registration process is done as simple as possible, asking the user to fill-in several health-related characteristics such as weight and the Public Health Personal Number (PHPN), but allowing the submission without filling them all right away. It is in the *registration* page that a user decides whether he/she desires to create a regular account or a doctor account. The difference between these two types was explained in a previous chapter, but they will be mentioned in more detail over this section.

Once the submission is considered valid, the user is automatically signed in and redirected to the *home* page. Here, two main columns split the user's attention: the first one represents a vertical timeline with the latest events detected / registered by the system; the second is dedicated to allowing the registration of the user's daily symptoms, emotional moods or any other information he/she finds relevant.

Each event belongs to one of two categories: either it contains the updates sent by the user from the second page division (e.g. 'Headache') or a warning message regarding one vital sign or environment condition under values considered harmful (e.g. 'High Heart Rate'). In either of these cases, the event box comes with a list of values related to the event as well as the respective metrics, if any (e.g. 'HeartRate: 102 bpm').





Figures 12 and 13: Two screen captures taken from the website's *registration* page.

The registration of daily moods was thought from the beginning of the project as a good form of allowing the user to keep track of his/her own development throughout any period of time. Presenting default options helps the user maintain the focus of his/her feedback for future analysis, while allowing the input of new options makes the feature more flexible and user friendly.
Through the timeline the user can make this analysis on past daily submissions, however the application offers a tab dedicated solely for this purpose. This tab is described further ahead.
Still in the *home* page, below the two main columns, the user has access to a line chart where the information about the previous night's sleeping pattern is presented. This feature is only useful if the user already has a Fitbit bracelet linked to his/her account and wears it while sleeping (connecting a new device to an account will also be explained in this section; the supported bracelet is designed to be used 24 hours per day, thought this is not mandatory in order for the account to work properly).

Once logged in, the second tab present in the top navigation bar directs the user to the *events* page. Here, a similar view is displayed, where on the left the user sees the same timeline previously presented, but on the right has access to a pie-chart. This chart is ment to give an intuitive visualization of what are the most frequent events. This in return helps the user better understand what might be his/her bad habbits and how does he/she most frequently feels.
If a user is interested in knowing the past occurrences of a specific event, he/she needs only to select one from the timeline or from the chart and a set of similar events from the same month is presented below the chart and timeline, with the information about each one of them. Having access to such feature allows the user to compare episodes from the past and take conclusions from them.

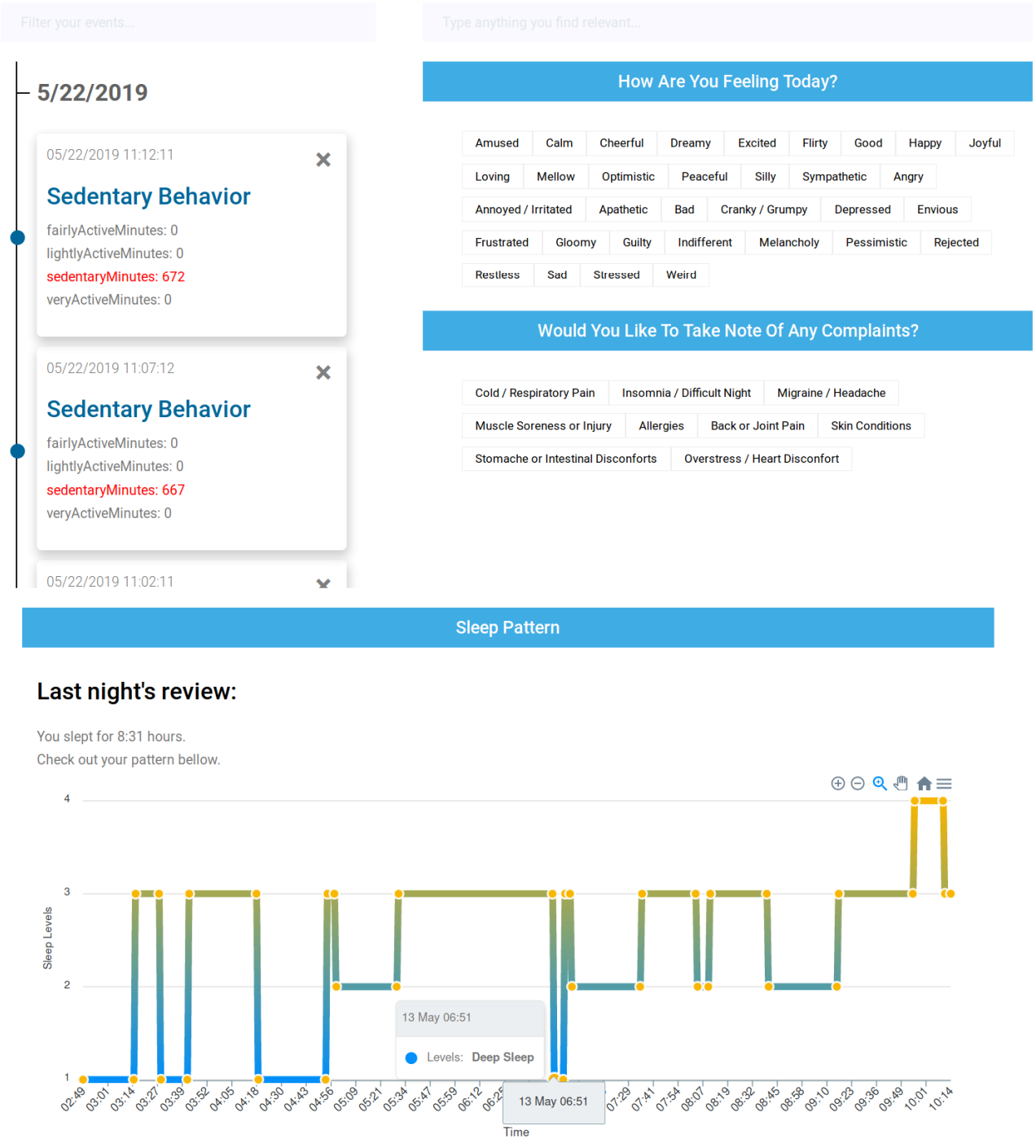


Figure 14: Screen capture of a regular user's *home* page.

Note: the screen captures taken from the web application exclude several page components such as page banner and footer to avoid presenting irrelevant information in this report.

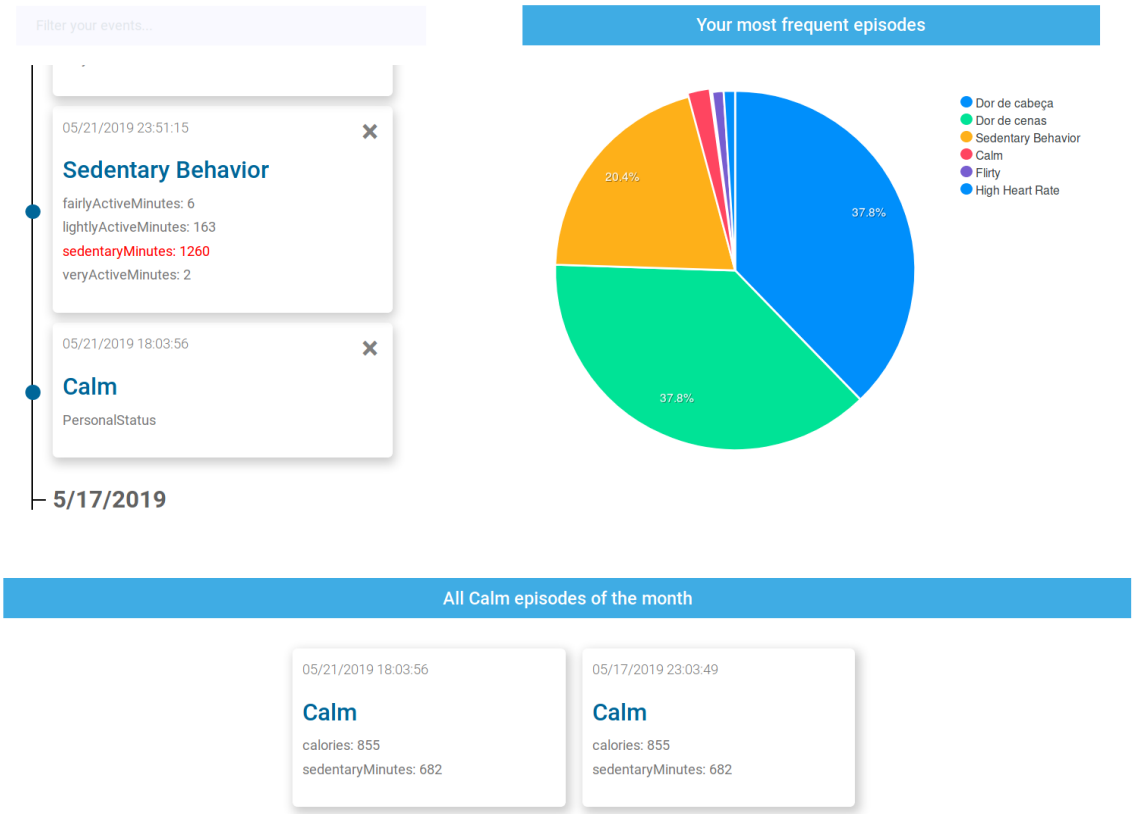
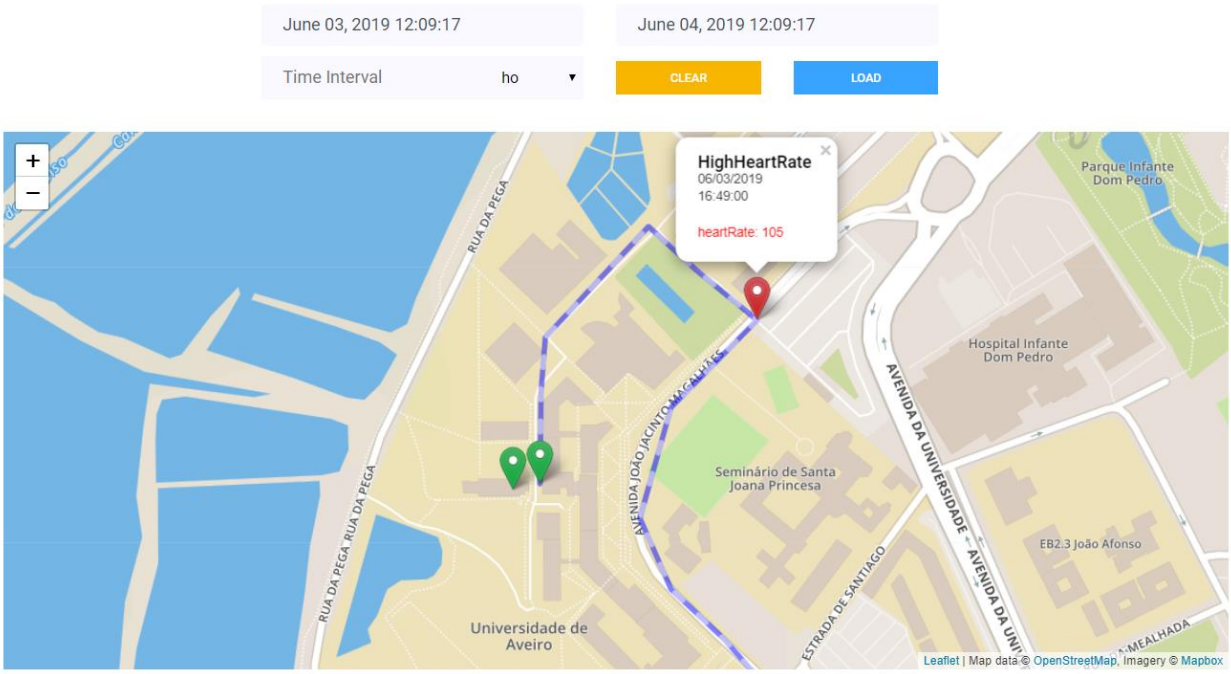
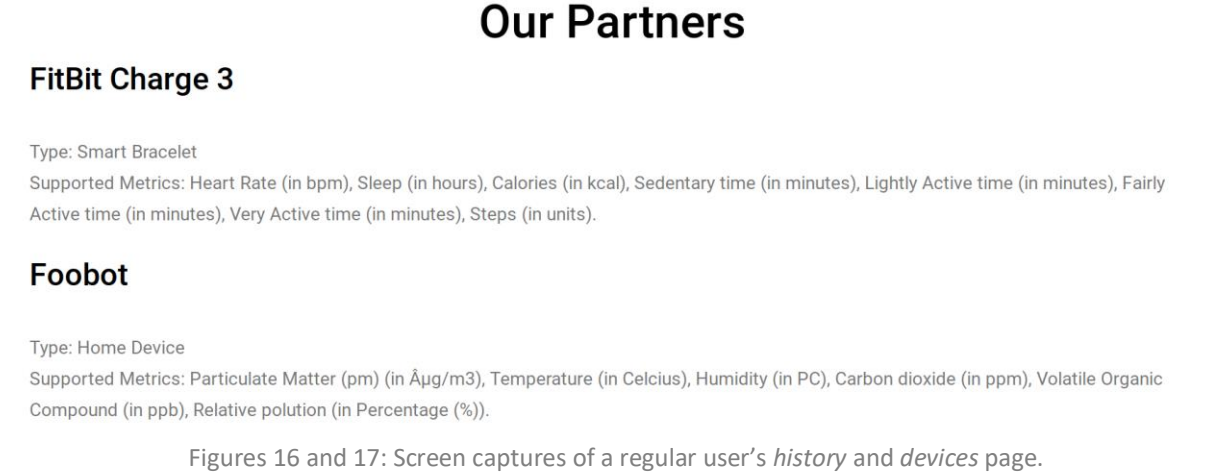


Figure 15: Screen capture of a regular user's *events* page.

The *history* page is where the geolocation is put to practise. The utility of the previous visualization methods for historical data is extended to the spacial dimension through the use of a world map containing the tracking records of the user.
The geolocation is tracked throught the mobile app (more on this in the next section) and serves as a form of understand the regions more usually visited by the user that have a greater occurrence of events. By placing a marker for each event on the map's user path, a person can follow their development and distinguish them by a color scheme (red markers mean unhealthy events, while blue and green mean user inputs and device locations respectively). The default search is for a period of time of one month, but the user is free to search for his/her path for any period of time desired.

What about the addition and removal of devices from the account? ContinuousCare makes this as simple as possible through the *devices* page. When in it, the page requests the information about all currently supported devices and the metrics they process.
Above their description, the user is presented with a plus sign and his/her devices already synchronized with the server (if any). All information about each device is presented when the respective box is clicked and updateable at any time. The plus sign is used to add new devices to the account.
To add a device, the user needs to fill in the small form presented once the box is clicked and after following a few steps explained in the mobile app (see next section for more information regarding these configuration steps). These configuration steps are necessary as we could not have made any sort of partnership with the devices' owners and therefore need to collect the required information from their services manually.





Figures 16 and 17: Screen captures of a regular user's *history* and *devices* page.

Finally we reach the *permissions* page, responsible for connecting the regular users with the doctor accounts. The ability to temporarily transfer access permissions to a certified professional in the medical field brings a new paradigm to the data analysis of what the system collects and aggregates – it makes ContinuousCare not only a personal monitoring tool, but also a valuable asset for medical consultations.
With this in mind, we provide the same set of visualization features for both user types and a few more for the doctor accounts with a more technical approach (less interesting for regular users). These are made available for a doctor through the request or grant of a read-only access permission. Users are notified in real time or after the login process when they receive a request or acceptance message, and doctors may use the access grant for only the amount of time allowed by the user. Doctor accounts are capable of having several access grants running simultaneously and can close them at any time.

The remaining web pages cover the profile settings (on the *settings* page), the information regarding ContinuousCare in general (on the *about* page) and the contacts area for giving the user a way of sending an email with any complaints or help requests or contacting us through another communication method (on the *contact* page).

The doctor's account is quite different in functionality but not so much in looks. The theme switches from light blue to soft orange and the number of tabs on the navigation page is fairly reduced.
Doctors have the ability to request access permissions to users by either username or PHPN and, once granted, analyse data through a set of tools, including those that regular users also have and one reserved for medical accounts that requires greater comprehension skills but allows a better understanding of health-related issues. The figure on next page ilustrates the use of the permissions by a user with a doctor account.

The design of the website was based on a template from Colorlib.com (22), where an appropriate visual basis was freely available. However, much was changed as the content was of a general purpose and had no well defined identity. The code suffered a second round of changes when adapting it to the NuxtJS framework.
All external components helped defining the visual concept for the website, with our own adaptations to the styling for a better UX. They were also responsible for the support of many of our features.
The main components were: axios (23), vuex (24), vue-apexcharts (25), nuxt-leaflet (26), vue-native-websocket (27). Followed by a few others like bootstrap-vue, js-cookie, leaflet-ant-path, vue-bootstrap-datetimepicker, vue-js-modal, vue-notification, vue-toasted, vuejs-datepicker, vuex-persistedstate, etc.

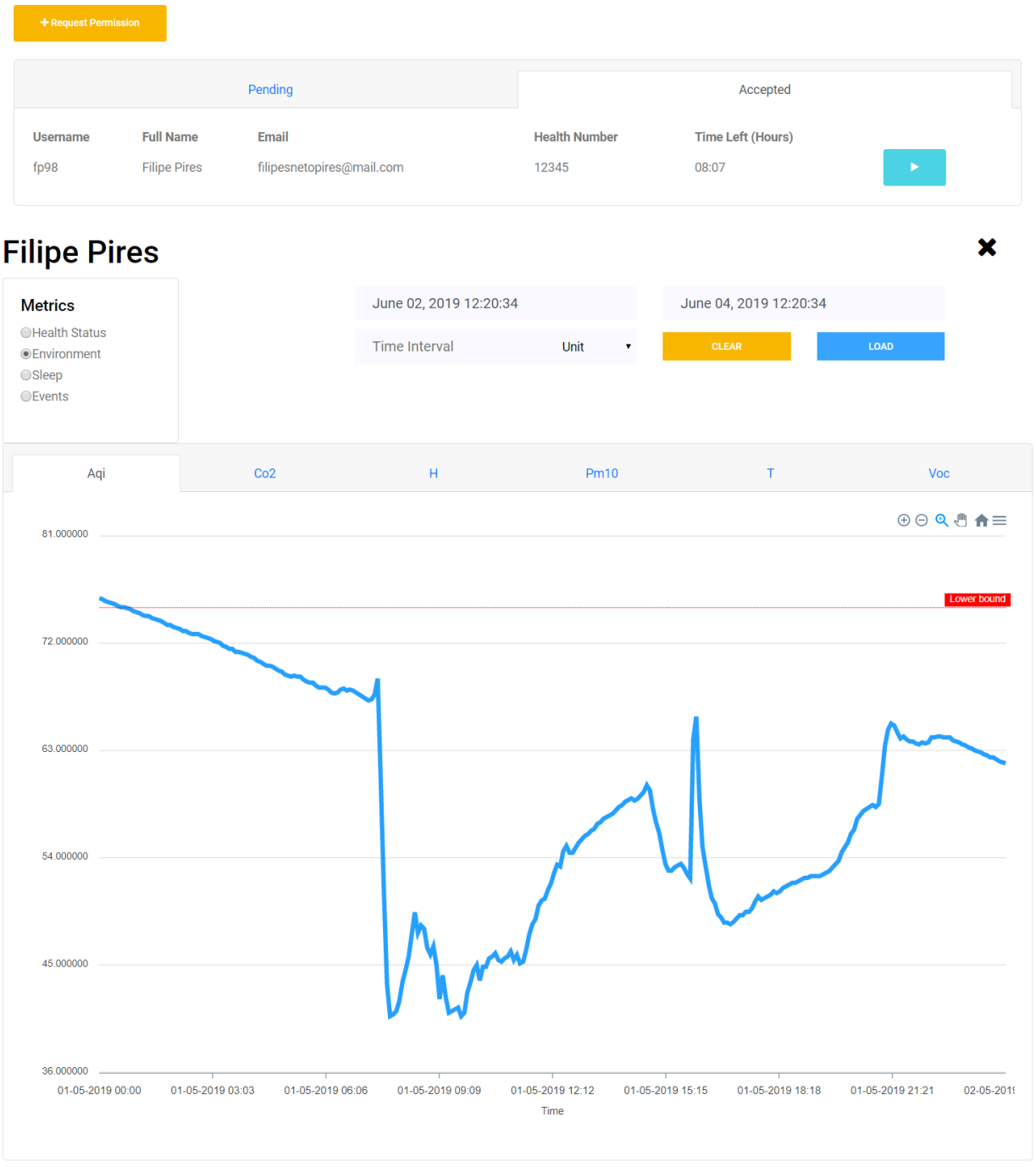


Figure 18: Screen capture of a doctor's *patients* page.

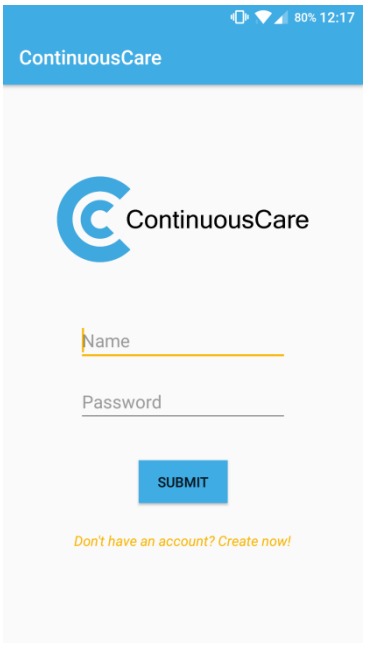
3.1.3. Mobile Application

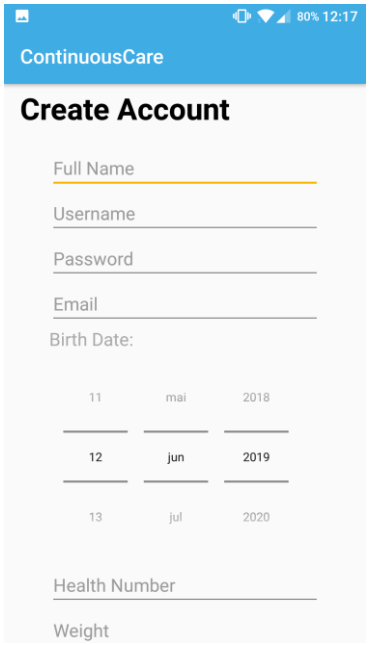
The ContinuousCare mobile application was introduced as a solution to the lack of valuable resources. Our environment data sources had the main purpose of bringing more value and completeness to the health-tracking features we offer, but without the notion of the users' location no trustworthy studies could be executed from this type of information – connecting an event related to the air quality that a user is exposed to with an actual notion of where the event occured was crucial to allow the extraction of conclusions from these events. Here is were our mobile app came in to place.

As none of the supported devices had a location tracking feature or GPS, the use of a phone app would allow us to keep track of the user's geolocation on the occurence of meaningful events. Our previous research showed that most of the healthcare systems already assume the sharing of location data, and our bracelet device also demanded the installation of an application with access to the user's location, so we came to the conclusion that adding this imposition would not affect much the user experience.

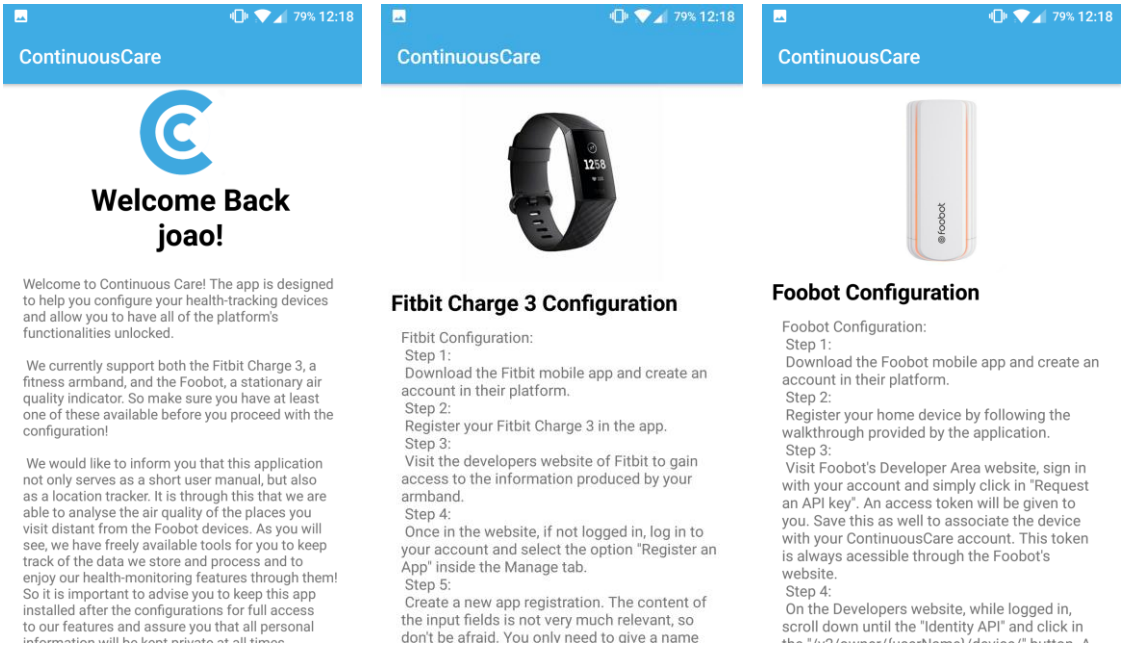
However, to avoid presenting a product with little use, we decided to increase the utility of the app by making it a simple and always-available tutorial explaining how a user can setup the system for him/herself.

Following we present screen captures of the final result of our mobile application. Any user that installs it is explained that the presence of the app in his/her smartphone is important for the correct functioning of his/her account, and receives instructions to follow along with the connection with the health-tracking devices to the account.





Figures 19 and 20: Mobile app registration and login views.



Figures 21, 22 and 23: Mobile app information views.

3.2. Deployment Instructions

The deployment of our system is made simple to anyone who desires to try it out. Every components of our platform, mentioned on section 2.3, was placed inside docker containers (28), having each containers its Dockerfile with everything specific to each component already configured. To launch CC, we used docker-compose - that is in charge of launching each components on the respective container. Therefore, the deploy of our platform only needs docker and docker-compose installed on the machine where the deployment will be done.
The first step is to clone the code repository (from code.ua) from the provided link:

https://code.ua.pt/git/personal_monitoring_system

Then just execute the command “docker-compose up” on the directory where the docker-compose file is (in our project, it is in the root directory). Now all components should be up and running. On the following table is presented the respective path for each component on port 8081.

Component	Path
Website	/
REST API	/api
GPS Module	/gps
Permissions Websocket	/websocket

3.3. Developers Guide

With the manner we implemented our health monitoring service, actions such as adding a new source of information, or a new metric to an existing supported device or even updating an already existing metric becomes as easy as changing a few lines in a configuration file.
Following are the procedures for each case.

3.3.1. Adding a New Supported Device

Accessing the Server folder, there is a file named devices.py. This file is where all the configurations for the devices we support are defined. The file follows a fixed schema, first defining the device and followed by all the metrics we want to extract from it. The configurations are written in a python class that extends one of the previously mentioned abstract classes: "DataSource" for a device or external API; "Metric" for the metric itself.
Adding a new device implies that you need to create a new class that extends "DataSource" with the name of your device, defines it's URL, headers, refresh information if needed and that's it; then you can create as many metric classes as you think necessary, making each one an extension of "Metric".
Afterwards you need to add the device to the database definition so that the entire system takes in consideration the new source. You need to add it in the "supported_device" table, the metrics it can export and the connection between device and metric in the "metrics" and "supported_metric" tables.
The only constraint when creating a new device is that the name of the class representing the device must correspond to the name inserted in the database - be careful with this. After all, the system will start processing the new source and you may start adding the frontend support to it, if necessary.

3.3.2. Updating an Existing Device or Metric

This operation is a lot simpler than the last. The only necessary changes occur in the devices.py file present in our Server folder. If the changes are related to URLs, headers and data formats, you can quickly find the spot where it is configured and alter it as you wish.
In case some some feature is no longer necessary you can simply delete the configurations, being only necessary to follow the defined abstract class protocol.

Conclusion

The purpose of this project was not only to develop a valuable product for the end-users, but also to gain knowledge on the entire development cicle of an engineering team and on several fields of software development. We set out to produce a proof of concept for a personal monitoring system and deliver one with potential to do more.

ContinuousCare presents itself as a healthcare tool for any person, empowering both people concerned with their own health and medical personnel looking for innovative analysis tools. It combines both the sources of personal devices and of air quality monitors, and it makes all data collected easy to read, process and feed to other systems, always assuring the privacy of each user.
Our web application sums a total of 7 maiin features and presents data from close to 20 metrics. The ContinuousCare API sums a total of 26 public end-points. Our back-end reaches a maximum capacity of 100 users accessing the main server at the same time, while ensuring all of the performance requirements established.

The final product leaves unresolved the complexity of the configurations needed for deployment, as no accord was done with the supported devices, though its impact was reduced with the help of the mobile app user guide. The lack of an additional user interface for the data extraction from the public API for other softwares also limits the usability of this data, as its format is very specific and optimized for our own applications. This shortcoming is also attenuated by the fact that all the documentation needed is delivered along with this document.

The efforts to be done after the delivery date will focus mainly on a new system module for data mining aiming to extract key information from events and offer tools to predict diseases with relatively good confidence and notify users of dangerous behaviours and hazardous environments.

Bibliography

1. **Costa, Carlos.** Sistema Pessoal de Monitorização de Condições Clínico-Ambientais . *Elearning, Universidade de Aveiro, Projeto em Informática.* [Online] 2019. https://elearning.ua.pt/pluginfile.php/970218/mod_resource/content/1/13%20-%20PersonalMonitoring.pdf.

2. *Health monitoring systems using IoT and Raspberry Pi — A review.* **Pardeshi, Vivek, et al., et al.** Bangalore, India : IEEE, 2017. 978-1-5090-5960-7.

3. *Angel-Echo: A Personalized Health Care Application.* **Ma, Mengxuan, et al., et al.** Philadelphia, PA, USA : IEEE, 2017. 978-1-5090-4722-2.

4. *An Android Application for Geolocation Based Health Monitoring, Consultancy and Alarm System.* **Tartan, Emre Oner and Ciflikli, Cebrail.** Tokyo, Japan : IEEE, 2018. 978-1-5386-2667-2.

5. **Daniel, Florian, et al., et al.** *Beyond Health Tracking, A Personal Health and Lifestyle Platform.* Trento : IEEE Computer Society, 2011. 1089-7801/11.

6. **Chu, Yu-Hsien, et al., et al.** *UPHSM, Ubiquitous Personal Health Surveillance and Management System via WSN Agent on Open Source Smartphone.* Taiwan : IEEE Computer Society, 2011. 978-1-61284-697-2/11.

7. Fitbit Charge 3, Advanced Health and Fitness Tracker. *Fitbit.* [Online] https://www.fitbit.com/eu/shop/charge3.

8. Foobot, the air quality monitor you need. *Foobot.io.* [Online] Airboxlab. https://foobot.io/features/.

9. World's Air Pollution: Real-time Air Quality Index. *Waqi.* [Online] https://waqi.info/.

10. MySQL 8.0 Reference Manual. *Dev MySQL Documentation.* [Online] Oracle. https://dev.mysql.com/doc/refman/8.0/en/.

11. InfluxDB 1.7 documentation. *Docs Influx Data.* [Online] https://docs.influxdata.com/influxdb/v1.7/.

12. Table of Contents. *GEvent Docs.* [Online] http://www.gevent.org/contents.html.

13. NGinX Documentation. *NGinX.org.* [Online] https://nginx.org/en/docs/.

14. websockets 7.0. *PyPi.org.* [Online] Python Software Foundation. https://pypi.org/project/websockets/.

15. influxdb 5.2.2. *PyPi.org.* [Online] Python Software Foundation. https://pypi.org/project/influxdb/.

16. mysql-connector-python 8.0.16. *PyPi.org.* [Online] Python Software Foundation. https://pypi.org/project/mysql-connector-python/.

17. Documentation for Application Developers. *Android Developer.* [Online] Google. https://developer.android.com/docs.

18. VueJS Documentation. *VueJS.org.* [Online] https://vuejs.org/v2/guide/.

19. NuxtJS Documentation. *NuxtJS.org.* [Online] https://nuxtjs.org/guide.

20. Documentation. *Swagger.* [Online] SmartBear. https://swagger.io/docs/.

21. User Guide. *Support.* [Online] Proto.io. https://support.proto.io/hc/en-us/categories/202758988-User-Guide.

22. Hospice Template. *Colorlib.* [Online] 09 2018. https://colorlib.com/wp/template/hospice/.

23. Axios Module. *Axios NuxtJS .* [Online] https://axios.nuxtjs.org/.

24. Vuex. *Vuex VueJS.* [Online] https://vuex.vuejs.org/.

25. Vue Charts. *Apexcharts.* [Online] https://apexcharts.com/docs/vue-charts/.

26. Nuxt Leaflet. *GitHub.* [Online] https://github.com/schlunsen/nuxt-leaflet.

27. vue-native-websocket. *NPMJS.* [Online] https://www.npmjs.com/package/vue-native-websocket.

28. Documentation. *Docker.* [Online] Docker Inc. https://docs.docker.com/.

29. **Lu, Ssu-Hsuan, et al., et al.** *Pervasive Health Service System, insights on the development of a Grid-based personal health service system.* Taiwan : IEEE Computer Society, 2010. 978-1-4244-6376-3/10.

Appendix A

Table of default moods a regular user can choose to describe his current emotional/physical state. It is important to notice that we do not limit his/her choice and allow any client to write down and submit a personalized mood or any note he/she finds relevant.

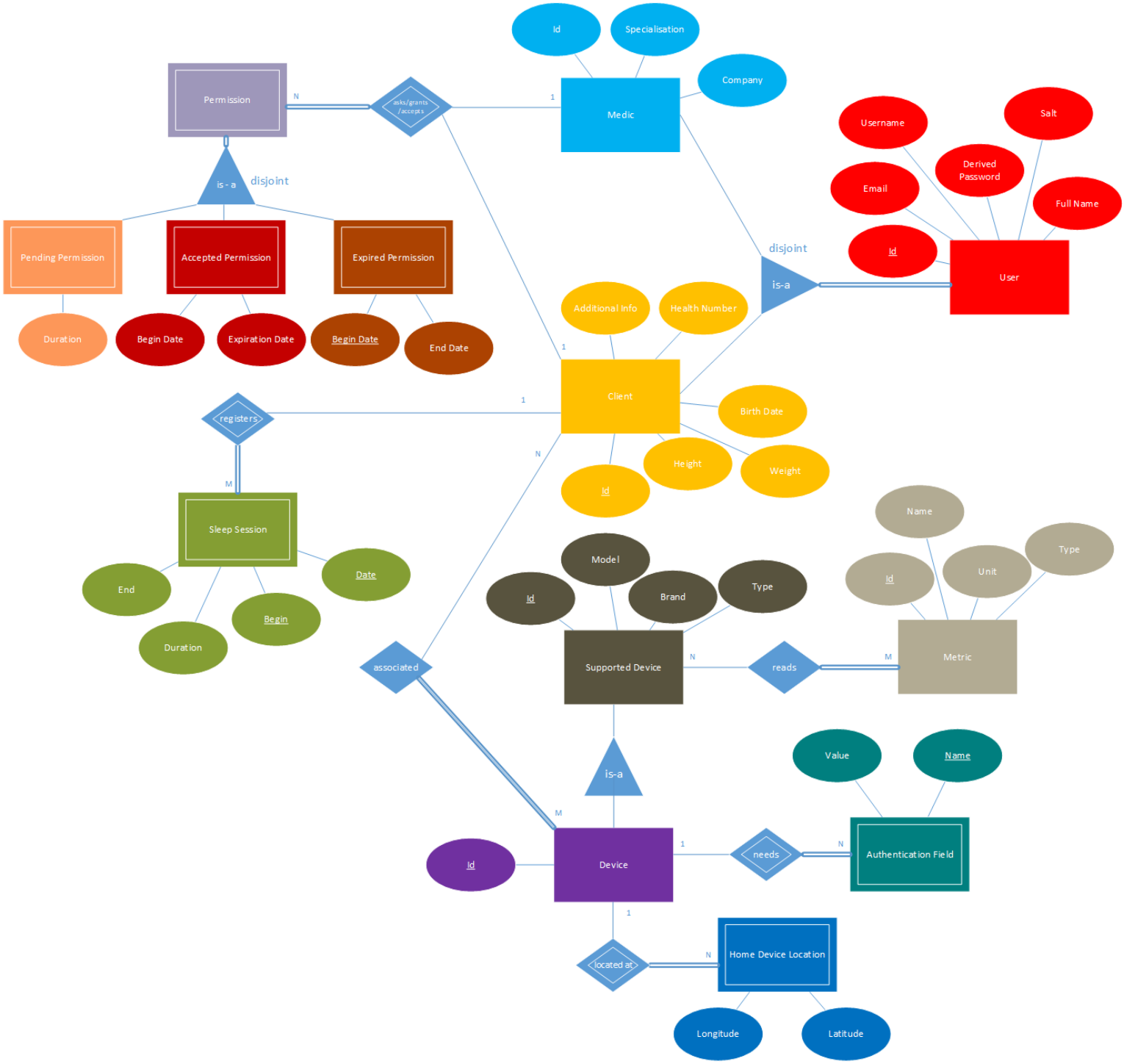
Positive Moods	Negative Moods	Health Issues
Amused	Angry	Cold / Respiratory Pain
Calm	Annoyed / Irritated	Insomnia / Difficult Night
Cheerful	Apathetic	Migraine / Headache
Dreamy	Cranky / Grumpy	Muscle Soreness or Injury
Excited	Depressed	Allergies
Flirty	Envious	Back or Joint Pain
Good	Frustrated	Skin Conditions
Happy	Gloomy	Stomache or Intestinal Disconforts
Joyful	Guilty	Overstress / Heart Disconfort
Loving	Indifferent	
Mellow	Melancholy	
Optimistic	Pessimistic	
Peaceful	Rejected	
Silly	Restless	
Sympathetic	Sad	
	Stressed	
	Weird	

Appendix B

Web application prototype, created with Proto.io.

Appendix C

Entity-Relation model of ContinuousCare's storage implementation.



Relational model of ContinuousCare's storage implementation.