FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Web Platform for Monitoring Field Trials

Pedro Daniel Viana Lima

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Correia Lopes

Co-Supervisor: Jorge Miguel Neves Ribeiro

July 17, 2019

Web Platform for Monitoring Field Trials

Pedro Daniel Viana Lima

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Doctor Sérgio Sobral Nunes

External Examiner: Prof. Doctor Maria Benedita Campos Neves Malheiro

Supervisor: Prof. Doctor João Correia Lopes

July 17, 2019

Resumo

Os testes de campo são normalmente destinados à validação de produtos num ambiente do mundo real, onde um produto é testado pelos utilizadores finais num contexto real, e não sob condições artificiais, em que geralmente estes testes são complementados com um procedimento de monitorização por parte dos investigadores. Mesmo que pareça uma tarefa simples, monitorizar o desempenho dos participantes ou do produto em si, pode acabar sendo uma tarefa difícil no caso de ausência de ferramentas essenciais (por exemplo, uma visualização clara dos dados) ou considerando que, o acompanhamento dos participantes todos os dias não é de uma maneira prática nem eficiente de como fazê-lo.

O procedimento normal de monitorização destes testes de campo envolve: em primeiro lugar, a recolha de dados referentes a cada utilizador, sendo esta obtida através de um vasto número de hipóteses, desde questionários diários / semanais a sensores. E em segundo lugar, a visualização e interpretação das informações armazenadas. Assumindo que a recolha de dados já é realizada com sucesso, ainda é necessário proceder à sua visualização. Como resultado disso, e considerando a web como a plataforma alvo, dado que esta concede acessibilidade, bem como a simplicidade no uso e desenvolvimento, uma nova aplicação web seria exigida sempre que um teste de campo fosse iniciado.

De forma a superar este problema, a solução deve envolver uma aplicação capaz de representar e visualizar dados. Assim, o objetivo principal deste trabalho é criar uma aplicação com a capacidade de adaptação a projetos com natureza de contexto distinta e que, como resultado final, possa apoiar investigadores no processo de monitorização de testes de campo a um ritmo mais rápido. A solução proposta segue uma arquitetura de 3 camadas, incluindo a definição de um componente intermediário — o mediador, responsável por oferecer serviços através de uma API ao cliente, enquanto solicita dados do fornecedor de base de dados. O mediador reconhece informações relativas à base de dados, como informações de conexão, estrutura de dados e também sobre a organização dos componentes como parte das interfaces do usuário, através da definição de uma configuração baseada em YAML (YAML Ain't Markup Language).

A validação da aplicação encontra-se dividida em duas etapas: a primeira, com o objetivo de validar as interfaces de usuário sendo esta realizada com o auxílio de um teste de usabilidade onde foram sujeitos um total de cinco participantes. O último, tendo em mente a validação de todo a aplicação, ao aplicá-lo a dois casos de estudo diferentes. A partir da análise dos resultados, em termos da interface projetada, estes foram bastante satisfatórios embora tenham sido identificados pequenos problemas com o design. Relativamente aos casos de estudo e tendo em mente o facto deles cobrirem a maior parte das funcionalidades, fomos capazes de provar que, com o trabalho desenvolvido, é possível construir uma aplicação a partir de uma configuração pré-definida tendo por base um mediador que permite a integração dos diferentes componentes e que estes funcionem corretamente.

Abstract

Field trials are commonly intended to validate products in a real-world environment where the product is being tested by the end-users in a real-life context rather than under artificial conditions, usually supplemented with a monitoring procedure from researchers. Even though it can seem a simple task, monitoring the performance of either the participants or the product itself can end up being a challenging task if there are essential tools missing (e.g. a clear visualization of the data) or considering that following up on them everyday is not a practical nor an efficient way on how to do it.

The normal procedure of monitoring these field trials always involve: firstly, the collection of data related to each user which can be achieved through a vast number of ways from daily/weekly questionnaires to sensors. And secondly, the visualization and interpretation of the information stored and transformed. Assuming that the collection of data is already being done successfully, there is still the need of displaying it. As a result of this, and considering the Web as the target platform, mostly because it grants accessibility as well as the simplicity in use and developing, a new Web application would be demanded whenever a field trial starts.

To overcome this problem, a solution must involve an application able to represent and visualize data. Hence, the main goal of this work is to create an application with the endowed capacity of being adapted to a heterogeneity of projects and that, as an end result, can support researchers in the process of monitoring field trials in a faster time pace. The proposed solution follows a 3-tier architecture, including the definition of an intermediate component — the mediator, responsible for offering services through an API to the client while requesting data from the database provider. The mediator acknowledges information relative to the database such as the connection information, data structure and also, about the organization of the components as part of the user interfaces, through the definition of a configuration based on YAML (YAML Ain't Markup Language).

The validation of the application is divided into two stages: the first aiming to validate the user interfaces and accomplished with the aid of a usability test comprising a total of five participants. The last, having in mind the validation of the whole application by applying it to two different case studies. From the analysis of the results, in terms of the designed interface, they were satisfactory although with a few minor design problems identified. On the other hand, considering the case studies and that they covered most of the features, we were able to prove with the work developed, that it is possible to building an application from a predefined configuration, having a mediator enabling the different components to work seamlessly.

Acknowledgements

First and foremost, I would like to thank my supervisors, João Correia Lopes and Jorge Ribeiro, whose expertise, guidance and support were invaluable during the whole thesis process. I can gladly say that all the tools and conditions, that made it all possible, were provided by you as well as the encouragement to push it forward.

Inevitably, to the colleagues at Fraunhofer Portugal for embracing me and for creating an excellent work environment where I felt part of. Among them, a special thanks to the volunteers of the usability tests, namely: Cristiana Braga, Nuno Cardoso, João Almeida, Ana Sampaio and Marcos Liberal, for your time and useful feedback. I am also extremely thankful for the opportunity to participate in the workshops offered, undoubtedly, I received way more than I could ever expect to give in return.

I would also like to acknowledge my friends, those I made along the way and those who were there since day one, for all the support and all the joy, happiness and peace of mind they all brought, always. For that, I am forever grateful.

To my family, blood or not, for always being there when I needed the most, for your wise counsel but also, for your comprehension. To my father, for raising me as a man of nothing but values and principles. Even knowing you would never see or share my success, it was for you I made it all along.

My sincerest gratitude to all of you.

Pedro Lima

‘Se puderes olhar, vê.
Se puderes ver, repara.”

José Saramago

Contents

List of Figures

List of Tables

Abbreviations

AICOS	Assistive Information and Communication Solutions
CMS	Content Management System
CRUD	Create, Read, Update and Delete
DOM	Document Object Model
EML	Ecological Metadata Language syntax
GDPR	General Data Protection Regulation
HCI	Human-Computer Interaction
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IE	Internet Explorer
InfoVis	Information Visualization
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
MVC	Model-View-Controller
OOP	Object-Oriented-Programming
ORM	Object Relational Mapping
REST	Representational state transfer
SPOF	Single Point of Failure
SQL	Structured Query Language
SUS	System Usability Scale
TTL	Time-To-Live
UI	User Interface
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Chapter 1

Introduction

Whether it is in clinical trials, nutrition or health-care solutions, graphical visualizations are usually presented to us with an intended explicit message. Such visualizations when complemented with contextualization and critical analysis by the observer, can support researchers in the process of converting raw data into pertinent information. Nowadays, it is even more important as the data being stored increases every year [34]. As IDC mentioned "Data is growing at a rapid pace. By 2020 the new information generated per second for every human being will approximate amount to 1.7 MB" [38]. And, as they also stated, only 0.5 % of the all accessible data around the world is being analyzed and used. This reveals an enormous amount of data and the potential that is still unfolded and topics such as data visualization are becoming more vital these days.

In research fields, where data is collected and stored, we face the same problems aforementioned. According to a survey lead by *MIT Sloan Management Review*, getting the data is commonly not the biggest obstacle that companies face as only one out of five respondents considered it as the primary obstacle [45]. Although, the lack of perception and insight about that same data and neglecting its potential would possibly lead to poor decision making when designing a new product, for instance. Also, monitoring a product or people and being able to gain a better understanding of them can prove crucial. Applying existent techniques conjugated with computational systems could slightly improve the way we absorb information.

1.1 Context

Field trials are a common practice among research-oriented institutions that validate their products in a real-world environment. Such practice can act upon a large spectrum of areas such as road traffic [26], psychiatry [30] and specially, in Human-Computer-Interaction (HCI) fields — studying the interaction with technology [9, 4], for instance using sensors for collecting bio-information [46]. And it is frequently used and applied to either a prototype or a completed product in order to provide an insight into the strengths and weaknesses of the product itself. Commonly, it can be

complemented with a monitoring procedure of the participant performance and product use, for instance, when new data was logged, which features were used according to technical reports sent from their devices.

The process of collecting data can use several distinct tools, from regular end-user surveys, interviews, observations and user reports made through the device. Also, the type of data collected may be completely different based on the project being applied to and its context (e.g. a project may require sensors, thus collecting data from the surrounding environment, while another emphasis the user interaction and aims to test the designed usability). Nevertheless, regardless of the topic, the objective settles a common ground that is to extract valuable information and that consequently, can have a direct impact on the final product shape, design and usability according to the end-users expectations. That is why it is so important in some cases that a product should be evaluated in a real-world environment before its commercial launch to the public.

Field trials can generate large amounts of data as a consequence of the required proceeding of collecting and storing data throughout the whole period of time. This, coupled with the necessary analysis by researchers working at Fraunhofer Portugal[1], make field trials so important and a great source of useful information.

1.2 Problem Definition

When conducting field trials with end-users, it is a challenging process to keep track of how well participants are doing and the performance of the product that is being tested. With that being said, following up everyday is not a practical nor an efficient way to do it, and issues can end up being imperceptible during the trial. Although, even when there is substantial data available, that same data are represented in a typical data structure, making it quite difficult to understand it. Thus, researchers are not able to extract valuable information from the collected data, as they desire, regarding the structure and format of the data stored, making the task of monitoring field trials almost impossible. Following that, it is clear that there are missing tools that could be addressed having in mind topics such as Information Visualization, Machine Learning (Data Mining), etc.

Such problem demands the use of well-established data visualization techniques developed during the past years, through a Web platform. The main purpose of its use is to bring to the end-users of the platform — the researchers — a deeper understanding of the data, granting them a wider knowledge and contextualization of the problem [2]. It is also important to notice that those visualizations still require an in-depth analysis and understanding [36], involving the researcher in a search for a result rather than present the results just as John Light described in “Portable Document Indexes” [47, 64].

Nonetheless, it standalone would not totally solve the problem due to the fact that a new Web application would be demanded whenever a new field trial began which consequently, leading to increased costs associated with the re-design and adaptation of the past platform. Hence, a need

[1] Company where this work took place

for a system endowed with both the capability of visualizing such data in an intuitive way and that could serve as a platform for representing data from multiple and independent projects emerged.

This problem, just like every other problem, raises a necessity for a solution and therefore, this one in specific originated the conception of this dissertation.

1.3 Motivation

With the exponential growth of data as a result of being collected in various ways, data requires to be processed and analyzed in order to extract value from it. Fraunhofer Portugal, in the past, in pursuance of addressing this issue, designed and developed a Web tool as part of the Clockwork project[2]. The main purpose of it was to help the researchers keep track of how participants were using the system. However, as they later mentioned, the solution taken, despite being useful for that project, was an ad-hoc solution. Since it was not simple to extend it to other projects without modifying a considerable amount of code, it would lead to significant costs by having to build up an application whenever there was a new field trial starting.

Thereby, the main motivation for the elaboration of this work is to help and support researchers in the proceeding of monitoring field trials by turning it into a way more simpler procedure by: (i) reducing the impact of starting a newer project; (ii) enabling researchers to follow participants from the beginning of the trial; (iii) reducing the time expended on the project set up and consequently allowing an increased focus on the analysis of the data.

1.4 Objectives and Contributions

As it is beyond doubt from the analysis of the problem described, the main objective affiliated with the realization of this dissertation is the conception of a Web platform for representing data from heterogeneous contexts, granting the possibility of adapting to whatsoever project being submitted to and the scope of it. Therefore, supporting the researchers in the process of monitoring field trials, tracking its participants and the performance of the product being tested.

The main contribution of this work comes in the form of a tool for visualizing data that can be applied to several field trials requiring, for that effort, the definition of a configuration, as we will see later on. Also, as an implicit contribution, it is expected to improve the monitoring of field trials by a faster adaptation without coding, granting researchers the possibility to track their end-users from early stages and identify problems faster than the conventional methods used until then.

1.5 Document overview

The remaining of the document is organized into seven chapters. By definition, each chapter was written to be largely self-contained.

[2]Self-care management system for supporting shift workers in their everyday life

Chapter 2, titled “State of the Art” presents a literature review of related work upon which our research draws, consisting in an overview of the topic of information visualization, data characteristics, their classification and the data visualization techniques developed, during the years, to represent it. Also, it covers the state of configurable interfaces and a comparison between existent tools are made.

Chapter 3, titled “Problem Statement and Solution Proposal” explains the problem in-depth, the requirements associated and describes the architecture of the approached system.

Chapter 4 title “Mediator” includes the explanation of the most important component from the proposed solution architecture, a broker from databases to visualizations.

Chapter 5 title “Dashboard” provides information about the path that led to the design choices of the user interfaces conceived, the main functionalities it offers and at last, an analysis of the results from the usability test applied.

Chapter 6 title “Case Studies” presents the main functionalities tested through the application of two distinct case studies and discusses the results obtained.

Chapter 7 title “Conclusions and Future Work” aims to describe the main findings of this work and what could be addressed in the future.

Chapter 2

State of the Art

The purpose of this chapter is to provide an overview of the background of information visualization, followed by more specific domain information such as the existing data visualization techniques, the different data types, the taxonomy for graphical representations and how can all of those be implemented recurring to established frameworks and JavaScript libraries considering the web as the platform. In addition, it is addressed the topic of metadata-driven approach as a path to achieve a generic UI.

We conclude by presenting the related work on the area, as well as summarizing the topics discussed throughout this section.

2.1 Information Visualization

Information Visualization (InfoVis) is intrinsically related to human visual perception, behaviour and interaction and is "one of the most promising approaches that investigate the entire pipeline of search, exploration, and analysis"[52, 61]. The principal reason behind it, is the leveraging of cognition through those human perceptions to achieve a state of wisdom acquisition, based on few steps: (i) The conversion of raw data into information by understanding relations between them (physical structuring); (ii) The conversion of the resultant information into knowledge by identifying patterns (cognitive structuring); (iii) And lastly, the transformation of the extracted knowledge into wisdom by understanding the principles that support them (belief structuring). However, software systems that follow an approach according to Information Visualization concepts are best suitable for exploratory tasks, involving browsing a large set of data. Especially because, as explained before, its main objective is to increase cognition, for instance, perceptual speed, visual working memory and verbal working memory [11, 61], which grants a better proficiency on identifying new patterns, make new findings, gaining a better insight about it and ask better questions. [42]

InfoVis treats data belonging to an abstract domain and uses intimately, a graphical representation for such data. Graphic displays can serve as an excellent tool for the exploratory analysis but also for information comparison and presentation of results. Depending on the purpose of either presentation or exploration, there should be evident differences in both form and practice. For instance, a presentation graphic should be static and exhaustive, containing the definitions and explanations of the variables as a support for a conclusion, while exploratory graphs are more informative and do not need to be as detailed as possible [13].

According to Edward R. Tufte's [62] and the article *Interactive Information Visualization of a Million Items* [25], InfoVis principle "is to map the attributes of an abstract data structure to visual attributes such as Cartesian position, color and size, and to display the mapping". This is the opposite of scientific visualization, which deals with physical data (with representation in space), with intrinsic visual representation and relations.

Throughout the years, researchers with expertise in the field of Human-Computer Interaction and Information Visualization have been designing and developing unique visual techniques and improving others. Yet, it is still difficult to evaluate those methods regarding the enormous time and cost consuming of those evaluations procedures. [52].

2.1.1 Data Characteristics and Types

According to the book *Information Visualization in Data Mining and Knowledge Discovery* [23], data has two characteristics: nature and domain. Nature can be divided into two possible alternatives: stable — in case the data will not change over time (therefore static), and dynamic — consisting in the opposite scenario, where the data needs to be updated constantly. Based on the domain it can be nominal or ordinal (qualitative) and discrete or continuous (quantitative).

Nonetheless, the visualization can either be classified as stationary, animated or interactive as aforementioned. In case it is stationary, the graphical display will remain the same, if it is animated, as the name implies, the graphical display will be animated which is very useful for simulations since it is more engaging and helps in the data analysis process. Finally, in case it is interactable, the user is able, for instance, to select and filter in order to reduce the data size that his working on.

The visualized data belong to one of several types depending on the dimension, it is defined in conformation to the number of variables or attributes, and frequently differ from one another. The data sets may be classified in one-dimensional, two-dimensional, multi-dimensional, text and hypertext, hierarchies and graphs, algorithms and software [41].

2.1.1.1 One-dimensional

This type refers to data that has only one dimension, typically related to temporal data. An example is time-series of purchases from a store, where each time can be associated with one or multiple variables.

2.1.1.2 Two-dimensional

Two-dimensional data, as the name suggests, has two different dimensions. This kind of data is commonly represented by X-Y plots (e.g. scatter plots, geographic maps) and establish a relation between those variables. The problem of such simple representations is that when dealing with large data sets, "axes and maps get quickly glutted" [41] and can directly interfere in the readability and understanding of such displays.

2.1.1.3 Multi-dimensional

When the data set is composed by more than two variables or attributes, not allowing a simple visualization from "flatland" domain (e.g. two dimensional plots) [62] or exceed the 3D domain as well, it requires more complex graphical representations. Examples of it and one of the most common, are the data sets usually represented on databases, constituted by multiple relationships and attributes between tables [41].

2.1.1.4 Text and Hypertext

There are some data types in which standard visualization cannot be applied because of their raw form is part of the text domain and not numerical. For instance, the "text and hypertext as well as multimedia web page contents" [41].

2.1.1.5 Hierarchies and Graphs

Sometimes data is not simply an aggregation and collection of variables, as there are records that can have a relationship with others information. Consequently, they can be structurally organized into hierarchies and graphs [37]. Graphs consist of a set of relationships, called edges, between objects, designated by nodes. Such properties turn them useful for these cases [41].

2.1.1.6 Algorithms and Software

Lastly, this kind of data appeared recently with the development of computer systems. Its main purpose is to aid in the process of conceiving new software by visualizing the flow of information in a program, the structure through hierarchical or graph representations. Besides, it can even help programmers debugging the code [41].

2.1.2 Classification

Data visualizations techniques can be described and classified based on their focus, assuming a geometric or symbolic designation, based on the dimension thus either 2D or 3D and also based on the display, static or dynamic [23].

Geometric techniques involve the graphical representations that are based on geometric forms such as lines, surfaces, volumes. Consequently, it includes techniques from exploratory statistics

for example, scatter plots but also from other domains (e.g. Parallel Coordinates visualization) [41, 23].

On the other hand, symbolic techniques are concerned in representing data using pixels, icons arrays, graphs, charts, hence emphasizing the graphical display of quantitative information and the relations within. Kosslyn, in his book *Understanding Graphs and Charts* [43], divided them into four main categories: (i) graphs; (ii) charts; (iii) maps; (iv) diagrams. In which they differ essentially in the way that information is presented and the interpretation that is expected from its analysis.

2.1.2.1 Graphs

Graphs are the most common visualizations, but also the most stifled, and they require at least two scales representing the variables that are being paired in order to describe a relation between both. Moreover, in order to represent higher values for the measurable variables, visual attributes such as area, length are increased proportionally to the difference among them.

2.1.2.2 Charts

Charts frequently act upon hierarchical data since they reflect relationships among entities. Graphically, charts have a structure in which the relationships are visible and entities are connected by them, the so called links. Links can provide some sort of indication through labels and have a direction associated with (e.g. directed or undirected).

2.1.2.3 Maps

Maps, as the name suggests, correspond to a graphical representation prescribing a portion of territory and "the internal relations among parts of a map are determined by the spatial relations of what is pictured" [43]. Furthermore, to identify distinct regions not only the borders are well defined but also, usually, they are colored differently.

2.1.2.4 Diagrams

Diagrams are another category of symbolic techniques, making use of conventional symbols to afford a better understanding. Diagrams are represented by a schema of entities or sub-components of entities and commonly, based on the display, they are classified as static by their lack of interactability. "Diagrams can lead to great insight, but also the lack of it" [11], the schema and the symbols used must have in consideration the targeted audience because it greatly depends on the illustration of the problem itself and the readability of it.

2.1.3 Data Visualization Techniques

As mentioned and described before, the visualization techniques were improved along the years, some epochs with greater contributions and others with few, some widely accepted as a recognition

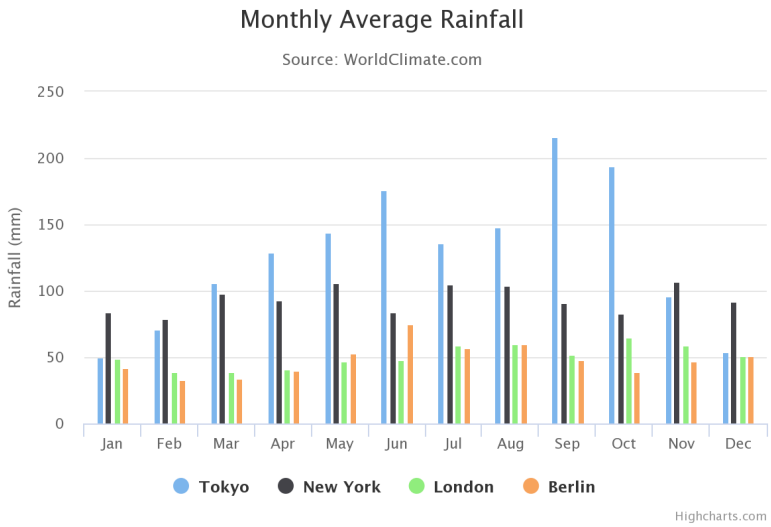


Figure 2.1: Column graph

of a well designed representation, some discarded [32, 13]. Recently, with the arrival of computer systems and their enormous power, those techniques were largely transposed from hand drawn to the digital field, incorporating a vast number of advantages: the capability to handle huge data sets [34], the graphical interactivity which allows a customized display for the analyst thus increasing the overall performance of the exploratory task [10, 1] and the possibility of being dynamic, reflecting the changes on data in real time [41].

These days, the number of adopted data visualization techniques for visual data mining is really tremendous. From standard 2D/3D techniques such as bar and line charts, illustrated in Figure 2.1 and Figure 2.2, to stacked displays (as a necessity from dealing with data highly related and multi-dimensional) [41]. The utility of each graphical representation highly relies on the characteristics of the data to be represented, for instance, in order to represent the variation of a variable a period of time it is more viable the use of a time-series, nevertheless, if the aim is to represent categorical data and their aggregation into categories, a bar chart should be considered.

Undoubtedly, the list of data visualization techniques is vast and because it was not possible to fully or even partially cover them, this section served as an overview of potential symbolic and geometric representations [43] that can be adapted to this work. The important concepts to acknowledge is that the graphical visualization should be always adequate to the data type and in any case forced. In addition, the same data can be represented in distinct ways, the essential is at the end keeping it simple and obvious. At last, there is also an opportunity of mixing multiple displays, for instance, a line graph with a bar chart. Whenever this happens we call it combined graphs.

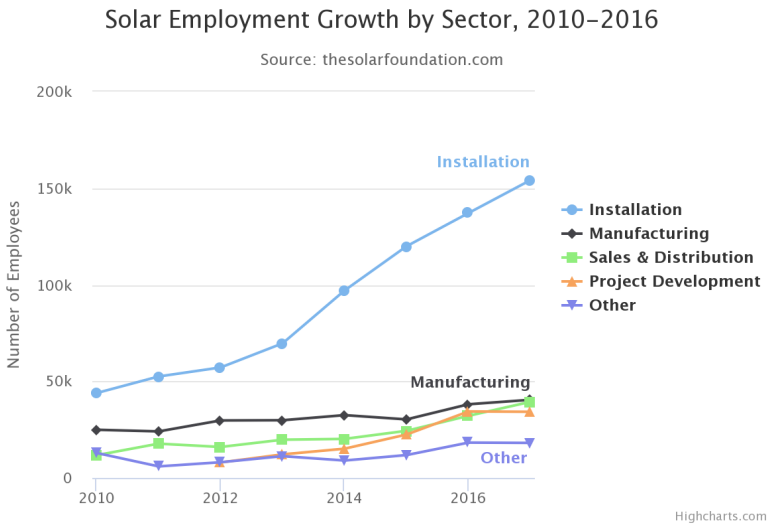


Figure 2.2: Time-series graph

2.2 Web Platform as the Presentation Layer

It is remarkable the transition of Information Visualization from hand made to computational systems. It brings the full potential and speed when in comparison to traditional methods, allowing the user to handle much larger data sets in a faster time pace, which would not be possible otherwise or would take a substantial amount of human resources. This interaction between humans and computers raised a necessity of another field of study — the Human-Computer Interaction, where the "key notion is that the user and computer engage in a communicative dialogue whose purpose is the accomplishment of some task." [12]. Thus, the system developer not only has to focus on the logical component but also the design and how perceptible and clear should its representation be, as it only reflects the mapping rules made by him.

The primary advantages of adapting the enumerated concepts of InfoVis into a web platform can be described as follow:

1. By building software upon supported web-standard technologies, it provides the user with the possibility of visualizing information everywhere using only a browser and an Internet connection, independently of the device (e.g. Computer, Mobile) and the operating system used (Windows, Linux, Mac, etc.) [50].

2. There are numerous cost-free tools for graphical displays that can be integrated with web platforms. Most of them implement widely accepted visualizations, for example, bar chart, scatter plot, maps and allow custom attribute adjustment such as color and also some sort of interactivity.

3. The associated cost of implementing a web platform is usually less on behalf of the no priory plugin or software acquisition requirement. Furthermore, the user does not need to spend time installing them either.

All of these advantages made the web the "perfect" platform for monitoring and visualizing data. Most of the visualization tools are now web-oriented contributing to a more broad acceptance of the web as a monitoring platform by enterprises and the general public. And, the combination of the web with visualization tools that are involved in a monitoring process leads to the concept — dashboard. A dashboard can be seen as a visual display of essential information with an established purpose through one or multiple visual representations consolidated so it can be monitored at a glance[27, 28]. These days there is an enormous number of existing dashboards, which can be applied in the most variety of contexts. Although, in a broad context, we found as a good reference Sisense[1], Klipfolio[2], MixPanel[3]. While in a clinical context (a context that our project belongs to), there are different applications for a dashboard, such as Smart-trial[4], Ethical[5], Talos[6]. Having a few focusing on biological statistics and others focusing in ePRO (Electronic patient-reported outcome) where the patient is asked to answer, commonly, a questionnaire with topics ranging from user engagement to quality of life, such as RAYLYTIC[7], Kaiku Health[8] and datacubed[9].

Such platforms follow good practices when it comes to design and content organization, thus, and as we will see later, some properties and design choices had them as inspiration.

2.3 Web-Oriented Visualization Tools

The topic InfoVis is getting more and more involved as individuals and enterprises find an opportunity in its adoption, a vast amount of frameworks were developed during the past years to fulfil this ever-growing need. The number of innovative graphical representations are also expanding largely due to the open-source frameworks.

According to the available visualization tools, we considered that there are two main distinctions, being:

Libraries — there is a considerable volume of charting libraries nowadays, which are focused on offering tools for representing data through predefined visualizations. Important to notice that, these libraries must be integrated with other tools or frameworks as they do not operate standalone. Example of such libraries are Highcharts.js[10], FusionCharts[11], D3.js[12], where the main differences reside in the fact that they comport different customization levels, pricing, number of graphical representations offered, etc.

[1] https://www.sisense.com/
[2] https://www.klipfolio.com/
[3] https://mixpanel.com/
[4] https://www.smart-trial.co
[5] https://www.ethicalclinical.com/
[6] https://www.biorasi.com/clinical-trial-optimization-software
[7] https://www.raylytic.com/en/
[8] https://kaikuhealth.com
[9] https://www.datacubed.com/
[10] https://www.highcharts.com/
[11] https://www.fusioncharts.com/
[12] https://d3js.org/

Platforms — platforms already provide the graphical visualizations integrated within the system, leaving for the user both the operation of choosing the adequate visualizations through the UI based on a drag and drop functionality and import the data to be later represented. It is important to notice that there is not any connection to a data source, demanding the extraction of data to a file format, for example, Excel and then the importation of it. Example of such platforms are Tableau[13], Data Wrapper[14] and Rawgraphs[15].

Shortly, web is one of the most widely accepted and popular platforms when it comes to monitoring, it provides scalability, accessibility since it is on the internet, every device from everywhere can have access to it, furnishes the facility of use, no required installation, no dependencies, so it can run everywhere.

2.4 Configurable Web Interfaces

Building a web platform and its user interfaces is a costly and lengthy one: whenever a new platform is needed, the procedure is repeated all over again and whether the developers are experienced or not, is a challenge to build it from scratch. The intended solution aims to define a web platform for representing data with a common interface among multiple projects without the need for coding or to re-arrange all the user interfaces.

There is an approach well-known for building applications out of a predefined configuration. Such a solution is intrinsically related to a metadata approach, as stated by Dimas Gilang Saputra and Fazat Nur Azizah in their paper "A Metadata Approach for Building Web Application User Interface" [59].

Metadata can be defined as "structured data about data" [22] that "describes, explains, locates, or otherwise makes it easier to retrieve, use or manage an information resource" [51, 59]. The process of creating metadata can either receive subjective or objective input; with objective being attributes that can be determined objectively (e.g. data corresponding to the definition of a set of properties) and subjective, attributes that are as the name implies subjective based on different perspectives [22].

Metadata can have multiple encoding schemes:

HTML (Hypertext Markup Language) — makes use of the HTML advantages as it has a hierarchical structure thus the data can be described and related in the form of a tree and has the virtue of simplicity [59].

XML (Extensible Markup Language) — highly used regarding its principles of modularity and extensibility. It is one most used markup languages for exchanging structured data between systems [22]

[13] https://www.tableau.com/
[14] https://www.datawrapper.de/
[15] http://app.rawgraphs.io/

JSON (JavaScript Object Notation) — it was created to allow data exchanging between heterogeneous platforms, with the great advantage of establishing a well-defined data structure and being simple to understand by humans and easy to parse by machines. It expects a key-pair: an attribute name and a value; or a collection of JavaScript objects.

YAML (YAML Ain't Markup Language) — as it is a data serialization language, the data has an unambiguous structure that is easily interpreted by humans.

Essentially, a metadata-driven approach can follow one or both of two paths, a first focused on the configuration of the user interfaces and the second focused on the configuration of the data access (e.g. database structure). Throughout this section, these will be explained in detail.

2.4.1 Metadata-driven UI

In this approach, "metadata is used to create a customized user interface (UI) portion of an application" [18]. A metadata model is then defined and responsible for storing information about the user interface. Moreover, it can be changed dynamically without further code implementation by the developers, whereas the main objective is to prevent the problems aforementioned. Following this approach, instead of developing the user interfaces programmatically, the developer defines the metadata model in which each web platform will always require its own metadata definition. In case modifications to the interfaces are expected, the metadata is the only thing to be updated and not the whole code structure [59].

The typical process of metadata-driven UI starts by defining a metadata file representing the interface components, their relations and structure. Afterwards, this file must be stored to be later accessed and once this procedure is finished, it will be interpreted in order to define the user interface accordingly. The intermediate node named by "Bind Data Sources" is only required in the presence of a scenario where there is a necessity of binding data to UI controls, otherwise, it should not be taken into consideration [18].

Nowadays, there are products that follow this approach. In which, based on a directory that holds a group of configuration files following one of the encoding schemes stated above, it is possible to build the interfaces of a system. Usually, this approach fits the needs of a CMS (Content Management System) where the system only deals with static files, however, monitoring field trials always involves a set of interfaces holding data that is the result of a process of recurrently storing data, so the application of these products methodologies is not enough and in this context, their functionalities must be extended for our needs.

Kirby[16] is a good example of this since, it is a tool that allows the creation of a CMS based on a predefined configuration, emphasized on the adaptability. In addition, it supports the definition of configuration related to the database connections and queries, thus, similar to the functionality we aim for.

[16] https://getkirby.com/

Another good example would be NetlifyCMS[17], in which there is a predefined file system structure that the user must follow. Since the configuration is, most likely, to be different from others CMS, it is possible to define a YAML file containing information related to the authentication and collections, being collections elements that define the structure for the different content types. The main difference relatively to Kirby is that it does not support database connections and the possibility to retrieve data from them, essentially because, it has as an assumption that the system is completely static.

2.4.2 Metadata-driven Databases

For most of the systems, following a metadata-driven UI approach is not enough since in order to the interfaces reflect dynamic data from a database, the system also needs to understand how it can have access to that same data. Furthermore, "Using these tools frequently requires developer intervention and knowledge of the data model, which works against the self-service orientation of the original web application interface" [7]. A viable solution would be a metadata-driven database, demanding metadata specification at multiple levels [49]:

Conceptual At this level, the specification should contain an explicit representation of data semantics. Semantic models raised to capture more meaning of the data and its structural organization and also for design to become more systematic [49, 35, 15].

Logical Another level apart from conceptual is the logical level. At this level, it must have a formal definition in order to be possible its implementation while maintaining a considerable degree of independence from the physical one.

Physical The latter level, consists of providing a mapping of the data belonging to the logical level. Moreover, it describes the details of how the data is stored physically through a file structure, the physical schema, etc.

Succinctly, it provides a precise description of each one of its components (tables), a full discretization of its attributes (type, name, etc) and in case its a relational database, the relationships among elements. Consequently, by having an implicit knowledge of the subject data the end-user can manage, through an interface, the data being presented. At any occasion, changes to the displays presented can be applied by changing the metadata file(s) expressing the database information, without changing the presentation module [20]. Example of such mechanism could be the search interface that "leverages dynamic HTML, JavaScript, and dynamic SQL" with the intention of providing a custom query without any further coding besides changing the respective metadata files [7].

[17] https://www.netlifycms.org/

2.5 Related Work

Along the years, the importance of web-based remote monitoring and data visualization is increasing due to the continuous progress in technology and the ever-growing data generated. Nowadays, the concepts of remote monitoring and web technologies are quite often used together and broadly applied in multiple fields such as business, health systems [14, 65], psychiatry [30], road traffic [26], therefore the existence of a large work in this area is imminent.

The possibility to infer a realistic diagnosis proves considerably crucial in fields such as healthcare systems, where patients require an unceasing tracking of their signals, for instance, vital signs and body temperature [65].

Through this section, a list of relevant related work is identified and succinctly described. To notice that most of the monitoring platforms are conceived to address a specific problem and not to be adapted to several different projects.

2.5.1 Visualizations for Mental Health Topic Models

This work had its origins in a master thesis and consisted in developing a system capable of converting data from texts, into information through visualizations based on a web platform, in order to facilitate counselors work and allowing them to spend more time with the individuals being submitted to this analysis. Only four different charts were used according to "four levels of granularity" [14], built with the JavaScript library, D3.js.

There are undoubtedly differences to our project, starting with this being an ad-hoc solution, with a smaller scope that does not require a larger number of graphical displays and where customization is intended. By this means, D3.js was an acceptable tool.

2.5.2 Minos: A Generic Tool for Sensor Data Acquisition and Storage

Minos is a Java tool for collecting and storing data relative to sensor's readings in wireless networks. The main explicit objective is to convert data collected into a general data model or schema "archived" at a central repository, endowing researchers with a tool for handling data in heterogeneous environments. As part of a larger project, *Dhestino*, the idea is to also visualize the data collected and stored by Minos through a unique web platform interface, based on the REST paradigm for the implementation of web services [58].

In this context, the data is initially collected and stored in resemblance to a determined generalized model and only then it is visualized. In contrast to our project, where data is already stored, demanding a different approach — one that could abstract the presentation layer.

2.5.3 Metadata-driven Delphi Rating on the Internet

In this work, a web platform was designed and developed following a metadata-driven approach, as mentioned in Section 2.4, to collect and analyze opinions using a Delphi process [31]. Due to

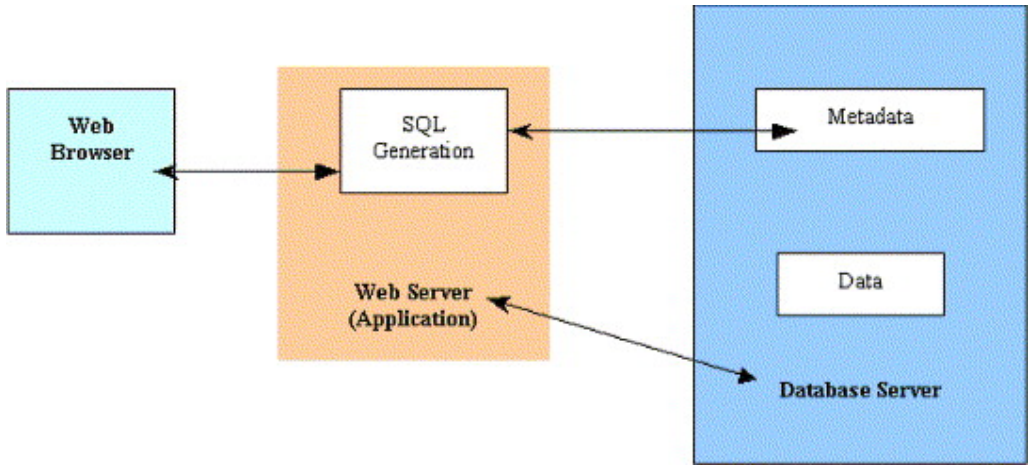


Figure 2.3: System Architecture from *Metadata-driven Delphi rating on the Internet*

the fact that they needed generic and reusable software without further coding tasks included, they decided to opt for this approach, ensuring it could later be adapted to different domains.

The system architecture follows the standard 3-tier approach [40]: presentation, logic and data. The end-user interacts with the web platform use pre-defined interfaces. Consequently, the client-side will communicate with a web server, which is responsible for mediating completely the communication between the web platform and the databases [19]. The architecture mentioned can be found in Figure 2.3, extracted from their document.

Despite being designed for surveys and questionnaires, a possible variation from this work can be extended to our project by the incorporation of data visualization techniques and a generalization of the domain being managed. At least, the system architecture and the metadata-driven approach can be inherited to confer the adaptability and ease of use expected.

2.5.4 A Metadata-Driven Framework for Generating Field Data Entry Interfaces in Ecology

As the name suggests, this project consists in the conception of a framework that serves upon structured data — the metadata — to generate automatically user interfaces. The metadata has a well-defined structure, an encoding schema based on XML-based that follows an EML (Ecological Metadata Language syntax). The objective is to describe completely arbitrary data schemes, enriching the whole process of creating those interfaces.

In order to support their idea, they employ a framework called Jalama that operates based on four steps: (i) Creation of an EML/XML document describing the data; (ii) Processing of the previous document by the UI generator module; (iii) Synchronization of the interface outputted on the client-side; (iv) Synchronization of the EML documents and the data collected in the field.

It is easily perceived that this is a generic solution but intrinsically related to the ecology field. Also, since it is fairly old (2005), so the existent tools were far more limited than the ones

available in the current days when it comes to visualizing data. Although, the core and fundamental principles are there, moreover, our solution can derive and have some guidelines according to it.

2.6 Summary

From the reading of this chapter, it is described Information Visualization as a really old topic, which started with the first diagrams and geographical maps in order to exchange information between individuals and suffer a major evolution through time with the arrival of more complex graphical representations until reaching the ones that are seen today. The idea of leveraging cognition through visual perceptions to obtain knowledge fits the current state of analysing data. Mainly because nowadays, with the ever-growing data stored, visuals are still better for exploratory tasks when compared to the automatic data mining tools. Furthermore, when conjugated with computer systems it is possible to bring the advantages commonly associated: the computational power and speed, the capability of leading with large volumes of data, etc. As a consequence, the information visualization was then adapted to a computational context, specifically to the web, which directly influenced the rise of multiple charting libraries and frameworks embedding data visualization techniques.

The process of monitoring field trials (independently of context inserted) now involves the recurrent use of web technologies as a manner of way for visualizing the data and tracking its subjects.

Throughout this section was also presented some of the research realized towards the adaptation of a single interface to a vast number of different domains, dynamically, by being able to represent their data. Such work relies on metadata-driven approaches either being based on database or user interfaces, where a metadata file is created defining and describing the structure of data stored in the databases and the organization of the UI, followed by an interpretation module in order to be reflected in the presentation layer without any coding task besides the modification of the respective metadata.

Chapter 3

Problem Statement and Solution Proposal

The primary objective of this chapter is to define and describe, the problem at hand and the solution developed to overcome it. Hence, the technological tools chosen for the solution as well as an overview of application architecture emphasizing its components and interactions will be considered.

3.1 Problem

Commonly, web and mobile application "prototypes" are being submitted to field trials and tested by a restrict number of previously selected participants, mainly, to understand strengths and weaknesses through received feedback to possibly redesign the product before releasing it to the public. The typical collection and storage of data from different sources such as sensors, self-report or behavioural, is already being applied, yet, there is not a standard approach nor a proper way to visualize that same data. Additionally, there is the need for a generic platform in which its interface can reflect the target project data without further coding process involved, greatly decreasing the costs of re-defining and modifying both back-end and front-end code whenever a new project is carried out.

The problem can then be deconstructed into two sub-problems: (i) building a platform, responsible for presenting the graphical visualizations of the data stored. In this scenario, the web is the targeted platform. It makes sense in the way it has the accessibility without requiring further installation, the computational power and speed at the client side, the ability to handle larger data sets and most important there is a vast number of charting libraries and front-end frameworks which work seamlessly; (ii) the ability to be extended to subsequent projects while supporting their distinct inner characteristics – data and project specifications (e.g. database type).

3.2 Hypothesis

We believe that an application following a 3-tier architecture, in which the presentation tier represented as a web platform and, the logical tier, acknowledging the data structure and UI interface through a predefined YAML configuration, then researchers are able to conduct the monitoring procedure while decreasing the costs of re-designing the platform.

3.3 User Stories

During the planning phase, we proceed to clearly understand and identify the requirements of the application as well as the identification of the common databases, data types, project scopes and the features expected by the end-users of the application since it was of extreme importance . In order to do so, two different semi-structured interviews were conducted (attached in Appendix A) according to each of one of the use case actors of our application: the first aiming to the end-users of the platform, the researchers; the last having in mind the developers that normally manage the data being stored. It was then submitted to a total of six participants, evenly distributed among these two distinct groups.

Based on the collected information, we were able to identify that: (i) MySQL and Firebase as the two main databases; (ii) field trials always involve participants and data is the reflection of their behaviour; (iii) the most commonly used charts were bar, line and pie charts and also time series; (iv) the collected data comes in multiple forms from sensors, questionnaires to applications. Consequently, the frequency in which they collect information is also different;

All of this information immensely aided us in the process of conceiving the application, as we had information about prior projects, suggestions and the requirements, we were able to have a more concrete outline of the architecture. Also, the insights from the interviews aforementioned provided by the participants lead to the user stories enumerated in Table 3.1 and Table 3.2.

Table 3.1: User Stories — Researchers

Number	Name	Description	Priority
US01	View participants	As a Researcher, I want to list the participants of the current field trial so I can easily check their information.	High
US02	Select participant	As a Researcher, I want to select a participant so I can follow his progress.	High.
US03	Filter participant	As a Researcher, I want to visualize the data related to a participant through graphical representations so I can extract information visually from it.	High

Continues on next page...

Table 3.1 – continued from previous page

Number	Name	Description	Priority
US04	Visualize aggregated data	As a Researcher, I want to visualize aggregated data so I can better understand the overall performance of the participants.	High
US05	Filter temporal data	As a Researcher, I want to be able to filter participants information according to a time range so I can better understand their behaviour during a certain time lapse.	High
US06	Compare multiple participants	As a Researcher, I want to be able to compare data between multiple participants so I can better understand their different progress during the trial.	Low
US07	Alert system	As a Researcher, I want to be notified whenever a participant reaches a specific variable threshold so I can minimize the expended time analysing the participants	Low
US08	Export Visualization	As a Researcher, I want to be able to export a specific visualization in a predefined format so I can store and share it with someone else.	Low

Table 3.2: User Stories — Developers

Number	Name	Description	Priority
US09	Configure pages	As a Developer, I want to be able to define a page and its components so I can have access to it in the website.	High
US10	Define several chart types	As a Developer, I want to be able to represent several types of charts so I can better understand the data collected in several different ways.	High
US11	Define retrieved table attributes and filters applied via YAML	As a Developer, I want to be able to define the data attributes to be retrieved while filtering based on a specific attribute threshold.	High

Continues on next page...

Table 3.2 – continued from previous page

Number	Name	Description	Priority
US11	Define retrieved table attributes and filters applied via YAML	As a Developer, I want to be able to define the data attributes to be retrieved while filtering based on a specific attribute threshold in order to access restricted data.	Medium
US12	Configure chart	As a Developer, I want to be able to configure charts title and labels so researchers can understand the data being represented.	High
US13	Define aggregation cards	As a Developer, I want to be able to define a card that will further reflect a specified table attribute aggregated based, also, on a specified operator (max, min, etc) in order to easily identify some indicators.	High
US14	Configure layout	As a Developer, I want to be able to configure a page layout so I can visually organize data.	Low
US15	Support SQL databases	As a Developer, I want to be able to connect and access to data stored in SQL databases so I can retrieve data following SQL paradigms.	High
US16	Support Cloudfirestore	As a Developer, I want to be able to connect and access to data stored in Cloudfirestore databases so I can retrieve the data stored in them.	High
US17	Multiple databases connection	As a Developer, I want to establish simultaneous connections to multiple databases so I can retrieve data distributed among them.	Medium
US18	YAML validation	As a Developer, I want to validate the YAML configuration files so I can better understand syntax errors that were made.	Low
US19	Define Cohorts	As a Developer, I want to be able to define cohorts so I can select participants based on their cohort.	High
US20	Authentication System	As a Developer, I want to have an authentication system between the presentation and logic tier so I can guarantee increased security of the whole system.	Low

Unfortunately, not every single requirement planned could be implemented, mostly, due to the time and effort required to overcome them. The requirements have priorities ordered, from the

highest to the lowest priority feature.

3.4 Solution Proposal

Resultant from an in-depth analysis of the problem stated in Section 3.1, it is perceived that there are different necessities arising from this project, being the visualization of data through a web application and the adaptation to distinct and heterogeneous contexts. The approach adopted comes as a proof of concept to a wider problem, therefore, we focused on achieving a certain functionality and then, with the right validation, prove it worked by being able to be extended to a bigger scope (e.g. supporting more visualizations, types of databases, etc.), applicable to field trial targeted for monitoring.

To come up with the solution, we considered wise to use a bottom-up approach, starting by understanding the data patterns transversal to all projects or the templates that can be induced, providing a standing point for the solution for the second sub-problem and afterwards, building the rest on top of that.

3.4.1 Application Architecture

The application architecture adapted to solve this problem is composed of three components that are interconnected and work seamlessly, as illustrated in Figure 3.1: (1) a web interface running on the client side (e.g. researchers devices); (2) a mediator providing an external API to the web platform to retrieve the data models with the main purpose of solving the principle of heterogeneity among projects; (3) the source of information — the corresponding databases from the diverse projects.

This software architecture approach is well known as the 3-Tier Architecture [40, 48], consisting of an architecture separation in 3 distinct tiers:

- **Presentation tier:** as the names suggest, this tier is in control of the presentation and the user interface, remaining concentrated in displaying information and handling user related events;
- **Application tier:** It contains the business logic and rules of the application;
- **Data tier:** represents the database access, management and information storage. Database access comprises the operations of accessing and retrieving data, while information storage relates to writing operations of the data to be later available. Lastly, management involves the management of the data stored. It is important to note that, this tier, can be composed by a unique or multiple databases hosted in one or more data servers.

Having as reference Channu Kambalyal and its article “3-Tier Architecture” [40], the 3-tier architecture comes with some disadvantages, for instance, it has a more complex structure, it is more difficult to set up and maintain and the extra tier adds vulnerability to the application (since it is also single-point-of-failure). However, the advantages outnumber the disadvantages, example of

them are: (i) complex application rules easy to implement an application server; (ii) business logic off-loaded from database server and client, which improves performance; (iii) changes to business logic automatically enforced by server – changes require only new application server software to be installed; (iv) Application server logic is portable to other database server platforms by virtue of the application software.

Relatively to the physical layout, the three different tiers, in our solution, were designed for a the one described in Section 3.4.1.3, although there are some other possible variations distributed along with the internet. Therefore it is crucial and indispensable the communication among those components. The communication flow always involves the mediator as the centre of all communication and the first-receptor of all messages exchanged. The expected flow is:

$$\text{client-side} \longleftrightarrow \text{mediator} \longleftrightarrow \text{database}$$

Of course, in the presence of a scenario where both frontend and backend are running on the same machine, the frontend could have direct access to the configuration files through the file system, having the advantage of reduced delay when compared to a remote service. However, following an approach based on web services, they could be physically separated and also, imagining a future and mature product where the mediator is a centralized component, storing all the configurations, this methodology would cover this case.

Our approach comes with the implementation of web services, through a well-defined protocol that ensures the preservation of integrity, confidentiality and reliability over the Internet. Following on that, the implementation of these web services will be based on a vastly accepted protocol, REST (Representational state transfer), due to the fact that is simple, fast and it offers CRUD (Create, Read, Update and Delete) operations by HTTP (HyperText Transfer Protocol) methods, in which messages should contain all the information needed to the server handle it since no state is stored [56]. In addition, it allows multiple encoding types besides JSON.

To better understand the different components, their interactions and how they are physically distributed, a set of diagrams will be presented next.

3.4.1.1 Component Diagram

The component diagram, illustrated in Figure 3.1, shows the organization and relationships among components in our application.

As mentioned before, there are three main components: the client, mediator and database provider. The client explicitly uses an API provided by the mediator to obtain the configuration, in which, then it accesses data through an API offered by the database provider. This provider, a built-in library that encapsulates a database, allows data access, modification and differs depending on its type. Therewith, for Cloudfirestore[1] and Realtime[2] it is Firebase[3] while for SQL it is Sequelize[4].

[1] https://firebase.google.com/docs/firestore
[2] https://firebase.google.com/docs/database
[3] https://firebase.google.com/
[4] http://docs.sequelizejs.com/

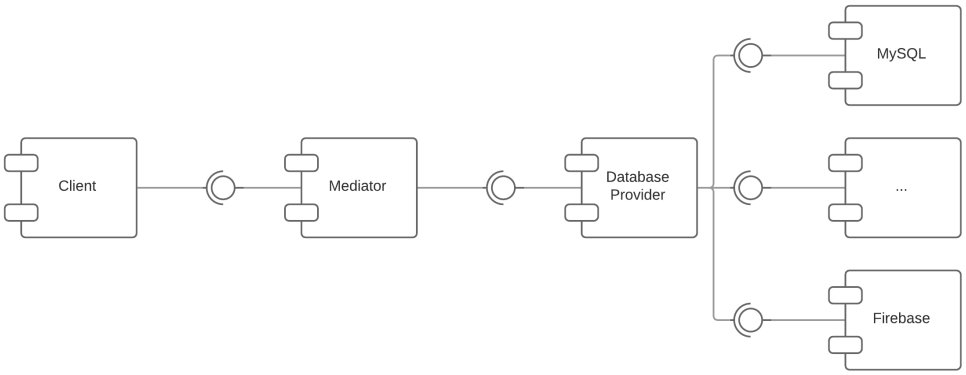


Figure 3.1: Component diagram

3.4.1.2 Process View

The process view, illustrated in Figure 3.2, is a comprehensive way of representing dynamic aspects, explaining the application processes and how they communicate while focusing on run-time behaviour.

The common flow involves the web starting by sending a request to the mediator, followed by a read operation of the configuration files written in YAML and only then it is able to request data from the database. Afterwards, it sends the response back to the web.

3.4.1.3 Physical View

The physical view, illustrated in Figure 3.3, "describes the mapping of the software onto the hardware and reflects its distributed aspect" [44].

This application was specially designed to have the layout above, yet there can be some variations, either front-end running on a different physical server than the mediator or the same server hosts both front-end, back-end and database. This last scenario can occur more often, regarding the network settings (in case it is internal with ports and availability restrictions to outside networks) or firewall accesses, it may need to be running on the same machine.

Even if these layouts differ in the number of machines used, they have something in common: the components. With that being said, it is always included the client device, which will be interacting through a website, the front-end, the mediator API and the database modules.

3.4.1.4 Package Diagram

The package diagram, illustrated in Figure 3.4, is intended to describe the package hierarchy and dependencies between them. The packages were divided into three main layers:

Config — Package containing every configuration file of the system.

Frontend — Package containing the presentation layer files.

Backend — Package containing the logical layers files.

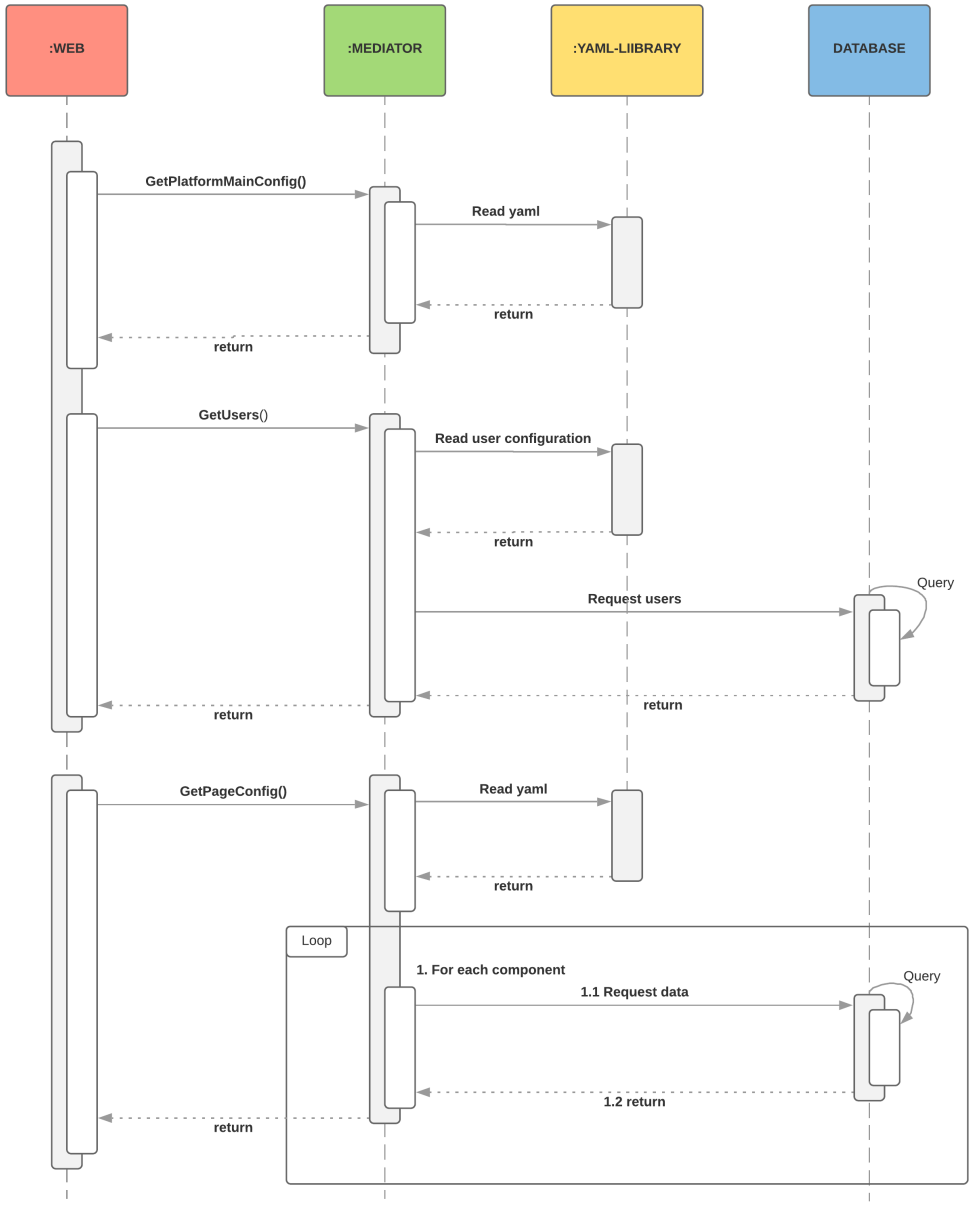


Figure 3.2: Process view

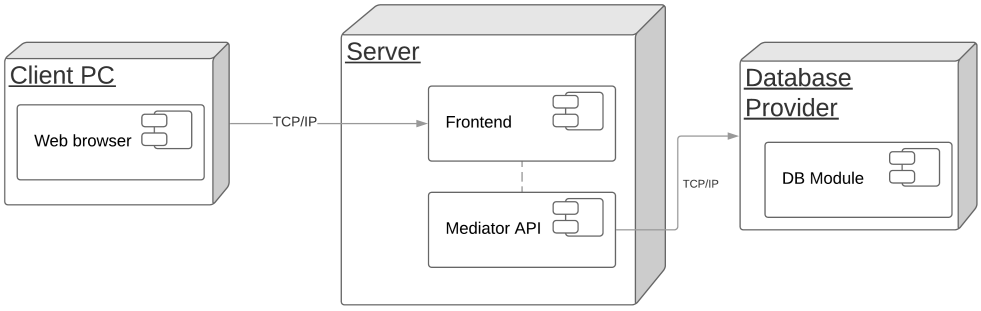
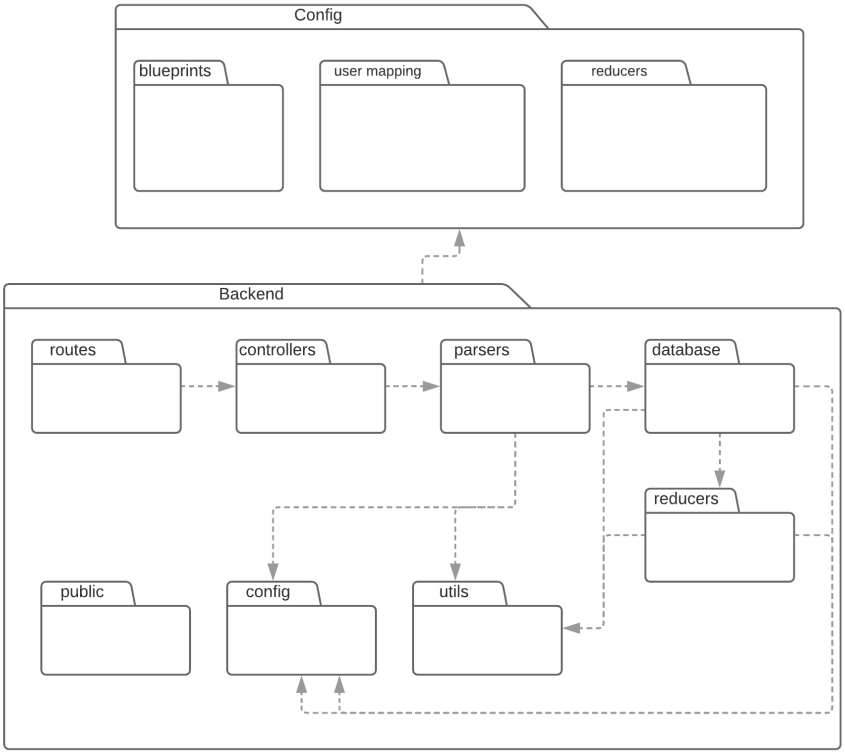
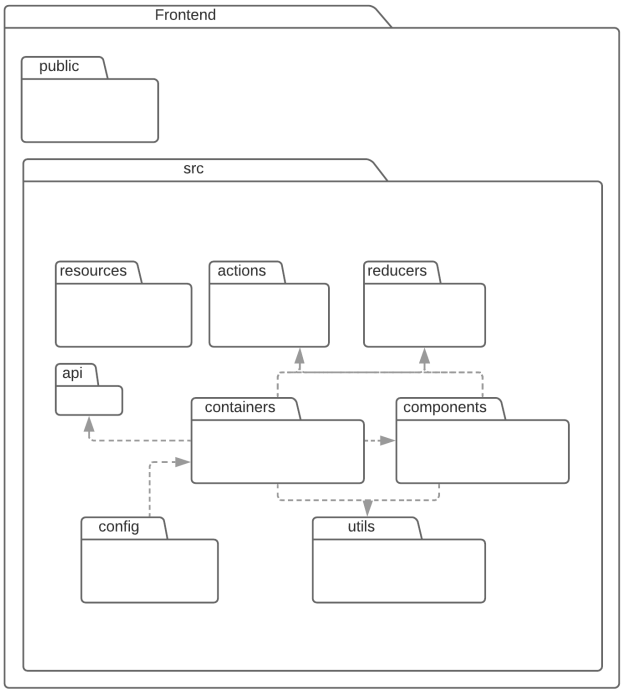


Figure 3.3: Physical view



(a) Backend and Config



(b) Frontend

Figure 3.4: Package Diagram

3.4.2 Tools and Technologies Adopted

3.4.2.1 Front-end

There is a vast number of JavaScript frameworks. However, in concordance with past experience of the team and also to facilitate the process of developing the user interfaces of the web platform, a front-end framework was used — React.js[5]. React.js is a JavaScript library to build user interfaces on the web. It comes with the concept of Virtual DOM (Document Object Model) in which, based on state modifications only the affected elements will be re-rendered on the screen, meaning the real DOM is also updated. Another great advantage of its use is the modularity and the division of UI partitions into individual components, which can also be included as part of other components [24, 33].

Furthermore, considering the multiple charting libraries available, a library such as D3.js[6] has the main advantage of being full-customized and allows the creation of visualizations completely from scratch, however, it does not support old browsers, on the other hand libraries just as High-Charts.js[7] provide strong default visualizations with the downside of its pricing label. Thus, to reach and cover the most browsers, bearing in mind that customization is neither a must nor an improvement for this project, and also the ease in use, the satisfactory number of visualizations available and the price (free is optimal), the tool used was Plotly.js[8], as it comprises all these advantages relatively to those libraries.

The combination of these two ensured that the presentation layer had all the needed tools and requirements to carry on towards their principal objective.

3.4.2.2 Mediator

For the mediator development, we ended up choosing Node.js[9] together with Express.js[10], not only considering the prior experience of the team but also a large volume of existent libraries available. This, together with the coding language used, Javascript, grants more consistency between the different components and allows code re-usability. Besides, it has also the benefit of having a great open-source supportive community.

3.5 Summary

In conclusion, the problem addressed throughout this chapter has a huge impact on the interaction and the information that can possibly be retrieved from the monitoring of field trials by the researchers. Prior to these projects, as stated in Section 3.1, there was not a proper way of visualizing such data which is continuously being collected and stored distributively among different

[5] https://reactjs.org
[6] https://d3js.org/
[7] https://www.highcharts.com/
[8] https://plot.ly/javascript/
[9] https://nodejs.org/en/
[10] https://expressjs.com/

database servers. The designed software architecture seems a solution to be greatly considered as it completely divides the different layers, granting an abstraction to the front-end: better modularity and most important, it fulfils the project primary requirements. Although, it also has its drawbacks such as ensuring the communication channels are secure and with an extra vulnerability added to the system as a whole by including a mediator.

Furthermore, each one of the components defined in the application architecture will be dissected and explained in detail.

Chapter 4

Mediator

In this chapter, the implementation details of the mediator component will be defined and described including the main services it offers as an API to the front-end, followed by the databases it supports, how it is achieved, the possible limitations and finally, the proposed data schema for future database implementations.

4.1 Description

The second component of this application naturally belongs to the application/logical tier regarding the fact it comprises the business logic and rules. The primary purpose of developing a mediator is to act, as the name suggests, as a mediator/broker between the web browser and the corresponding database, controlling how these interact with each other. Hence, instead of those two tiers establishing a direct connection between them, the communication flow will always pass through this component. The main advantage of its use is that this mediation "simplifies, abstracts, reduces, merges, and explains data" [63], also, a 2-tier architecture would imply more code in the client side leading to a less smooth user experience and in case the logical rules must be changed, the client software must be distributed again, where the only good side of its use would be a simpler structure and an increased facility in maintaining it [40].

When applied to a real-life context, there is the possibility of projects using different databases that can be physically separated or not. Nowadays, there is even a broad number of database types, however, in order to simplify, we chose a few that will be initially supported coming as a proof-of-concept, as described in Section 4.4. Relatively to the client-side, it demands data to be later displayed on the screen, although it does not have the fundamental knowledge about the data layer (e.g. the schema, endpoint connection, etc), for the most part, because it is built with a predefined skeleton that is later filled in resemblance to the configuration. Such configuration must be defined a priori by the researcher and must contain all the necessary information readable by the front-end, for instance, the number of pages, the name of the platform and the layout for

each one of the pages. For this reason, it fulfils the basic idea behind the use of a mediator — supply a level of abstraction to the presentation tier, leveraging most of the data process and the required prior knowledge of the database schema, data attributes and connection configurations, leaving only small and necessary operations to it.

This knowledge is granted to the mediator by following a metadata-driven approach based on information systems, as mentioned in Section 2.4.2. By creating an entry point metadata file containing all the pertinent and vital information describing the general configuration and database accesses without redundancy, using, for that purpose, a YAML encoding schema. As a justification for that choice, YAML among the others (e.g. JSON, XML) has the advantage of being simple and human-friendly.

Consequently, the front-end is now able to display the desired interfaces, considering its configuration was previously defined in the proper and expected file system location. The metadata files are expected to have information relative to the:

Database — such as type (e.g. MySQL, SQLite, PostgreSQL etc.), the connection properties: username, password, host, etc.

Database Schema — its name/identifier, the relationships with other tables, its attributes and for each attribute the variable type.

Important to note that we considered as an assumption, when conceiving the application, that the data is already being collected and stored in one or multiple databases, which means our application works only with read operations, there are not, in no case, modifications to the current state of the information.

4.2 API

Services are offered in the sense of a RESTFul API, allowing other devices to operate over them. Here, due to the fact that the main goal is to monitor field trials where data is already stored, being them weekly reports, passive data recording or special events triggered from third-parties devices (e.g. sensors, applications), the services provided are read-only. There are just three of them, some related to the users and some to the platform configuration.

The requests are then handled using routers and controllers. The routes are defined in Node.js and associated with the server App, in the same way, controllers are associated with each one of the routes of the router. Whenever a specified route is called it will delegate it to the controller, containing the correct information on how to handle the request.

Moreover, applicable for every service, they either return the expected information (it can be null in case there is none) with a status of 200 or a response with a status of 500 whenever an exception was thrown at the server side.

GET /api/users

Gets the participants identifiers from the corresponding database (as we will see later on, it is possible to retrieve a subset of attributes according to the configuration, the same applies to the number of participants retrieved as they may as well be filtered).

Table 4.1: GET /api/users

Input	Success	Failure
GET http://<domain>/api/users	[{ "key": "079a3ad9-2f96-4bd6-a4d8-719 b8ba75fb0", "id": "Participant 1", "onboarding": true, "mealplanTimestamp": 1559054872767, "npsTimestamp": 1559054872767 }, { "key": "08c3b991-09b4-44d4-953d- d7219ea82006", ... }, ...]	Status 500

GET /api/config

Gets the main configuration of the platform such as name, abbreviation and the pages that will later be presented in the left sidebar.

Table 4.2: GET /api/config

Input	**Success**	**Failure**

GET
http://<domain>/api/config
	{
	"title": "Lifana",
	"abbreviation": "LF",
	"pages": [
	{
	"fileName": "Activity",
	"name": "Walking Activity"
	},
	{
	"fileName": "MealPlanning",
	"name": "Meal planning"
	}
	...
	]
	}
		Status
		500

GET api/config/pages/<page>/users/<id>

Gets the configuration for the page provided, filtered or not by user, depending on the request parameters. The parameter “page” must be defined while the “id” is optional.

Table 4.3: GET api/config/pages/<page>/users/<id>

Input	Success	Failure
GET api/config/pages/Activity/users/user0001	[{ "type": "time series", "plurality": "single", "title": "Activity registry", "ylabel": "Number of steps", "xlabel": "Grouped by Day", "specifications": { ... "data": { "x": ["2019-01-18T00:00:00+00:00", "2019-01-27T00:00:00+00:00", "2019-02-01T00:00:00+00:00", "2019-02-12T00:00:00+00:00", "2019-02-14T00:00:00+00:00"], "y": [156, 679, 2446, 2906, 6325] } } }, { "type": "Histogram", "title": "Activities Histogram", ... },]	Status 500

GET api/resetCache

Service offered by the back-end to allow the reset of the complete cache state.

Table 4.4: GET api/resetCache

Input	Success	Failure
GET http://<domain>/api/resetCache	 1 { 2 success: "Cache Reseted" 3 }	Status 500

4.3 Mediator Workflow

The mediator itself has a well-defined working flow, involving all the functionalities and sub-components mentioned above. Starting through the web services exposed over the Internet, whenever a request is made, the router will delegate it to the corresponding controller, which is responsible for handling all the process thereafter. The controller for every request will first check its cache, if it has stored before and that same data still "lives", sends a response with it otherwise it follows the next approach: (1) Calls the `.getData()` function from the database module API; (2) The data return is then submitted to the reducer based on the `.submitToReducer()` function, a function that converts the raw data to the expected format by the UI component; (3) The resultant formatted data is sent back as a response.

4.4 Database Support

As part of the data tier, the databases are an essential and indispensable component of this application, holding all the information relative to the field trials being the object of the monitoring process. These databases can be heterogeneous or homogeneous, according to their type. Although, as we mentioned earlier, this project appears as a proof-of-concept, therefore, only the most relevant database types for the company were featured, namely: Cloudfirestore and SQL databases. Firebase Realtime Database, a Google database, is also supported but at a minimum level when compared to the others.

The main functionalities relative to the databases are the attribute selection, the inner join aggregation and the filtering operation. Their implementation depends on the database type, some using built-in methods already provided through an interface while others achieve the same functionality using JavaScript methods or libraries.

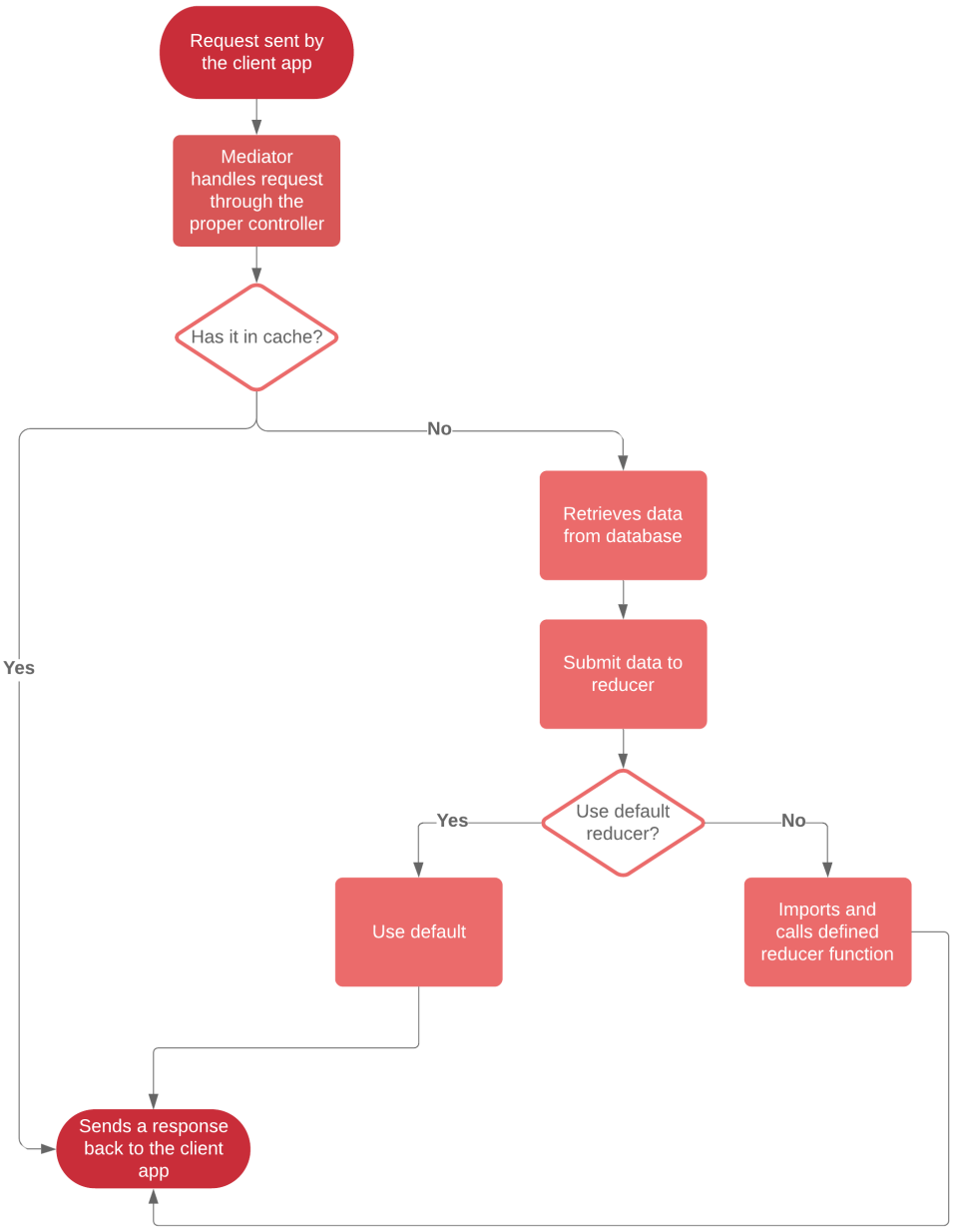


Figure 4.1: Mediator processing flow

4.4.1 Assumptions

Having in mind that every field trial always involves users, a table representing those users must always exist and other tables or collections must reference it in some way, depending on whether it is a Firebase or a SQL database.

4.4.2 Attribute Selection

The attribute selection is quite relevant since, usually, a table contains more information than the one we actually need for representing in a chart. Sequelize already offers a way to filter attributes, if there is just one table (no inner join required), the attributes provided will be directly filtered; otherwise, based on a recursive inner join function, if the current table to be joined has any of the attributes specified (the model allows reflection, therefore, acknowledge its own attributes and keys), those who match will be filtered. On the other hand, Firebase databases do not extend this functionality so they were replicated via the JavaScript library lodash[1], more precisely, with recourse to the function `.pick()`.

Assuming a case scenario where two tables will be joined together and they both have an attribute with the same name and it is supposed to filter that same attribute, there are two possibilities: (i) the user just specifies the attribute and as a consequence the attribute from the first table will be considered, ignoring the second; (ii) the user specifies through YAML < TN >.< attr >, where TN is the table name and attr is the target attribute and as a result, the attribute with that table name will be filtered.

```
1 var queryObject = { ... };
2 var attributes = await retrieveTableAttr(model, properties);
3 queryObject.attributes = (attributes.length) ? attributes : [];
```

Listing 4.5: Sequelize attribute selection

```
4 data.reduce((arr, el) => {
5   arr.push(pick(el, properties));
6   return arr;
7 }, []);
```

Listing 4.6: Cloudfirestore attribute selection

4.4.3 Inner Join Aggregation

Having into consideration the typical SQL, tables refer to one another, using foreign keys. Moreover, in case the complete information is to be retrieved, then a join must be done based on the

[1] https://lodash.com/

foreign and primary key from both the one referring and being referenced, respectively [6]. There are multiple types of joins, although, at this stage, we support only one — the inner join. As data is to be submitted to specific charts, comprehending predefined target axis, their values should not be null. Thus, inner joins are the commonly expected joins to be executed as they return records with matching values in both tables. For instance, considering a case scenario where we want to visualize a line chart and one of the tables contains the attribute corresponding to one of the axes (e.g. x-axis) and the second containing an attribute to be represented in the remaining axis (e.g. y-axis), considering one of them already has a reference to the other one, an inner join will give as the proper data while other joins would not.

Regarding Cloudfirestore and our proposed schema, the data is considered denormalized by either replicating or referring other tables data and being completely flat, in other words, there are no sub-collections derived from each one of those tables. Furthermore, if data is being replicated, each table has all the information needed and no additional process is required. However, if the approach followed is based on references then, and knowing that Cloudfirestore as a NoSQL database does not support inner joins, this has to be replicated by the server based on: (i) multiple queries (one for each table); (ii) inner join through JavaScript code; This procedure is illustrated in Figure 4.2.

4.4.4 Filter Selection

Another functionality of our application is the filtering selection of the data. It is possible to specify an array of filters which will be applied in the form of a concatenation of filters through an AND operator. The typical structure of an operator is an object containing a: (i) target — the targeted table; (ii) operator — the filter operator, that must assume one the following values: “!==”, “==”, “>=”, “>”, “<=”, “<”; (iii) value — the value to be compared with.

This functionality is achieved in Sequelize through the `where` property when defining the object to be submitted to the function `.find()`, while in Cloudfirestore it is achieved based on the `.where()`, a function provided by its API. To note that when using Cloudfirestore to do such filtering it requires the definition of indexes, which can be done through the Firebase project console.

These filters can be utilised in two different moments during the configuration definition. Firstly, when defining the users’ location, it is possible to filter them as well as select the attributes wanted. Lastly, in each component, their data can be filtered according to the filters specified.

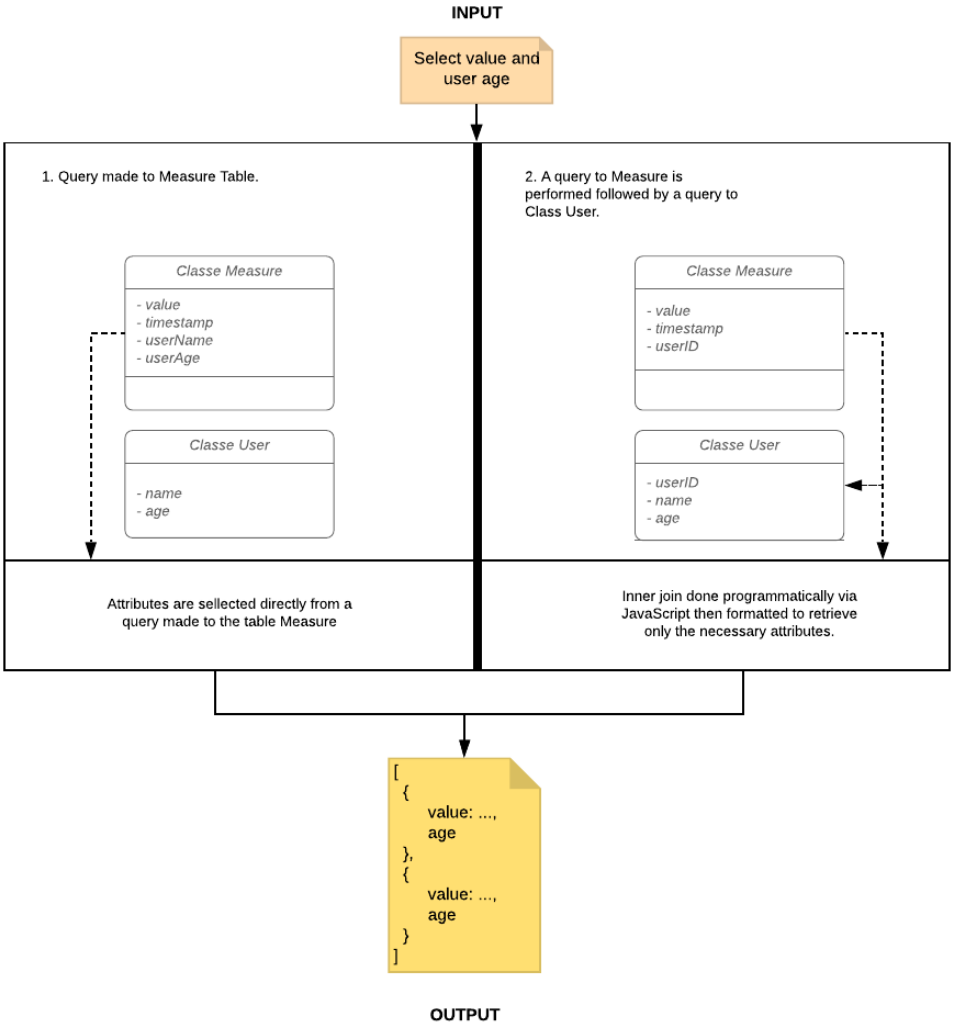


Figure 4.2: Example of a Cloudfirestore inner join

```
Post.findAll({
  where: {
    authorId: 12,
    status: 'active'
  }
});
```

Listing 4.7: Sequelize filtering Example

```
db.collection("Posts")
    .where("authorId",
            "==",
            "active");
```

Listing 4.8: Cloudfirestore filtering Example

4.4.5 SQL

The standard SQL databases are still quite often used, particularly MySQL, which made their support so important. Nonetheless, supporting only MySQL would lead to major changes in the future, so bearing in mind the SQL well-known structure definition we searched for an abstraction layer that could encapsulate several relational databases. That is when we came across with the concept of ORM (Object Relational Mapping), an ORM allows the manipulation of data from a database through an OOP (Object-Oriented-Programming) paradigm by mapping objects with underlying databases and providing an API for querying or modifying those objects [39, 57]. Although, it also appeared with one drawback, as it comes with the notion of models, where models are, in simple words, mappings between objects and tables, meaning they must be defined a priori. To counter this problem, the proposed solution provides a higher level definition of these models, instead of the usual coding process, they can be defined through YAML without having to deal with implied information. Also, because there are not any writing operations, it is neither necessary the model synchronization nor the mapping of the whole database but rather the tables and attributes targeted by the monitoring process, for instance, extra tables inside a wider database.

Because the mediator was built with Node.js, it was inevitable an ORM composed for JavaScript. Hence, Sequelize was chosen due to being, most likely, the most used and well-known JavaScript ORM and also because it supported the necessary features.

4.4.6 Cloudfirestore

Cloudfirestore is the second database type supported, mainly because, it is considered an improved version of Firebase Realtime database, highly recommended by the Firebase team. There is a great number of advantages, among them we have [29]: (i) it has a new and more intuitive data

model with richer and faster queries; (ii) it scales better; (iii) the scaling is automatic, no need for sharding; (iv) includes chain filtering and sorting functionality; (v) has the possibility to query a document instead of an entire collection. There were a lot more advantages arising from its adoption, although, considering the features required, some of them did not seem worth to mention. Those who were mentioned backed up our decision towards its support.

Regarding the fact is follows a NoSQL paradigm, it has a dynamic schema for unstructured data when compared to SQL that has a standard structure. Therefore, it is possible to have a collection within collections recursively, which may reach an immense depth. In case we face a similar scenario, which is most likely to occur, whenever a collection is queried, only the attributes from its documents will be retrieved and not its sub-collections (precisely what was needed). There are four approaches to overcome this problem, namely:

- Instead of using the concept of sub-collections, a combination of map objects and arrays could be used. The main disadvantage associated with it, is that a document has a max limit of 1 MB (when using a sub-collection, its size is not accounted for).
- Use of `.collectionGroup` function provided by Cloudfirestore to obtain a sub-collection data nested inside another collection.
- A flatter structure, one level max, through data denormalization, thus avoiding sub-collections. This solution has two variations: either the data is replicated in each node, and each node is self-contained (when in the context of the data it stores) or it has a reference to another table in similarity to SQL structure.
- The last possibility involves querying the wrapper collection followed by a query for each child document to obtain its sub-collections. Such an approach will induce a bigger overhead in comparison to the ones described before. In addition, since the Cloudfirestore pricing is made according to both write and read operations, it would have some significant implications.

In order to have consistency among future projects, a convention for a schema was established, based on the denormalized approach. The only drawback is that denormalization usually implies replication of data, moreover, there is the necessity of keeping its integrity which will cause an increasing amount of operations in order to keep data similar in multiple places.

4.4.7 Firebase Realtime Database

Firebase Realtime Database has minimum support, but mostly because it was one of the databases used in a prior project so we needed it for the platform validation. The minimum level of support is justified because Cloudfirestore is increasingly getting more users and as aforementioned, it is considered an improved version of it. Minimum level support is understood as the possibility of query and apply reducers without the possibility of doing multiple queries or filter data by some operator and attribute.

4.4.7.1 Assumptions

Having in mind it is a NoSQL database and that every field trial always involves users, a relation representing those users must always exist. Either it is a loner collection and the other tables documents ids refer to a user (equal to the respective user document id) or it has those tables as children which automatically links both.

4.4.7.2 Limitations

As a minimum support level, it means that there are features missing when comparing to the others databases supported, namely: (i) the UI component that aggregates a value by an operator; (ii) the filtering selection, as a result of how limited is Realtime relatively to filtering, meaning it would have to be done via Javascript on the server side; (iii) the aggregation — since it is aimed for a NoSQL paradigm, the aggregation should not be required as it is expected that linked tables are nested in one another.

4.4.7.3 Implementation

In order to retrieve data, and acknowledging that same data is structured as a huge JSON in its database, whether a property is a list of objects, an object or a primitive value, a path to the target table must be provided. As an explanation, considering a table User with a nested HeartRegistry table, the query should be similar to `db.ref("User/" + userID + "/HeartRegistry")` as it is not possible to retrieve it without providing the "userID" since User is a collection of documents.

The typical process comprises the table name definition for each UI component, in which, based on the database configuration defined a priori, the table is then found recursively and its path obtained by a mix of backtracking and concatenation. This is mainly achieved due to the fact that JSON, in this context, is interpreted as a tree where a table is viewed as a node/leaf, thus, an in-depth search is applied through recursion. In each step of backtracking, a verification is made regarding its node type, in case it is a collection then the path is a sort of "ref = "Table/" + id + resultantPath;" otherwise it is 'ref = "Table/" + resultantPath;".

All of this information is illustrated in Figure 4.3, where the input table is "Bp" and the numbers inside brackets indicate the order that steps are performed. Resultant from the analysis of it and in conformity to what has been said before, the ordering is justifiable by the in-depth search approach while the path creation is achieved by the backtracking as a consequence of the recursion. As we can see, the path created in step 5 does not include a measureID because Measure is not a collection but rather an object.

4.4.8 Multiple Database Connections

When conducting the initial interviews, one of the needed requirements (US17 in Table 3.2) was the possibility of connecting to multiple and/or distinct databases. For instance, one of the projects

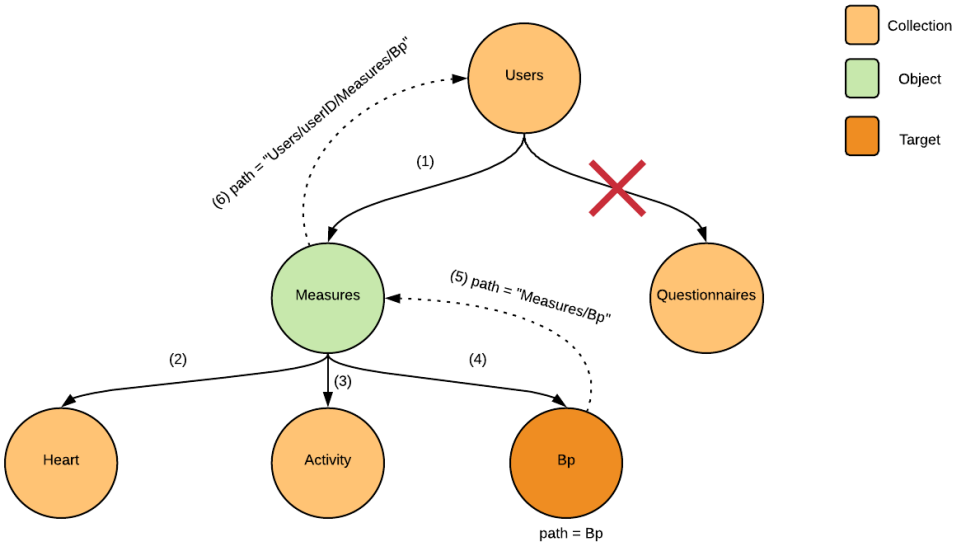


Figure 4.3: Example of a path construction process

at Fraunhofer and target of a case study in Chapter 6, had two databases storing different information, linked to one another based on a stored attribute that references the same user as the other one.

Here, to fulfil this functionality, the developer with insight about the databases at stake, must define the configuration of each one of them. In addition, each one must have an identifier so later, when defining the UI components, it is possible to reference which database we intend to query and obtain the corresponding information. Nevertheless, since now there can co-exist multiple databases it is important to decide from which the users will be retrieved. It is equally important to guarantee the link between both databases, as it is possible to users to be mapped differently, for example, in one it can be the attribute “user_id” inside “Users” table while on the other can be “userID” from “User” table.

There is no restriction when it comes to the type of the database, so it is conceivable to have multiple SQL databases as well as multiple Firebase databases.

4.5 Cohorts

In some cases, users can be grouped into cohorts without their explicit declaration in the database. This functionality aims to help researchers selecting users that belong to the same group based on some characteristic (e.g. location), by selecting a cohort, the researcher can only select users from that same cohort so he does not need to search between a full list of users.

4.6 Reducers

Once the data is retrieved, it still needs to be formatted based on the chart type read from the page configuration file. In order to do so, the data processed by the database module is then sent to the reducer module. These reducers functions follow a convention, although, in case the user wants to have control over this formatting operator or do additional filtering operations, he is able to define his own model (that must be placed inside the folder "config/reducers") and afterwards, define its name in the component configuration so it later overrides the default reducer function. The reducer always receives as first argument the data retrieved by the database, and as second an array of arguments defined via YAML when setting the reducer, as shown in the code listing 4.6, provided below:

```
18 exports.flattenSingleTimeChart = (data, args) =>  {
19     var x = args[0];
20     var y = args[1];
21
22     x = getPropertyName(x, data[0]).shift();
23     y = getPropertyName(y, data[0]).shift();
24     data = orderBy(data, [x], ['asc']);
25     // x-> first key & y -> second key
26     return { x: data.map(el => extractDateFromTimestamp(el[x])), y: data.map(el =>
           parseFloat(el[y])) };
27 }
```

Listing 4.9: Example of a reducer

4.7 Cache

From the analysis of the field trials already applied, there is not the need of constantly reading the configuration and querying the database as users are set, usually, at the beginning (in some cases they start in distinct time stages) so they are not being updated every day, hour or even minutes. In addition, the devices that collect data differ in the frequency they register thus, for instance, if it is a questionnaire, it commonly appears once a week while a sensor can have a rate of one registry per minute.

By introducing a cache mechanism, consecutive requests have a significantly decreased response time, therefore improving the user experience perceived by the researchers while decreasing the time spent analysing the participants. The TTL (Time To Live) set for a request aiming to retrieve a page configuration response or the field trial participants must also differ, one day for users and five minutes to the page configuration.

Nevertheless, as we mentioned, if data is updated it will not be reflected in the monitoring platform until the cache content times out and is deleted. In order to diminish this problem, the server offers a service to reset its own cache.

4.8 Application Configuration

The configuration can be divided into multiple pieces with different responsibilities, namely: (i) platform — configuration that provides which and how components are displayed in the web platform; (ii) database — configuration that defines how and with whom the database connections are established and also its structure; (iii) reducers — configuration that defines the set of reducers that can later be applied when defining the UI components; (iv) users mapping — configuration that allows mapping user identifiers retrieved from the database to a more user-friendly format.

4.8.1 File system

The typical structure, as represented in Figure 4.4 as a tree of components, in which, the leaves represent the files and the intermediate nodes the directories, contains the following main directories:

blueprints — this directory contains the information related to the platform interfaces through the files declared inside the sub-directory "pages" and also, the database structure which is defined in site.yaml.

reducers — the reducers directory can have multiple files containing multiple functions. Also, it has a default entry file named "index.js" which is responsible for exporting all functions from all files.

cohorts — this file contains the definition of the users grouped into cohorts.

user-mapping — At last, this directory contains a file that will operate on top of the retrieved users.

The required directories and files, at all times, are "blueprints", "site.yaml", "pages" sub-directory and its files. The rest of them are optional, although it is obvious that, for instance, when it is intended to use reducers, the corresponding directory, index and reducer file must be provided. The same applies to cohorts definition and user-mapping;

4.8.2 Database

To connect to the databases, our application requires the file "site.yaml" with the database configuration. This configuration must include information related to the connection establishment, its database type, its overall structure, etc. Going into more detail, the main properties available are: (i) usersLocation — This property is required whenever there are multiple database connections; (ii) databases — the list of the databases configuration; On the other hand, the properties associated with each one of them are described in Table 4.5.

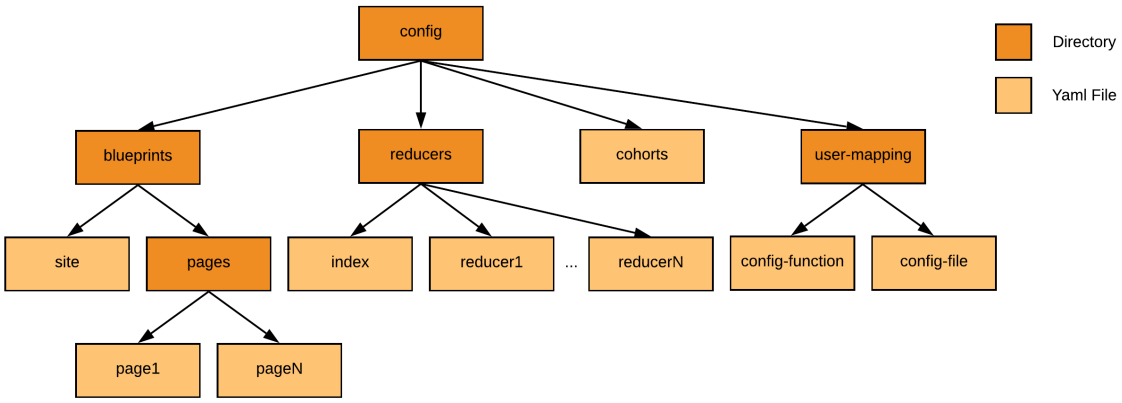


Figure 4.4: Configuration file structure

Table 4.5: Database configuration

Property	**Values**	**Notes**
Identification and Classification		
id	Can assume any string besides empty.	No default. Required in case there are multiple databases.
type	<firebase> | <mysql> | <postgresql> | <mariadb> | <sqlite> | <mssql>	No default. Always required.
subtype	<realtime> | <cloudfirestore>	Default is realtime. Only required if it is a Firebase database.
Access configuration		
uri	A string containing a valid URI.	No default. Only required if config property is not set.
config	Object containing database, username, password and host property.	No default. Only required if uri property is not set.
Users Access and Mapping		
users	Object with table, idAttribute and nmAttributes property.	If property not provided assumes the location as being table “Users”, the idAttribute as being its usual id and no additional attributes to be retrieved besides the common id.
Structure		
structure	Structural object.	No default. Always required.

4.8.2.1 Identification and Classification

Whenever there are multiple database connections, there is a property “id” — the database identifier, which must be set, providing a way of unambiguously identify them. In order to classify the database and to choose the proper module to handle the data, the property “type” — the type of the database, must be defined. Also, if it is a Firebase database, the property “sub-type” must also be defined.

4.8.2.2 Access configuration

The database connection establishment depends on whether it is a SQL or a Firebase database. For Firebase, it comes with the definition of a property “config” with the sub-properties: apiKey, authDomain, databaseURL, projectId, storageBucket, messageSenderId, which are obtained via the Firebase console. When it comes to SQL, the connection can be done through the property “‘uri”, which is the database connection URI, or through the property “config”, in similarity to Firebase, although with some nuances relatively to the properties it receives, where the accepted ones are: database — the name of the target database and host — the database host connection point; The Listing 4.10 appears as an example.

```
28 config:
29   database: smSQL
30   username: postgres
31   password: 123
32   host: 127.0.0.1
33 # OR
34 uri: mysql://username:password@host:port/dbname
```

Listing 4.10: SQL connection configuration

4.8.2.3 Users Access and Mapping

Since there is the need of accessing a list containing all users, the corresponding table or collection, depending on the database type, must be provided (the default table name is “Users”). If the developer wants to obtain additional information stored inside this table, can do it by providing a list of elements or an element to the property “nmAttributes”.

In terms of linking multiple databases, the user identifier must be the same. To do so, it is possible to define the attribute through the property “idAttribute” which will serve as the user identifier when querying that same database.

In the sense of filtering the users, it is possible to configure a list of filters to be applied based on the property “filters”. Each filter is composed by a property “target”, “operator” and “value”. The first corresponding to the target attribute, the second to the operator which can take the following values: “!==”, “==”, “>=”, “>”, “<=”, “<” . And finally, the third consonant to the value to be compared with.

```
users:
    table: User
    idAttribute: externalID
    nmAttributes: gender
    filters:
      - target: age
        operator: '>='
        value: 23
```

Listing 4.11: Users configuration

The Listing 4.11 serves as a confirmation to what has been said previously.

4.8.2.4 Structure

The database structure configuration is greatly influenced by the type and sub-type of each database. Furthermore, there are three distinct variations, one for Firebase Realtime Database, one for Cloudfirestore and another for SQL databases.

SQL The structure configuration can be seen as a list of tables. For each table, we must define a set of properties, that have no default values and are always required, that will later be mapped to Sequelize models, namely: (i) "PK" — represents the primary key and accepts a string; (ii) "attributes" — as the names infer, accepts a list of strings (attributes — columns in SQL); (iii) relations — represents the relations with other tables, accepts a list of relations.

A relation object can have up to four properties: (i) "type" — the relation type which can assume three values, "belongsTo", "belongsToMany" and "hasMany"; (ii) "target" — the table it is related with; (iii) "through" — in case its a *N-M* relation, through assumes the value of the intermediate table that relates both of them; (iv) "FK" — a relation of type "belongsTo" or "belongsToMany" must have defined the name of the foreign key. With respect to SQL, the relation object has also three variations according to the relation type. To note that whenever two tables have a relation both will have a relation object in their relations list object. If it is a 1-1 relation we choose one table that will reference the other by having its type defined to "belongsTo" and the remaining with "'hasMany". In case its a 1-*M* relation, just need to add a relation "belongsTo" to the one referencing and "hasMany" to the one being referenced. At last, in a scenario where both tables are related to each other based on an *N-M* relation, naturally, there will be an intermediate table resultant from this relation. The intermediate table will not require additional information besides its attributes, while the others two must reference each other based on the "through" property. Examples of both *N-M* and 1-*M* relation are illustrated in Listing 4.12. and Listing 4.13, respectively.

Cloudfirestore The structure adopted for Cloudfirestore is a denormalized and flat one as aforementioned. In certain circumstances, we can have data replicated, or we can have data

referencing one another. This structure definition is not needed if it follows a completely denormalized approach with each table being independent and self-contained as it has all the essential information, although based on the principle of replication of data. However, if a collection is related to another then such information should be specified.

Usually, each table comports two properties, "attributes" and "FK". The first represented as a list of attributes (strings) and the second as an object composed by "name", the foreign key name, and "reference", the related table name. The join is then made according to each document key. The Listing 4.14 illustrates what was mentioned before.

Firebase Realtime Database Taking into account the typical NoSQL nested structure, the database structure configuration is similar to describing it as a JSON. Imagining the JSON structure as a tree graph, each intermediate node has three properties: "plurality", "document" and "children". The first is presumed to be either "collection" in case it has a list of elements as its children or "unique" in case the children are just properties (in which those can have nested properties or not). Relatively to the "document" property, it refers to the name of the node. And at length, "children" refers to the node children, that can assume as value a list of nodes, a list of leaves. To note that leaves do not have properties, their name is used explicitly. Also, the starting node must be called "root". Both listings, Listing 4.15 and Listing 4.16, exemplify the explained configuration.

```
43 structure:
44  - product:
45      PK: product_id
46      attributes: [ name ]
47      relations:
48        - type: belongsToMany
49          target: users
50          through: productUser
51          FK: product_id
52  - productUser:
53      attributes: [ product_id, user_id ]
54  - users:
55      PK: user_id
56      attributes: age
57      relations:
58        - type: belongsToMany
59          target: product
60          through: productUser
61          FK: user_id
```

Listing 4.12: Example of *N-M* relation configuration definition

```
62 structure:
63  - activities:
64      PK: activity_id
65      attributes: value
66      relations:
67        - type: belongsTo
68          target: users
69          FK: user_id
70  - users:
71      PK: user_id
72      attributes: age
73      relations:
74        - type: hasMany
75          target: activities
```

Listing 4.13: Example of a 1-*M* relation configuration definition

```
structure:
  User:
    attributes: [ name, teste ]
  Activities:
    attributes: [ timestamp, value ]
    FK:
      name: user
      reference: User
```

Listing 4.14: Example of Cloudfirestore structure

```
{
    Users:
        "x":
            Measures:
                Activities:
                    1553212800000: 2955
                        ...
}
```

Listing 4.15: Example of Firebase Realtime Database content

```
document: root
children:
- document: Users
  plurality: collection
  children:
    - document: Measures
      plurality: unique
      children:
        - document:
            Activities
          plurality:
              collection
                ...
```

Listing 4.16: Example of the corresponding configuration

4.8.3 Reducers

The custom reducers, that are to be defined, must be placed inside the directory “config/reducers” and can be distributed along with multiple files, inside this same directory, or mixed up in one single file. Differently, the entry file “ìndex.js” must always be defined (because it exports those reducers) and its content must be similar to the one illustrated in Listing 4.17. Bear in mind that, each reducer function receives as an argument the data retrieved by the server when it makes a request to the database.

```
103 require('fs').readdirSync(__dirname).
        forEach(file => {
104   let name = file.split('.')[0]
105   exports[name] = require('./' + name
          );
106 });
```

Listing 4.17: Reducer entry point content

```
107  - type: time series
108          ...
109    specifications:
110          ...
111      map:
112          file: mappers
113          method:
                 _histogramMultiExample
114          ...
```

Listing 4.18: Reducer usage through the UI component

Once these reducers are defined, the application is able to read and apply them. In order to do so, whenever the user is defining a UI component, under the "specifications'" property, the file name of the reducer and the name of the reducer function must be provided. As an example of it, the Listing 4.18 comes as an illustration.

4.8.4 User Mapping

Sometimes the common database identifier of each user is illegible to the researcher, for example, if it is a hash string with a ten character length. Thus, it might be necessary the definition of a map between the real identifier and a pseudo-random identifier (the one that the researcher knows it for). Moreover, this is supported in the case where we do not want to have personal information in the database, that could easily identify the field trial participant (e.g. name, address, etc), or even have this information spread across multiples databases.

In this scenario, the researcher responsible for conducting the field trial has a way of clearly identify the participant. Furthermore, this mapping between the database id and the identifier that the researcher implicitly knows it for can be done in two different ways: (i) through a file named "config-file.yaml", as represented in Listing 4.19, where the left side is the database id and on the right, the desired identifier; (ii) through a function designated by "config-function.js", as shown in Listing 4.20, receiving as a single parameter the data retrieved relatively to the users; Ultimately, in case none of those was specified, the default would be just using the database identifier itself.

```
# <database_id> : <desired_id>
user0001: Pedro Lima
user0002: Ana Viana
user0003: Miguel Silva
```

Listing 4.19: User mapping by file content

```
exports.default = (data) => {
    # Format data as desired
    # return the resultant data
}
```

Listing 4.20: User mapping by function

```
# site.yaml
title: Lifana
     ...
userMapping: function
     ...
```

Listing 4.21: User mapping application

Afterwards, the researcher is able to define it in the configuration in order to apply it. To this end, inside the file "site.yaml", the property "userMapping" can take, naturally, in resemblance to what has mentioned above, three different values: file, function and id. By way of example, Listing 4.21 is presented.

4.8.5 Cohorts

In similarity to user mapping, the developer can define a file named "cohorts.yaml" inside the "config" directory, where there is established a mapping between users identifiers and the corresponding cohort, just as represented in Listing 4.22.

```
USA:
    - user0001
        ...
PT:
        ...
    - user004
```

Listing 4.22: Cohort definition

4.9 Summary

The application architecture component described in this chapter lead to the interoperability between the remaining heterogeneous components, since it is the mediator of all communication between the client and the databases through offered services in the form of a RESTful API. To reduce some of the overhead and improve the user experience, a cache system was implemented with different TTL (Time-to-Live) depending on if its user or page configuration data. Regarding the data tier and its interactions, the databases supported were chosen, mainly, according to past projects. While the functionalities explored were based on necessity.

In its essence, the mediator is responsible for leveraging and abstracting the presentation tier while simplifying the whole system, requiring for that purpose, the configuration declaration, also documented throughout this chapter.

Chapter 5

Dashboard

The objective of this chapter is to present the process that led to the design and implementation decisions, as well as the main functionalities developed, followed by evaluation through the application of a usability test and the analysis of the results obtained.

5.1 Conception

Regarding the modular nature of our intended platform, we had to identify common components that are presented for data visualization. To do so, we proceed with the analysis of the state of the art and we came up with the identification of a few things, namely: (i) almost everyone had a sidebar on the left containing multiple links to distinct pages; (ii) an authentication and notification system was always featured; (iii) the pages represented an overview of the data; (iv) the layout, most of the time, followed a grid system; (v) some of them used a header panel to separate different sections in the same page; (vi) the UI components were limited to charts and also cards with a single value (displaying the aggregation of an attribute). In the sense of inspiration, not only the interfaces were designed according to the state of the art but also they comprehend the user stories since they were filtered according to it.

Moreover, since there is a filtering process going on, both temporal and user related, they had to be placed in an adequate position where they were grouped together. Also, these filters are expected to be applied to an entire page.

The process of designing the interface is usually accepted "as a process that is intrinsically open (new considerations may appear in time), iterative (several cycles are needed to reach an acceptable result), and incomplete (not all required considerations are available at design time)" [16, 17]. The same applied here, where the process of conceiving these interfaces was not done at first try, instead it was an iterative process. For that purpose, medium fidelity prototypes were constructed using Adobe XD[1], as attached in Appendix E. Once the first prototype was finished, it

[1] https://www.adobe.com/pt/products/xd.html

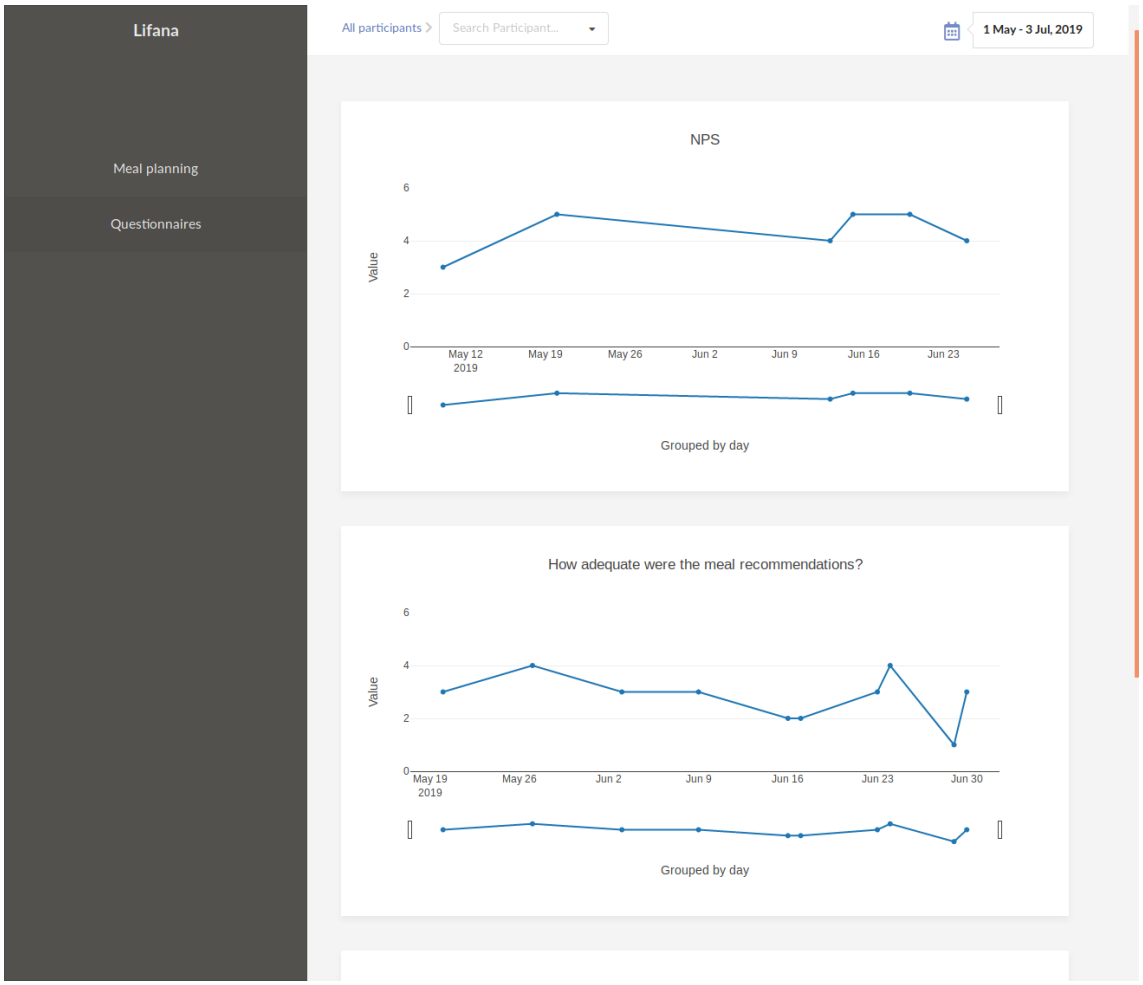


Figure 5.1: Example of a platform page

was then analyzed and some design problems identified, which lead to the next iteration. In every following iteration, the previous problems were corrected and the UI validated again. When we felt confident with the design, it was then transposed to a real prototype, meaning it was implemented via code.

5.2 Skeleton page

The designed skeleton page — the set of components that are common to every page of our platform — is composed, as illustrated in Figure 5.1, by: (1) a sidebar containing the principal navigation of our platform; (2) the list of links that redirect to each one of the page of our application; (3) the filtering bar which contains, as the name suggests, all the filters to be applied; (4) and at last, the area corresponding to the mutable page content.

The navigation between pages is essentially done through the left sidebar, where each button acts as a link and redirects to the corresponding page which will then call an endpoint that retrieves its configuration. The number of buttons presented is in conformity to the number of YAML files

inside the directory “config/blueprints/pages”. The breadcrumb aligned on the left and inside the filtering bar provides a second level of navigation, in which the page remains the same, while the request made and the data received as a response, changes accordingly.

5.3 UI Components

As aforementioned, the UI components selected derived, as a source of inspiration, from enterprises that work essentially with analytic tools. Thus, those components can be divided into cards, section panels and charts.

5.3.1 Card

Such component is mainly used to provide an overview, for either a participant or multiple participants, relatively to a specific table attribute, comprising a title, a value and an aggregation operator identifier. It admits the aggregation based on several operators being: “max”, “min”, “count”, “avg”, “sum”, with SQL databases supporting as much as Sequelize grants.

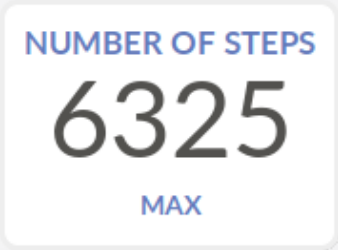


Figure 5.2: Card component

5.3.2 Section Panel

The section panel, as the names suggest, introduces a section in which its primary objective is to organize components inside a page since a page can contain multiple sections and each section contain multiple other components.

5.3.3 Charts

Initially, when we were still in a stage of exploring the visual representations, a total of 6 types of visualizations were used, namely: time-series, bar chart, pie/doughnut chart, histogram, parallel coordinates and scatter plot matrix. Although, when they were first tested by end-users some feedback was received and we realized that last two of them were slightly advanced and more difficult to interpret as these representations have strong statistical foundations. Naturally, the most used visualizations are expected to be the first four as they are the most simple, far reaching a higher number of researchers.

Time Series The typical process of monitoring a field trial has a certain limited duration, and participants are followed during this time, which in some cases, their activity is registered as temporal data (as it has a timestamp associated). The best way to represent such data is by using time-series where x-axis reflects the time and the y-axis the corresponding value registered. Illustrated in Figure 5.3, there is an example of the application of time-series visualization with real participants' data.

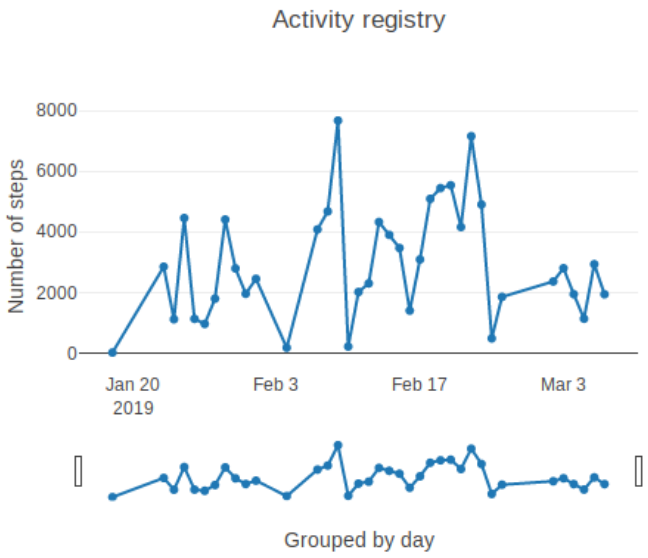
Bar chart Whenever there is categorical data, that same data can be comprehended into multiple and distinct categories (x-axis) and their respective values (y-axis) aggregated according to an operator such as mean, maximum, minimum, etc. The optimal representation would be either a bar chart or a column chart, where the orientation decision depends on the insight gained from the researcher configuring the application. Despite its common use for categorical data, this chart can also be applied for representing temporal data, emphasizing the area and allowing, in certain circumstances, a better interpretation when comparing to the typical time-series, as the area is directly proportional to the y-value. Although, time-series can also be represented with their areas coloured in similarity to an Area-chart. As a side note, the visual representation of this component through the platform includes an additional button that allows multiples types of aggregations such as prior mentioned.

Histogram The time to live of a field trial for a determined participant can be considerable extensive (duration of a few months), normally involving the execution of repeated procedures over time. With that being said, it could be of extreme value, the frequency analysis of those procedures turning the histogram a valuable representation as it is the best suitable for these occurrences.

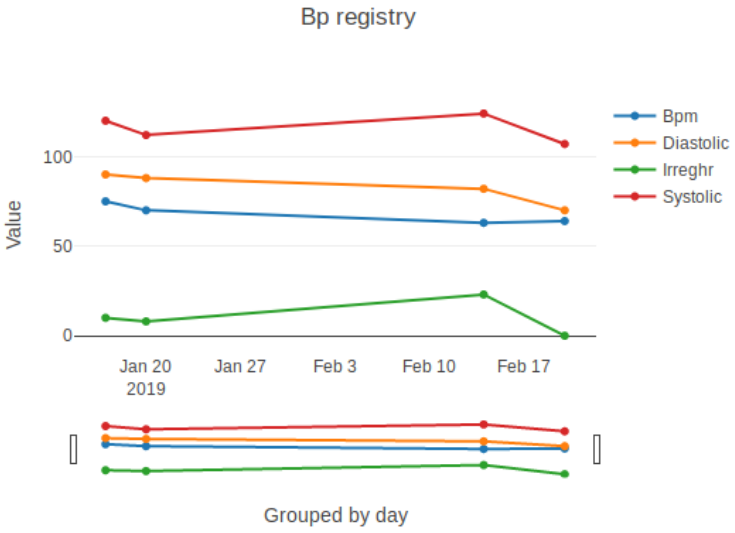
Pie chart Sometimes is also useful to compare categories in terms of the proportion of their frequency. In such cases, a pie/doughnut chart would be appropriate since the arc length is proportional to the frequency, evidencing, the proportions to one another.

5.4 Participants Listing

Throughout this document, we emphasized that field trials always involve users in order to keep in mind that the data collected has at some point a reference to them. These participants usually have personal information associated as well (e.g. age, profession, blood type, etc) that allows their identification, characterization and aggregation in groups. Furthermore, it is important to have access to this information through the platform instead of the default or defined id as researchers may not able to retrieve much information from it, so a page was designed especially for listing all the users that are participating in the corresponding field trial (or the ones actually filtered in case they were defined in the configuration). Participants are listed on a table and this table is complemented with a search and pagination. To note that this kind of filtering is achieved through a `.filter()` function on the front-end side.



(a) Single variable



(b) Multiple variables

Figure 5.3: Time series representations with participants' data

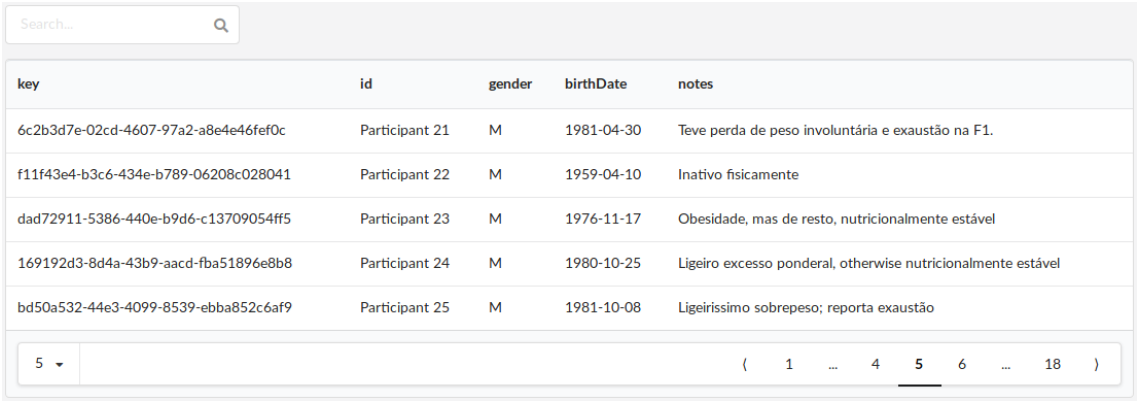


Figure 5.4: Example of how participants are listed

5.5 Filtering

There are two different filters in our platform, as shown in Figure 5.5, one that acts upon the user(s) selected and the other upon the time range.

The first allows changing between two views: a general overview of the data and a specific user view, both share the same configuration differing from one another only on the data that is displayed.

The second filter is responsible for selecting time ranges, reflected on temporal charts (time series or bar charts that deal with temporal data in their x-axis). Its UI component is represented as a date picker placed at the right upper corner that, whenever the user clicks inside it, a collapsed view becomes visible allowing the user to change between default time ranges (e.g. Today, Yesterday, This Week, Last Week, This Month, Last Month) or directly select a custom time range.

5.6 Interface Configuration

The interfaces configuration appear as an extent and complementation of the application configuration explained in Section 4.8, therefore, they are mainly configurable through the pages inside the directory “config/blueprints/pages”, with exception to the platform title which is set in the file “site.yaml” by the property “title”. Each page is composed of two different properties, “title” — a string representing the page identification in the platform sidebar and “components” — a list of components that reflect as the page content. Bear in mind, each component is restricted to the UI components defined in Section 5.3 and its classification into a type is done due to the property “type”.

(a) Participant filter

(b) Time filter

Figure 5.5: Different filters provided by the application

For components that contemplate data, they share a configuration property designated by "specifications". Since it highly depends on the UI component being represented, it takes the form of an object that contains, as a template, a property "database" — a string representing the id of the database (in case there are multiple databases), "tables" — a list of strings, representing the tables needed to retrieve the desired information, "filters" — representing a list of filters. Each filter composed by three properties, namely: (i) "target" — the attribute to be filtered (we need to make sure this attribute exists in the table defined or in one of the tables defined); (ii) "operator" — the filtering operator, the ones supported are: "!==", "==", ">=", ">", "<=", "<"; (iii) "value" — the value in which the target attribute will be compared with.

These components have, normally, the definition of one or more attributes that hold the data needed. Although, when configuring this attributes and considering there is an inner join between data and that both tables have attributes with the same name as the desired, the user is able to specify which one will be picked by converting the name of the attribute to the concatenation between the table that holds it with a dot and the attribute name (e.g. value -> Activity.value).

5.6.1 Card

As we have seen, a card contains a title, a value and an operator. The configuration mirrors these by offering a property "title", and a property "specifications". The second appears to be a variation of the object mentioned above, with the additional properties: "x" — a string serving as the name of the attribute and finally, "operator" — a string representing the operator to be applied which can take the values: avg, max, min, sum, count for Cloudfirestore and for Sequelize the same but complemented with other operators that the library offers. There is also a property named "round" that can assume the values of "u" (units), "d" (decimals), "c" (hundredths) or a number, being units — zero, decimals — one and so on so forth.

```
134 - type: Card
135   title: Number of Steps
136   round: 1
137   specifications:
138     database: smSQL
139     tables: activities
140     x: activities.value
141     operator: max
```

Listing 5.1: Card configuration

5.6.2 Charts

5.6.2.1 Shared

The graphical representations share some configuration such as: (i) "type" — the type of visualization, taking as values "time series", "Histogram", "barchart" and "piechart"; (ii) "title" — a

string representing the chart title; (iii) “specifications” — an object that defines the data access and retrieval.

On behalf of most of the graphical representations supported having two axes, apart from pie charts, they have properties that allow the denotation of a custom name to both axes, being those properties: “xlabel” and “ylabel”.

5.6.2.2 Time Series

A time series chart can have, in terms of plurality, a single or multiple values. Meaning that there can co-exist multiple lines in the same visualization. To do so, it is possible the declaration of a property “plurality”. When it comes to the specifications object, it is somewhat equivalent to the template defined previously, although with some modifications. It is expected to have “database”, “tables” in resemblance to it, but since it is a chart and contains two axes, both properties “x” and “y” are offered. Not only that, it also offers a property “groupby” which contains as sub-properties: (i) “time” — the time aggregation reference, having as possibilities: hour, day, week, month; (ii) “operator” — representing the aggregation operator to be applied, accepting avg, max, min, sum, count.

5.6.2.3 Pie Chart

Due to the fact that a pie chart is used in the context of analyzing the proportion of the occurrences of a determined attribute, the specifications object only requires the property “database”, “tables” and “x”. The last accepts an attribute which usually reflects an enumeration (e.g. foodType: vegetables or meat or fish).

There are some more specific properties, such as “subtype”, which can take only one value, “donut” and is used whenever we want to switch the representation to a donut chart. Also, it supports the definition of labels that allow the mapping between the attributes retrieved and a name, as shown in Listing 5.2.

```
142 labels:
143     meat: Meat
144     fish: Fish
145     veg: Vegetables
```

Listing 5.2: Labels configuration on a pie chart

5.6.2.4 Histogram

As we know, histogram groups numeric data into segmented columns. Be that as it may, it is possible the definition of an interval in the x-axis and the size of each segment. For instance, in Listing 5.3, the displayed values will be limited by 0 and 6000, having each segment a length of 500, so we have a total of 12 segments.

```
146 interval:
147     start: 0
148     end: 6000
149     size: 500
```

Listing 5.3: Interval range definition for Histogram

The specifications object contains the property “database”, “tables” and “x”. However, in case it is a Firebase Realtime database, the property “x” is not needed as it requires a reducer.

5.6.2.5 Bar Chart

The supported bar chart visualization can be applied in order to represent categorical or temporal data. The property “domain” indicates which one will be employed, accepting as strings: categorical or temporal. In case it is temporal the rest of the configuration is identical to a time-series. Otherwise, there are three possible configurations according to three different scenarios:

1. In order to act as a histogram, we must define a property “x” similar to a histogram with the additional property “y” with the default value “IS_COUNT”.

2. To express categories, and being in the presence of a case scenario where a single variable can have multiple categories, the property “x” will be the name of that variable and the property “y”, the value to be represented in the y-axis.

3. Also when expressing categories, imagining there are multiple attributes as part of each data row. It is only required the definition of “x”, being “x” a list of attributes representing each one of them a category.

5.7 Usability Testing

Once the mockups were done and a first sketch of the platform coded, usability tests [5, 55] were conducted. Helping us in the evaluation process and also in identifying usability problems on the designed interfaces.

5.7.1 Protocol

Fraunhofer Usability Test Protocol[2] was followed as it describes the usability testing process with a preferred level of granularity, from the selection of the users to the evaluation according to predefined metrics.

[2]Protocol available in Fraunhofer intranet.

5.7.1.1 Users

In a general context, usability testing involves a set of pre-selected and real users as "it is the most fundamental usability method" [54] and "a focus on usability requires users" [21]. Ideally, that have a background and a level of familiarization similar to the targeted end-users of the final platform since they eventually drive the design decisions through their needs.

Five users were selected because, as suggested by Nielsen [54, 53], because it is considered the optimal number as most of the usability problems are detected and an increased number is likely to not provide additional information. The selection of the users had in mind the essential characteristics needed, therefore, every user belonged to the HCI department as they are the end-users of our application as well as they have a minimum level of familiarization with technology.

5.7.1.2 Setup

The setting and type of space in which the evaluation was conducted were carefully selected in order to represent a real-life situation. It occurred in the Fraunhofer meeting room, a quiet environment without anyone surrounding the user besides the testing representative. To the user it was provided a laptop with a browser installed that was used to display and interact with the targeted website, so relevant aspects of the intended environment were simulated. Although, the test settings can probably differ from the normal context of use based on the screen resolution and the browser used. The web app was running on Firefox 67, on a laptop running Windows 8.0 through a wireless connection, with a screen resolution of 1920x1080 and a size of 15 inches.

5.7.1.3 Test procedure

To achieve our goal with the usability test, a set of procedures had to be followed. Firstly, it was important to identify what we would be testing and why it was so important, secondly, which users should be approached taking into consideration the necessary characteristics and finally, how can we evaluate and measure their performances, which metrics do we require.

That being said, all of that information took a form of a simple script. Composed by a little introduction and contextualization, a section aiming to describe the general instructions that the users would require, the tasks to be performed (a total of five tasks), a SUS (System Usability Scale) questionnaire in order to measure the satisfaction and at the end of the document, a text area for suggestions.

To the extent of preparing the usability test, a pilot was performed with a length of one session. Pilot testing helps tuning usability test conditions, in which the main objective is to test the study to make sure it goes as planned without any mishap, obtaining more reliable results as a consequence [60].

Table 5.1: Effectiveness and Efficiency results

Status	Task 1	Task 2	Task 3	Task 4	Task 5
C	3	5	0	4	5
CA	2	0	5	1	0
E	0	0	0	0	0
Total	5	5	5	5	5
T	5	5	1	1	5
NT	0	0	4	4	0
Total	5	5	5	5	5

5.7.1.4 Performance and Satisfaction Metrics

Effectiveness based on per cent task completion, frequency of errors, frequency of assists to the participant from the testers, and frequency of accesses to help or documentation by the participants during the task. Efficiency based on the time needed to finish the task and satisfaction according to SUS[3] questionnaire. A SUS is "a simple, ten-item scale giving a global view of subjective assessments of usability" [8], mainly through the final score calculation of the answers to those scales.

5.7.2 Results

5.7.2.1 Performance Results

In order to evaluate the performance of each one of the participants during the tasks, effectiveness and efficiency served as metrics. While effectiveness is focused on completeness and accuracy, efficiency relates commonly, and also in the context of our test, to the mean time required to meet the criteria for successful completion.

In Table 5.1, we present the effectiveness as well as the efficiency evaluation which was recorded during the test with the aid of a recording software previously installed on the computer. Consider: **C** - completed, **CA** - completed with assistance, **E** - Error, **T** - In time, **NT** - Not in time.

5.7.2.2 Satisfaction Results

Satisfaction describes a subjective evaluation when using the product and are applied via questionnaires. Here, the satisfaction metric was measured based on the SUS, consisting of a Likert scale composed of 10 distinct scales, each one evaluating a specific parameter. Inside the usability test script provided to the users, Appendix B, the SUS questionnaire was addressed and leading to the data collection of those answers. The score is then reflected as a value ranging from 0 to 100, classified according to Table C.1.

[3] System Usability Scale

Calculating the SUS score, we obtained 77.5 (also presented in Appendix C), which classifies the platform design as a robust "Good"[3].

5.7.3 Evaluation

As part of every test, usability or not, there is always an evaluation followed by an analysis of the results obtained. Thus, from the analysis of these metrics, it is perceptible that:

- In terms of effectiveness, all users were able to complete their tasks although in some cases requiring assistance. Tasks 1, 2 and 5 were not challenging at all, while for task 3 and 4 users showed some consistency and needed assistance, either because the guideline was not completely explicit on what should have been done or because there were design problems associated.
- In terms of efficiency, there is a strong positive correlation between the tasks that demanded assistance and the incapability to complete them in time.
- Relatively to satisfaction, it proved consistent although there is enough room to improve namely in layout.

5.7.4 Discussion

From the suggestions, it was clear there was some resemblance regarding the design problems of the platform. Although, since it was not possible to implement every single one, they were ordered by priority, mostly, considering the effects on the chart interpretation.

By a top-bottom priority, they can be distributed as follows: (i) Renaming the chart titles; (ii) the layout and organization of the components in the interface and the possibility to resize them according to the window size changes; (iii) The aggregation button in the bar chart should be highlighted and more self-explanatory; (iv) Separation between the overview area and the chart itself; Changes to the date picker by removing unnecessary buttons and making more explicit the range selection.

5.8 Summary

During this chapter, it was discussed the first component of our application architecture — the client, and the iterative path, composed by an intermediate evaluation and analysis of the interfaces, that posteriorly lead to the final product. Each one of the designed UI components has a purpose, either being the representation of data through graphical visualizations or even to improve the layout organization as well as its interpretation. All of it is achieved by defining the proper configuration that will later be reflected in the platform interfaces.

Another key component described throughout this chapter was the Usability Tests applied, where the environment and tools set up tried to mimic a real case scenario complemented with a process of carefully selecting the participants in order to provide significant and valuable feedback

that later on, conferred an additionally confidence relative to the design choices. Despite the fact that from the results obtained, the designed interfaces were satisfactory, improvements could be addressed in the future.

Chapter 6

Case Studies

The goal of this chapter is to define and describe the process that involved applying the solution developed to real case scenarios, essentially, to validate it and to that end, verifying if it fulfils the user stories raised.

Regarding that the solution developed comes as an extent of already existing projects and is built upon them, two different projects were analyzed separately namely, SmartBEAT and Lifana, in a form of case study diverging from one another in their context, database structure and configuration.

6.1 SmartBEAT

SmartBEAT is a project carried by Fraunhofer AICOS, that aims to address the needs of senior heart failure patients and their formal and informal caregivers by offering an integrated solution to leverage patient self-care through autonomous condition monitoring and real-time feedback to their carers. Patients are followed according to specific parameters input: (i) the activity level — reflecting the number of steps done by the patient; (ii) the weight (iii) heartbeat rate per minute; (iv) arrhythmia — corresponding to the irregularity of the heartbeat; (v) systolic pressure; (vi) And lastly, diastolic pressure.

All of this information is collected via an application installed on the patient smartphone. Some of these actions are executed in background, passively, without user interaction while others require the patient to be active. In this regard, parameters such as activity level are obtained based on sensors although for measuring the weight the patient has to use a proper balance and record it. Also, the application provides an interface in which questionnaires are presented to him so he can fill them and send that same information to the database.

Data Structure

This project uses a Firebase Realtime Database, and because it was already in the production phase, the data structure was already defined. In order to have the retrieved data in an acceptable format, it was necessary the definition of reducers.

The schema followed, represented through Listing 6.1, has as its root node the collection Users in which each child document refers to a patient. Inside each one of the patients' document, there are two different collections: Measures and Questionnaires. Questionnaires holding documents referring to a single answer by a unique timestamp, while Measures containing sub-collections relatively to each registry type, for instance, Activity, Heart, Height. Its sub-documents refer to each one the measures made identified by the unique timestamp of its registration.

```
{
    Users:
        "x":
            Measures:
                Activities:
                    1553212800000: 2955
                        ...
                Weight: ...
                HeartRate: ...
                    ...
            QuestionnaireResponses: ...
        ...
}
```

Listing 6.1: SmartBEAT database structure

Configuration

Here it is presented the main functionalities tested against SmartBEAT as well as their configuration in order to accomplish the end results.

6.1.0.1 Platform

The main configuration of the platform is obtained based on the file "site.yaml" and the files inside the directory "config/blueprints/pages". While the first provides the title of the platform the second gives the pages that will compose the web sidebar. Hence, in the directory "config/blueprints/-pages", two different files were added each one representing a separate page. Such configuration led to two different links in the left sidebar and the title to be set to "Smartbeat" on the platform interface.

6.1.0.2 Pages

The platform is configured having in mind two different pages being, “General Information” and “Histograms”. The first containing graphical representations such as bar charts and time-series and the second only histograms, having both pages reflecting data according to multiple sources such as heart rate, activity level, questionnaire answers, etc. On this regard, each one of these pages contains a specific configuration file. As an example of its application, Figure 6.1 comes as an illustration.

```
163 - type: time series
164   plurality: single
165   title: Activity registry
166   ylabel: Number of steps
167   xlabel: Grouped by week
168   specifications:
169    database: smartbeat
170    table: Activities
171    groupby:
172      time: week
173      operator: avg
```

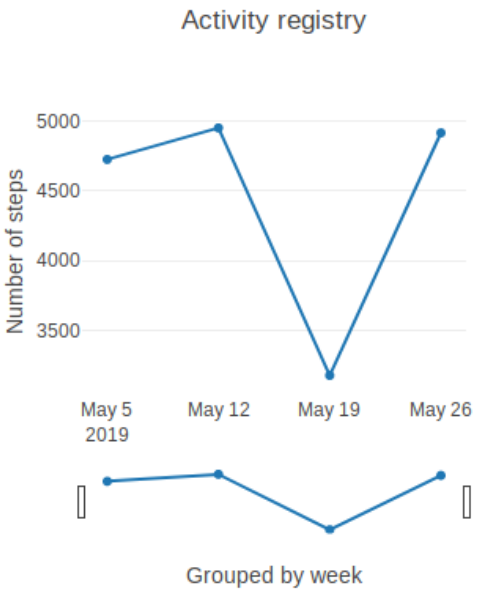


Figure 6.1: SmartBEAT time series example

6.1.0.3 Chart Properties

As we know by now, it is possible to change some properties of the chart such as title, labels and others that are restricted to a specific type of a chart. With this subsection, we intend to test and validate this functionality. Hence, a simple demonstration will be presented in order to do it.
Starting with title and labels, as seen in Figure 6.1, the configuration defined had “Activity registry‘” as title, and the x-axis and y-axis defined as “Grouped by week” and “‘Number of steps”, respectively. Furthermore, some charts differ from one another in the configuration they offer, for instance, time-series allow the aggregation of data per hour, day, week and month. While others such a histogram allows the definition of the x range and the in-between range values. Subsequently, in this example, the “groupby” was set to week and its aggregation operator to average.
Both examples are expressed through the same figure, and it is clear that the data is being grouped by week where the different four points registered to reflect the last month recordings (4 weeks). In addition, the chart side configurations are exactly the way they were defined. Having all this into consideration, we believe this proves the functionality is working as intended.

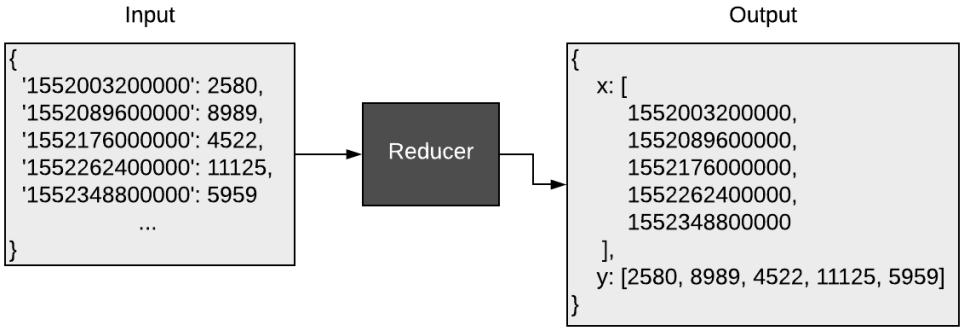


Figure 6.2: Application of a SmartBEAT reducer

6.1.0.4 Users

In this context and at the time that the validation was made, only some of the patients from the whole set were needed (as others already had finished their trial or some of the accounts were made regarding testing purposes). Thus, as we support alternatives to overcome this problem, here it was selected the approach of defining a user-mapping file, meaning that only the users presented in that file would actually appear in the platform.
The output resulted in the provided users being displayed in the form of a table, also, the user identifier was mapped accordingly to the specified via the file “config-file.yaml”. Proving once again, the desired functionality is working as expected.

6.1.0.5 Reducers

Referring back to Section 6.1, the data structure was not compatible with our convention mainly the project had already started when the validation came. So it was necessary the definition of reducers more particularly, seven reducers, which are in other words, functions that convert the data retrieved to the format that is expected to be received by the front-end in order to be able to represent it through the corresponding chart.
As specified in Figure 6.2 and for the sake of simplicity, only one reducer is represented in order to provide some guidance and explanation. Here we can see that the input data is an object with multiple entries, being timestamp the key and the activity level the value. Which, after being submitted to the reducer, a reducer related to a time-series component, it is, naturally, converted to an object with two arrays: one holding the values from the x-axis and the second the values from the y-axis.

6.2 Lifana

The last case study to which the proposed solution was confronted with, is the Lifana project. Lifana is also a project designed by Fraunhofer AICOS, that acknowledges the fact that many

elderly suffer from nutritional problems that can cause chronic health conditions such as high blood pressure or cardiovascular diseases. The primary goal of it is to develop and evaluate the Lifana Nutrition Solution that supports healthy nutrition through all phases of ageing, from active seniors to elderly users and patients in need of daily care. Individual meal recommendations are provided based on personal advice from professional nutritionists in which it is complemented with a decision support system so the patients are able to change their eating habits in order to maintain a healthier lifestyle.
This product can be seen as a recommender system that takes the form of a mobile application, where patients are able to set their preferences while having some guidance from criteria and rules set by health-care professionals. The prototype made, was submitted to field trials (that already started), applied in both Portugal and The Netherlands in order to test its usability and will have a twelve-month duration length. Besides the common analytics related to the application use (e.g. screens navigated to, mobile phones OS, periods of time which it is used, etc) being collected through Firebase, as we will see later on, there is also other information that requires to be analyzed patient by patient in order to understand how the product is evolving and if it is becoming more and more mature.

```
Users:
    "id":
        mealPlanTimestamp: 1560758076593,
            ...
        trialUser: true
    ...
SurveyNps:
    "idSurvey":
        answer: 3,
        timestamp: 1561053500331,
        user: "user_id"
SurveyPlanAdequated: ...,
SurveyPlanFollowed: ...,
```

Listing 6.2: Lifana Cloudfirestore database structure

Data Structure

The data structure is quite different comparatively to SmartBEAT, mostly because, it has two distinct databases: Cloudfirestore and MySQL, linked to each other based on the user id from Cloudfirestore and the “external_id” field from the corresponding user table in MySQL database.
As the test is having in mind the usability, design problems and functionality, a set of questionnaires from time to time appear in the screen of the application, mainly, to collect feedback. This questionnaires consist of a set of Likert scales ranging from zero to five and can evaluate, for

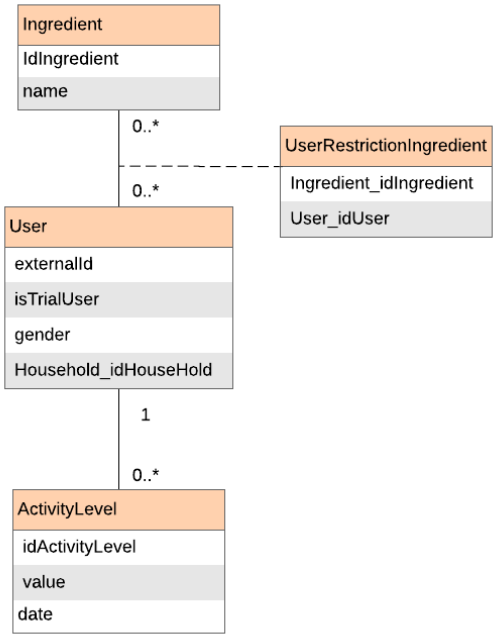


Figure 6.3: Lifana SQL structure

instance, whether the application provides adequate recommendations based on the patient preference or not, and these ones are stored in Cloudfirestore. In the other hand, information relatively to recommendations, ingredients, user preferences, are stored in MySQL.
The Cloudfirestore structure follows the schema defined as a convention by us, totally flat and denormalized. As it can be seen in Listing 6.2, there is naturally a collection of users that refer to the patients' target by the field trial, and a set of collections, similar in structure, representing the different questionnaires applied. Each one of its entries contains a reference for a unique user.
Regarding MySQL, and as aforementioned, it comports the majority of the project data. However, most of it is useless for the process of monitoring the field trial, so the simplified structure that was used for validating our application is presented in Figure 6.3. In order to test it, we chose to pick up an example representing an *N-M* and a 1-*M* relation to cover the typical set up. Moreover, there is a table named "Ingredient" that has an id, name, dose, protein and many more as attributes and naturally, a table for the participants called "User". Because a user can have multiple ingredients associated and vice-versa, there is another table that relates both of them, designated by "User-RestrictionIngredient", containing foreign keys to each one of them. In addition, there is a table "ActivityLevel" that references a participant.

Configuration

Through this section, the remaining most important functionalities, that were not tested with SmartBEAT, are covered.

6.2.0.1 Multiple Database Connections

The requirement of supporting multiple database connections arrived as a necessity of Lifana handling two distinct databases. Therefore, it is here that we will be testing this feature by defining in the file "site.yaml", two different databases components. As we can see in Listing 6.3, the "databases" properties consist in a list of two databases, one being a MySQL database where its connection is made accordingly to the URI specified, and the other being a Cloudfirestore database, where the connection is made based on the property "config". The establishment of these connections is in conformity to the Section 4.8.2.2, however, for obvious reasons, the configuration accesses are not the ones defined in the example.

```
187 databases:
188   - id: lifanaMySQL
189     type: mysql
190     uri: mysql://username:password@i-662.cloud.fraunhofer.pt
            :30306/CordonGrisPT
191         ...
192   - id: lifanaFirebase
193     type: firebase
194     subtype: cloudfirestore
195     config:
196       apiKey: 'random_key'
197       authDomain: 'lifana-app-abcdef.firebaseapp.com'
198       databaseURL: 'https://lifana-app-abcdef.firebaseio.com'
199       projectId: 'lifana-app-abcdef'
200       storageBucket: 'lifana-app-abcdef.appspot.com'
201       messagingSenderId: '63512332875'
202         ...
```

Listing 6.3: Lifana multiple database configuration

Once the server starts, the initial process involves the connection to all databases which for SQL, in specific, also involves the incorporation of the models and their associations. Through this example, it is shown that both connections were established correctly so, consequently, the server started listening and handling requests normally.

6.2.0.2 UI Filtering

Another functionality from our platform is the filtering that can be applied to both participants and the time range. For testing the time filtering, as demonstrated in Figure 6.4, it was defined as a custom time range from 1st to 30th of June. The time-series charts, represented on the whole page, changed accordingly to the date set.
In relation to participants filter, it is possible to switch between an overview and more focused view through the selection of a particular participant or by clearing its selection. This feature was

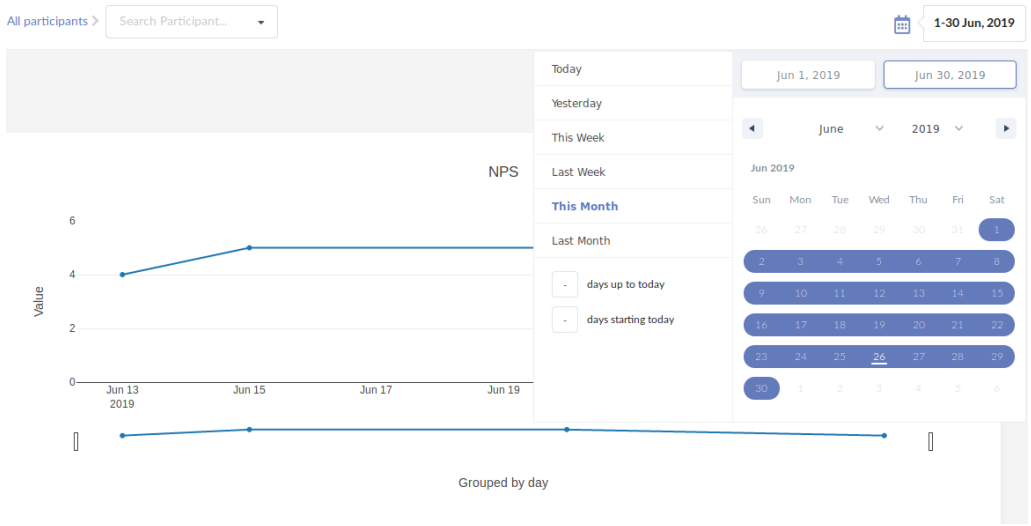


Figure 6.4: Lifana time filtering example

tested based on two examples, the first representing a case scenario where there is not any user selected so the aggregated data is retrieved while the second representing that when selecting a participant, for instance the participant number 29, the chart changes its data accordingly to that same participant.

6.2.0.3 Modelling and query

A crucial feature from our application is the SQL support, which is achieved through the use of the Sequelize library. Also important, are the features implemented based on the offered API such as the inner join, attribute and filtering selection. Regarding the fact that Lifana uses, as mentioned before, a MySQL database, it served as a validation of these aspects.
In the example below, Listing 6.4, the bar chart represented aims to act as a histogram by providing the frequency of each ingredient in the food restrictions defined among all participants from the field trial. In order to work, the database structure for SQL was modelled according to the specifications stated in Section 4.8.2.4, where both “Ingredient” and “User” tables are linked to each other based on the “through” property that is defined as being “UserRestrictionIngredient”. Not only *N-M* relations are possible but also 1-*M*. We set up an example where a time-series chart is used to represent the activity level that is collected over the past days related to each user. The “ActivityLevel” table contains a reference to a user, a date which is stored in the format “yyyy-mm-dd hh:mm:ss” and a value, corresponding to the level registered. The modelling procedure involved, as shown in Listing 6.5, the definition of a “hasMany” relation in the table “User”, as it is the one being referenced, and the complementary relation “belongsTo” declared inside “ActivityLevel”.

```
structure:
  - User:
      PK: idUser
      relations:
        - type: belongsToMany
          target: Ingredient
          through:
              UserRestrictionIngredient
          FK: User_idUser
  - UserRestrictionIngredient:
      atributes: [Ingredient_idIngredient
          , User_idUser]
  - Ingredient:
      PK: idIngredient
      atributes: [name]
      relations:
      - type: belongsToMany
        target: User
        through:
            UserRestrictionIngredient
        FK: Ingredient_idIngredient
```

Listing 6.4: Lifana *N-M* modelling results

```
structure:
  - User:
          ...
      relations:
              ...
        - type: hasMany
          target: ActivityLevel
  - ActivityLevel:
      PK: idActivityLevel
      attributes: [value, date]
      relations:
        - type: belongsTo
          target: User
          FK: User_idUser
```

Listing 6.5: Lifana 1-*M* modelling results

6.3 Summary

This chapter had the established goal of validating the solution proposed and developed during the past months. Here, in order to cover a wider number of user stories we tested our solution against two different projects that came to be, throughout the whole chapter, case studies. We started by doing a brief description and contextualization followed by an analysis of the database structure and only then we proceed to enumerate the tests performed. Since both projects differed in their database structure and configuration, thus, different levels of testing were done, as we tried to not repeat the same tests for different projects, we ended up validating and prove that the application works as expected. Although, worth to mention that, improvements could be addressed in the future.

Chapter 7

Conclusions and Future Work

In this chapter, the conclusions derived from the work carried on throughout the conception of this dissertation are presented. In addition, these conclusions are complemented with a description of the features that were not implemented but that could be considered as future work.

7.1 Conclusions

Remote monitoring a field trial through a web application has the advantage of being simple to use as well as being accessible everywhere, although building one from scratch whenever there is a field trial comes at a great cost. Be that as it may, the work demonstrated here comes, in simple words, in the form of a web platform representing agnostic data and that could be adapted to multiple projects.

During the literature review in Chapter 2, we were able to understand the main categorization of data based on their characteristics into types (e.g. graphs, charts, maps, diagrams, etc). Not only that but also, the data visualization techniques that are applied in the current days and how they represent the data, providing better or worse insights compared to others based on these same data characteristics. Another key factor found was related to the conception of systems based on metadata, a structural way of defining a configuration such that it can be interpreted by a computer according to an existing encoding schema represented such as JSON or YAML. As a sense of inspiration from all the information and knowledge obtained during this phase of literature review, the application built follows a 3-tier architecture composed by a client, a server and a database provider. The server is responsible for providing services to the client while requesting data from the database, thus acting as a mediator between the two parts.

Having in mind the goal of this dissertation as well as the target end-users (researchers), it was important to have a deeper understanding of the needs and requirements of the platform, which led to the conception of two semi-structured interviews aiming to both researchers and developers. From the developers, we were able to identify common databases used, data structures, data

registration frequency while from the researchers we intended to understand common problems but also good practices related to the user interfaces and information needs. The development of the mediator had considered the most important database types in order to support them, that consequently, led to the usage of Firebase databases and Sequelize as an abstraction to SQL ones. On the other hand, the conception of the user interfaces involved analysing different interfaces and well-design dashboards and once extracted the design guidelines filtered by our necessities, we were able to conduct usability tests to evaluate them according to specific metrics. The results from the usability tests were positive, although at the same time revealing a few design problems. Afterwards, the application was subject of testing for two different case studies with different contexts, database types and structure. From this, we were able to test most of the features that were initially established, thus, giving us confidence about the correctness of them. In conclusion, the proposed solution confirms our hypothesis and demonstrates that is possible to use a configurable application for monitoring field trials.

7.2 Future Work

This section aims to explain the work that should be carried out, having in mind a mature and robust product built upon the current state of the application. For the sake of simplicity, only the most important are enumerated:

Alert System — The alert system, for this platform, could consist in defining a set of rules in the database configuration file. That, whenever a value for a specified attribute surpassed a defined threshold, either a notification would be generated requiring for that purpose the storage of it (e.g. file system, local or remote database, etc) or the number of trespassed rules would be provided when retrieving the database data.

YAML Validation — This feature is one of the most important since it could immensely decrease the time required for configuring a new instance of the platform. Ideally, it consisted in having a set of defined rules for the configuration syntax, therefore, in case the user misspelt a word, inserted a non-existing property or provided the wrong property value, a message would be displayed. There are multiple ways of doing this; example of them are: (i) whenever the service starts, it firstly reads and analyzes all the files inside the "config" folder, through a JavaScript module that should be implemented, and only then it proceeds with the normal execution; (ii) the use of a JavaScript library named "YAML-Validator" that already allows the definition of the syntax to be confronted with; (iii) a Visual Studio Code plugin that could give, in real time, suggestions or prompt the errors found.

Compare Data Between Participants — Such a feature would allow the researcher to analyze multiples participants having the same reference through a single graphical visualization. It would be quite a useful tool for a researcher in order to compare different participants progression during the same interval of time.

Authentication — The implementation of an authentication system could allow the definition of roles, thus, each role could have restricted permissions on what information could it have access to.

Additional Representation — More graphical representations could be supported, for instance, to represent tabular data.

Overall, the implementation of these features would improve the user experience perceived by both researchers and developers, while facilitating the process of generating an instance of a platform.

References

[1] Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *The Craft of Information Visualization*, pages 7–13. Elsevier, 2003.

[2] Chandrajit Bajaj. *Data Visualization Techniques*. John Willey & sons, 1998.

[3] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[4] Louise Barkhuus and Jennifer A. Rode. From mice to men – 24 years of evaluation in chi, 2007.

[5] Brenda Battleson, Austin Booth, and Jane Weintrop. Usability testing of an academic library web site: a case study. *The Journal of Academic Librarianship*, 27(3):188–198, 2001.

[6] Alan Beaulieu. *Learning SQL: Master SQL Fundamentals*. " O'Reilly Media, Inc.", 2009.

[7] Michael Blakeley, Syme Kutz, and Carl Backstrom. Web-based user interface for searching metadata-driven relational databases, November 30 2010. US Patent 7,844,587.

[8] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[9] Barry Brown, Stuart Reeves, and Scott Sherwood. Into the wild: challenges and opportunities for field trial methods. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1657–1666. ACM, 2011.

[10] Andreas Buja, John Alan McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *Visualization*, pages 156–163. IEEE, 1991.

[11] Mackinlay Card. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[12] Stuart K Card. *The psychology of human-computer interactionCard, S. K. (2017). The psychology of human-computer interaction.* CRC Press, 2017.

[13] Chun-houh Chen, Wolfgang Karl Härdle, and Antony Unwin. *Handbook of data visualization*. Springer Science & Business Media, 2007.

[14] Ge Jackie Chen. *Visualizations for mental health topic models*. PhD thesis, Massachusetts Institute of Technology, 2014.

[15] Edgar F Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.

[16] Adrien Coyette, Suzanne Kieffer, and Jean Vanderdonckt. Multi-fidelity prototyping of user interfaces. In *IFIP Conference on Human-Computer Interaction*, pages 150–164. Springer, 2007.

[17] Adrien Coyette and Jean Vanderdonckt. A sketching tool for designing anyuser, anyplatform, anywhere user interfaces. In *IFIP Conference on Human-Computer Interaction*, pages 550–564. Springer, 2005.

[18] Patrick M Dengler, Arvind K Krishnan, Jagdish Singh, Lawrence M Sanchez, Sai Shankar, Satish Kumar Chittamuru, Zoltan Pekic, Nabarun Mondal, Namendra Kumar, Ricard Roma i Dalfó, et al. Metadata driven user interface, January 10 2012. US Patent 8,095,565.

[19] Aniruddha M. Deshpande, Richard N. Shiffman, and Prakash M. Nadkarni. Metadata-driven delphi rating on the internet. *Computer Methods and Programs in Biomedicine*, 77(1):49 – 56, 2005.

[20] Ronald J Dovich and Peter J Eppele. Metadata-driven data presentation module for database system, October 23 2001. US Patent 6,308,168.

[21] Joseph S Dumas, Joseph S Dumas, and Janice Redish. *A practical guide to usability testing*. Intellect books, 1999.

[22] Erik Duval, Wayne Hodgins, Stuart Sutton, and Stuart L Weibel. Metadata principles and practicalities. *D-lib Magazine*, 8(4):1082–9873, 2002.

[23] Usama Fayyad, Georges G Grinstein, and Andreas Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[24] Artemij Fedosejev. *React. js Essentials*. Packt Publishing Ltd, 2015.

[25] J-D Fekete and Catherine Plaisant. Interactive information visualization of a million items. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 117–124. IEEE, 2002.

[26] Joaquim Ferreira, Muhammad Alam, Bruno Fernandes, Luis Silva, João Almeida, Lara Moura, Rui Costa, Giovanni Iovino, and Elena Cordiviola. Cooperative sensing for improved traffic efficiency: The highway field trial. *Computer Networks*, 143:82–97, oct 2018.

[27] Stephen Few. Information dashboard design. 2006.

[28] Stephen Few and Perceptual Edge. Dashboard confusion revisited. *Perceptual Edge*, pages 1–6, 2007.

[29] Firebase. Choose a Database: Cloud Firestore or Realtime Database . `https://firebase.google.com/docs/database/rtdb-vs-firestore`. [Online; accessed 18-February-2019].

[30] Michael B First. The importance of developmental field trials in the revision of psychiatric classifications. *The Lancet Psychiatry*, 3(6):579–584, jun 2016.

[31] Kathryn Fitch, Steven J Bernstein, Marfa D Aguilar, Bernard Burnand, and Juan R LaCalle. The RAND/UCLA appropriateness method user's manual. Technical report, RAND CORP SANTA MONICA CA, 2001.

[32] Michael Friendly. A brief history of data visualization. In *Handbook of data visualization*, pages 15–56. Springer, 2008.

[33] Cory Gackenheimer. *Introduction to React*. Apress, 2015.

[34] Amy Genender-Feltheimer. Visualizing High Dimensional and Big Data. *Procedia Computer Science*, 140:112–121, jan 2018.

[35] Terry Halpin and Tony Morgan. *Information modeling and relational databases*. Morgan Kaufmann, 2010.

[36] James Douglas Hamilton. *Time series analysis*, volume 2. NJ: Princeton university press, Princeton, United States of America, 1994.

[37] Jeffrey Heer, Michael Bostock, Vadim Ogievetsky, et al. A tour through the visualization zoo. *Commun. Acm*, 53(6):59–67, 2010.

[38] IDC. The Digital Universe of Opportunities. `https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf`. [Online; accessed 14-December-2018].

[39] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. Understanding object-relational mapping: A framework based approach. *International Journal on Advances in Software Volume 1, Numbers 2&3, 2009*, 2009.

[40] Channu Kambalyal. 3-tier architecture. *Retrieved On*, 2, 2010.

[41] Daniel A. Keim. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization & Computer Graphics*, 7(14):8, 2002.

[42] Andreas Kerren, John Stasko, Jean-Daniel Fekete, and Chris North. *Information Visualization: Human-Centered Issues and Perspectives*, volume 4950. Springer, 2008.

[43] Stephen M. Kosslyn. Understanding charts and graphs. *Applied Cognitive Psychology*, 3(3):185–225, jul 1989.

[44] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12:45–50, 11 1995.

[45] Steve LaValle, Eric Lesser, Rebecca Shockley, Michael S Hopkins, and Nina Kruschwitz. Big data, analytics and the path from insights to value. *MIT sloan management review*, 52(2):21, 2011.

[46] Matthew L Lee and Anind K Dey. Lifelogging memory appliance for people with episodic memory impairment. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 44–53. ACM, 2008.

[47] John Light. Information visualization in data mining and knowledge discovery. chapter Portable Document Indexes, pages 99–102. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[48] Paul D Manuel and Jarallah AlGhamdi. A data-centric design for n-tier architecture. *Information Sciences*, 150(3-4):195–206, apr 2003.

[49] Marius Muji. Metadata Repositories in Database-driven Information Systems. *Procedia Technology*, 19:816–819, 2015.

[50] Scott Murray. *Interactive Data Visualization for the Web: An Introduction to Designing with*. O'Reilly Media, Inc., 2017.

[51] National Information Standards Organization (U.S.). *Understanding metadata.* NISO Press, 2004.

[52] Kawa Nazemi, Dirk Burkhardt, David Hoppe, Mariam Nazemi, and Jörn Kohlhammer. Web-based Evaluation of Information Visualization. *Procedia Manufacturing*, 3:5527–5534, jan 2015.

[53] Jakob Nielsen. How many test users in a usability study? NNGroup.com [Online; posted 04-June-2012].

[54] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[55] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[56] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[57] Craig Russell. Bridging the object-relational divide. *Queue*, 6(3):18–28, 2008.

[58] Silvia Santini and Daniel Rauch. Minos: A generic tool for sensor data acquisition and storage. In *19th International Conference on Scientific and Statistical Database Management IEEE*, 2008.

[59] Dimas Gilang Saputra and Fazat Nur Azizah. A Metadata Approach for Building Web Application User Interface. *Procedia Technology*, 11:903–911, jan 2013.

[60] Amy Schade. Pilot testing: Getting it right (before) the first time. NNGroup.com [Online; posted 05-April-2015].

[61] Ben Steichen, Giuseppe Carenini, and Cristina Conati. User-adaptive information visualization: using eye gaze data to infer visualization tasks and user cognitive abilities. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 317–328. ACM, 2013.

[62] Edward Tufte. *The visual display of quantitative informations 2nd ed.* Graphics Press, Cheshire, Conn., 2001.

[63] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[64] LO Yusuf, O Folorunso, AT Akinwale, and AI Adejumobi. Visualizing the behaviour of reinforced concrete beam structure under various types of loadings. *African Journal of Mathematics and Computer Science Research*, 2(10):202–217, 2009.

[65] Qi Zhang. Web-based medical data visualization and information sharing towards application in distributed diagnosis. *Informatics in Medicine Unlocked*, 2018.

Appendix A

Semi-structured Interviews

Semi-structured Interview - Researchers

<Introdução>
Bom dia, o meu nome é Pedro Lima e no âmbito da minha dissertação de mestrado terei de conceber uma plataforma para monitorização de pilotos (field trials). Aquilo que é pretendido é que esta seja genérica, podendo ser adaptada a outros projetos com o intuito de facilitar o trabalho dos investigadores. No entanto, é preciso primeiro perceber os end-users (investigadores), o processo atual, as dificuldades encontradas e possíveis sugestões, sendo esse o objetivo fundamental desta entrevista.

Q1. Com base na metodologia aplicada até então, como decorre atualmente o processo de monitorização dos participantes durante um piloto?
- **Q1.1.** Consegue-me enumerar as principais dificuldades com que se depara?
- **Q1.2.** Consegue dar exemplos reais para cada uma dessas dificuldades?
- **Q1.3**. Falou de dificuldades em X, considera que existe algo que possa ser feito a esse respeito?
- **Q1.4.** De acordo com o que mencionou, quais as condições necessárias para se encontrar na presença de um cenário ideal?

Q2. Que tipo de registos são normalmente feitos e que viabilizam a interpretação dos dados?
- **Q2.1.** É possível demonstrar e exemplificar com casos reais?

Q3. Quais as principais métricas que são analisadas aquando do acompanhamento dos participantes? (Médias, somatórios, números totais, etc)
- **Q3.1.** Existe alguma em especial que considera relevante e que não é tida em conta?

Q4. Quais são as visualizações gráficas mais utilizadas em projetos passados?
- **Q4.1.** Essas visualizações são satisfatórias face ao propósito? Em caso negativo, porque não o são?
- **Q4.2.** Segundo a sua opinião, dentro desse espectro, quais são aquelas que transmitem com mais eficácia a informação?
- **Q4.3.** As visualizações comportam algum tipo de interação com o utilizador? Por exemplo zoom, selection, etc. Julga ser uma mais valia ou representações estáticas satisfazem plenamente as necessidades?
- **Q4.4.** Que outras possíveis representações acha que fariam sentido incorporar e que eventualmente ajudariam neste processo?
- **Q4.5.** Seria útil poder exportar esses gráficos?

Q5. Consegue enumerar quais as principais características comuns nas plataformas web enquanto ferramenta de monitorização?
- **Q5.1.** Essas mesmas características são úteis ou podem ser melhoradas?
- **Q5.2.** Que features devesse a plataforma ser dotada de forma a facilitar o seu trabalho?

Semi-structured Interview - Developers

<Introdução>
Bom dia, o meu nome é Pedro Lima e no âmbito da minha dissertação de mestrado terei de conceber uma plataforma para monitorização de pilotos (field trials). Aquilo que é pretendido é que esta seja genérica, podendo ser adaptada a outros projetos com o intuito de facilitar o trabalho dos investigadores. No entanto, é preciso primeiro perceber os end-users (investigadores), o processo atual, as dificuldades encontradas e possíveis sugestões, sendo esse o objetivo fundamental desta entrevista.

Q1. Pode descrever, de forma geral, como é feito atualmente o registo de dados dos pilotos e de que forma?
Q1.1. A solução é feita de raiz ou servindo-se de ferramentas já existentes?
Q1.2. (Se serve-se de ferramentas já existentes) Que ferramentas ou soluções são essas adotadas para auxiliar no processo de analytics? (Escolha de base de dados, paradigmas - SQL ou NoSQL).
Q1.3. Ordene-as da mais utilizada para a menos utilizada.
Q1.4. (No caso de mencionar Firebase) Que tipo é usado Cloud Firestore ou Realtime database?
Q1.5. De acordo com a sua experiência, no futuro pensa voltar a usar essas ferramentas?

Q2. Que registos são usualmente feitos?
Q2.1. Pode demonstrar com exemplos? Se possível através de um diagrama UML.
Q2.2. Pode falar-me mais acerca desses exemplos?
Q2.3. Tipicamente, qual a dimensão média da base de dados, tendo em conta apenas as usadas para processos de monitorização.

Q3. Em projectos que envolvem pilotos com utilizadores quais as principais estruturas de dados usadas (classificações/ratings, contagens, frequências, medições)?

Q4. Os dados sujeitos à monitorização são normalmente de que domínio? (categóricos, discretos, contínuos ou textuais)
Q4.1 Quais os mais e menos frequentes?

Q5. Os dados coletados têm, sempre ou maioritariamente das vezes, um registo temporal associado? (Por exemplo à semelhança do Sequelize que guarda o registo do createdAt e updatedAt)
Q5.1 Esses dados estão sempre associados a um participante, certo? (Confirmação)

Q6. A base de dados para efeitos de monitorização encontra-se normalmente integrada com a base de dados do projeto ou separada? São independentes (i.e as tabelas para analytics pode não precisar de relações/ligações com as tabelas relacionadas com a lógica de negócio da aplicação se for o caso)

Appendix B

Usability Test Document

Informação pessoal:

Idade: ___

Browser usado normalmente: _____

Instruções gerais:

O sistema encontra-se dividido em múltiplas seções. A secção 'Walking Activity' alberga os registos relacionados com o número de passos dados pelos participantes. A secção 'Questionnaire responses' dados acerca da resposta dada pelos participantes ao diferentes questionários. A secção 'Swimming Activity', contendo informação relativa à prática de natação e por fim, 'Heart Measurements' contendo registos de batimento cardíaco e da sua irregularidade assim como da pressão sistólica e diastólica.

Tarefas:

1. Identifique o número máximo de passos dados num dia, pelo participante número 1, durante o mês de Fevereiro.

2. Agora, faça o mesmo mas analisando a partir da informação agregada de todos os utilizadores do sistema.

3. Consegue enumerar os intervalos de números de passos, nos quais o participante número dois possui uma frequência equivalente a zero?

4. Identifique agora, ainda para o mesmo participante, os valores máximos e mínimos registados para cada uma das medidas coletadas referentes ao coração (batimento cardíaco por minuto, irregularidade, pressão sistólica e diastólica).

5. Procure pelo utilizador número 50 na lista de participantes. Seguidamente, remova-o enquanto participante selecionado.

Questionário - SUS (System Usability Scale)

1. Eu acho que gostaria de usar este sistema com frequência.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

2. Eu acho o sistema desnecessariamente complexo.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

3. Eu achei o sistema fácil de usar.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

4. Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

5. Eu acho que as várias funções do sistema estão muito bem integradas.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

6. Eu acho que o sistema apresenta muita inconsistência.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

7. Eu imagino que as pessoas aprenderão como usar este sistema rapidamente.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

8. Eu achei o sistema dificil de usar.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

9. Eu senti-me confiante ao usar o sistema.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

10. Eu precisei de aprender várias coisas novas antes de conseguir usar o sistema.

Discordo Totalmente 1	2	3	4	Concordo Totalmente 5
☐	☐	☐	☐	☐

Sugestões:

Appendix C

System Usabiliy Scale

C.1 Observations

The script given to the participants for the usability test was composed, at the end of it, by a text area for suggestions and observations where they were encouraged to provide relevant feedback which consequently, that feedback aided us in understanding existing design problems and where we should have focused our attention. Despite the time available, only a few could be taken into the development stage and the rest of it to be considered in future work.
Here, both verbal and textual feedback given by the participants will be presented in conformity to the interviews.

C.1.1 Participant 1

Age: 32
Common used browser: Chrome

- "The charts should have a better designation since it was a little confusing, for instance, the one containing the heart measurements, Bp registry as a title is not that clear."
- "Better adaption of the components to the window size."

C.1.2 Participant 2

Age: 22
Common used browser: Mozilla Firefox and Opera

- "In the option to adjust the temporal range of the data visualized on the chart, the overview should be separated from the rest of the chart and include an icon allusive to the functionality (a magnifying glass icon, as an example"
- "The range selection of dates should be explicit in some manner."

- “Should highlight the element of the selection of the aggregation operator (min, max, etc) displayed in the "Bp by attribute" chart.”
- “The chart titles should be renamed in order to be more descriptive.”

C.1.3 Participant 3

Age: 23
Common used browser: Chrome and Mozilla Firefox

- “Internal buttons that display the date from the date picker are not necessary since they are just repeating information and also, they have the form of a button but are not interactable.”
- “In the heart measurements chart it is not perceptible the aggregation being made if it is the average of the maximum, the minimum or the average of values.”
- “The second chart was not visible, a better layout of the components should be considered."

C.1.4 Participant 4

Age: 25
Common used browser: Firefox

- “In my particular case, because of my practice, my eyes were focused too much on the sidebar menu and in the central area. Minor options in the upper right corner of the screen weren’t inside of my field of vision. Although it was just a question of paying more attention.”

C.1.5 Participant 5

Age: 27
Common used browser: Chrome

- “Platform should have a better schema color, a set of colors less fluorescent.”
- “Search should be more intuitive.”

C.2 SUS Score Calculation

Table C.1: SUS system classification

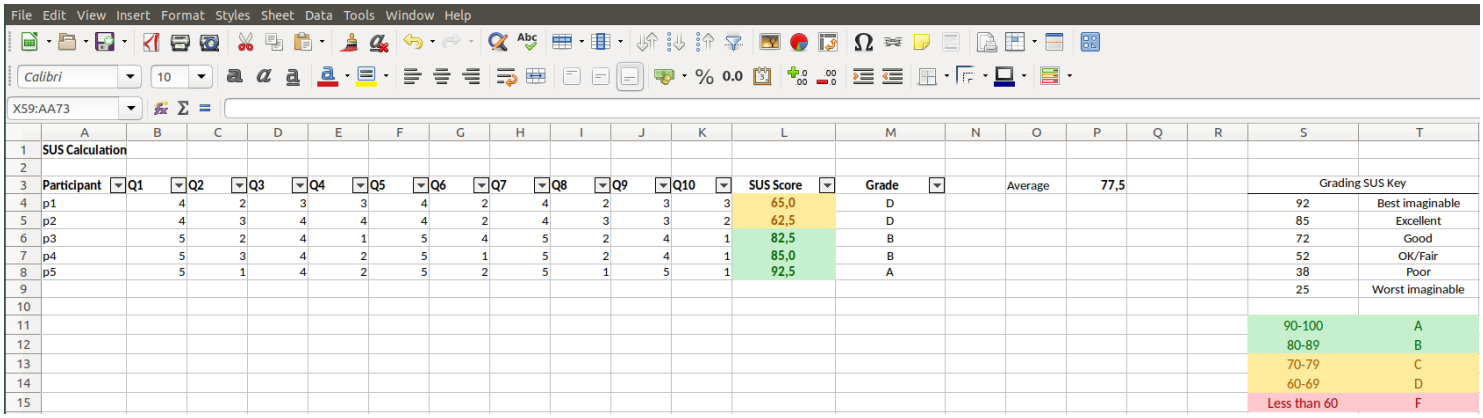
>= 95	>= 85	>= 72	>= 52	>= 38	< 38
Best imaginable	Excellent	Good	OK/Fair	Poor	Worst imaginable

The score calculation is done according to a specific set of rules [8]:

Table C.2: Satisfaction results from SUS

User	Scale 1	Scale 2	Scale 3	Scale 4	Scale 5	Scale 6	Scale 7	Scale 8	Scale 9	Scale 10
U1	4	2	3	3	4	2	4	2	3	3
U2	4	3	4	4	4	2	4	3	3	2
U3	5	2	4	1	5	4	5	2	4	1
U4	5	3	4	2	5	1	5	2	4	1
U5	5	1	4	2	5	2	5	1	5	1
Mean	4.6	2.16	3.8	2.4	4.6	2.2	4.6	2	3.8	1.6
Standard Deviation	0.55	0.75	0.45	1.14	0.55	1.1	0.55	0.7	0.84	0.89
Max	5	3	4	4	5	4	5	3	5	3
Min	4	1	3	1	4	1	4	1	3	1

1. Sum the contributions from each question.
2. For every question with an odd number (1,3,5,7,9), subtract the scale position by 1.
3. For every question with an even number, subtract the scale position from 5.
4. Multiply the sum of the scores by 2.5.



File Edit View Insert Format Styles Sheet Data Tools Window Help

Calibri 10 X59:AA73

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	**SUS Calculation**																			
2																				
3	**Participant**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**SUS Score**	**Grade**		Average	77,5			Grading SUS Key	
4	p1	4	2	3	3	4	2	4	2	3	3	65,0	D						92	Best imaginable
5	p2	4	3	4	4	4	2	4	3	3	2	62,5	D						85	Excellent
6	p3	5	2	4	1	5	4	5	2	4	1	82,5	B						72	Good
7	p4	5	3	4	2	5	1	5	2	4	1	85,0	B						52	OK/Fair
8	p5	5	1	4	2	5	2	5	1	5	1	92,5	A						38	Poor
9																			25	Worst imaginable
10																				
11																			90-100	A
12																			80-89	B
13																			70-79	C
14																			60-69	D
15																			Less than 60	F

Appendix D

Usability Test Scenarios, Tasks and Preparation

D.1 Preparation

To the extent of preparing the usability test, a pilot was performed with a length of one session. Pilot testing helps tuning usability test conditions, in which the main objective is to test the study to make sure it goes as planned without any mishap, obtaining more reliable results as a consequence [60]. There are many advantages from its use, but in the context of our study the most important were: (i) Rehearsal - it prepared us for improvisations just as in a real-life situations and guaranteed that the website was working, the script printed and the recording software installed; (ii) Tests the tasks - It allowed us to test the tasks, if they were possible and if the written text was self-explanatory or it needed further explanation; (iii) Timing - We were able to test how much time was required for each task and estimate the possible duration of each interview;
After this test, the script was reformulated as well as the instructions given to the users. Nevertheless, since the user participating in this pilot already knew a priori the platform, the timing stipulated was somehow biased.

D.2 Scenarios

Scenario 1: The maximum activity level registered and the corresponding day for participant number 1 in February 2019
Sub-tasks: (i) Click on the Activity tab displayed in the left sidebar of the screen; (ii) Select participant number 1; (iii) Select the whole February month; (iv) Hover the maximum registered value;
Criteria for successful completion of the goal of each scenario: Participant identified the maximum level as 7675 and the day as 9th of February.
Maximum time limit for completing the scenario: 60 seconds.

Policies and procedures for interaction between tester(s) and test participants: Whenever the participant asks for help or takes too long to complete a task or a sub-task, the tester is allowed to intervene.

Scenario 2: The maximum activity level registered and the corresponding day considering the overall participants' registry.
Sub-tasks: Starting on the home page: (i) Click on the Activity tab displayed in the left sidebar of the screen; (ii) Select the whole February month if not selected already; (iii) Hover the maximum registered value;
Starting on the activity page with a previously selected participant: (i) Click either on the drop-down clear button or on 'all participants' link; (ii) Select the whole February month if not selected already; (iii) Hover the maximum registered value;
Criteria for successful completion of the goal of each scenario: Participant identified the maximum level as 7675 and the day as 9th of February.
Maximum time limit for completing the scenario: 30 seconds + 30 depending on the starting circumstances.
Policies and procedures for interaction between tester(s) and test participants: Whenever the participant asks for help or takes too long to complete a task or a sub-task, the tester is allowed to intervene.

Scenario 3: The range of steps with a frequency of 0 for participant number 2.
Sub-tasks: (i) Click on the Activity tab displayed in the left sidebar of the screen if not selected already; (ii) Select participant number 2; (iii) Select the whole February month if not selected already; (iv) Hover the ranges with a zero frequency;
Criteria for successful completion of the goal of each scenario: Participant identified the intervals as [3600, 4000 [and [4400, 4800 [.
Maximum time limit for completing the scenario: 30 seconds + 30 depending on the starting circumstances.
Policies and procedures for interaction between tester(s) and test participants: Whenever the participant asks for help or takes too long to complete a task or a sub-task, the tester is allowed to intervene.

Scenario 4: Minimum and maximum value registered for each one of the Heart metrics for participant number 2.
Sub-tasks: (i) Click on the Bp tab displayed in the left sidebar of the screen if not selected already; (ii) Select participant number 2 if not selected already; (iii) Select the max opt presented in graph UI dropdown; (iv) Hover each one of the columns; (v) Select the min option presented in graph UI dropdown; (vi) Hover each one of the columns;
Criteria for successful completion of the goal of each scenario: Participant identified the maximum values as [75, 90, 10, 134] and minimum values as [45, 60, 0, 104]

Maximum time limit for completing the scenario: 75 seconds.
Policies and procedures for interaction between tester(s) and test participants: Whenever the participant asks for help or takes too long to complete a task or a sub-task, the tester is allowed to intervene.

Scenario 5: Search participant number 50 and then clear the participant selection field.
Sub-tasks: (i) Select participant number 50 by searching or scrolling the corresponding dropdown; (ii) Click on the dropdown cross button;
Criteria for successful completion of the goal of each scenario: Participant number 50 is selected and ultimately filter is cleared.
Maximum time limit for completing the scenario: 30 seconds.
Policies and procedures for interaction between tester(s) and test participants: Whenever the participant asks for help or takes too long to complete a task or a sub-task, the tester is allowed to intervene.

D.3 Participant Task Instructions

1. Can you identify the maximum number of steps registered in February for participant number 1?

2. Now, can you do the same thing but for the overall users of the system?

3. Can you enumerate, the ranges, in which participant number 2, has a frequency of zero according to the number of steps?

4. For each one of the heart metrics being measured, identify the maximum and minimum values for participant number 2.

5. Search for participant number 50 among the participants displayed in the dropdown. After, deselect it.

D.4 Participant general instructions

Orientation to the test context and consents given to the participants: Whenever the participant felt stuck onto something or couldn't understand how to overcome the task provided, the tester would assist him. In any case, it was not the user that was being tested but the system usability instead. All the information needed to solve each task was provided with a priori.

General instructions to be given to the participants: The system is divided into multiple sections. The 'Walking Activity' section contains the records related to the number of steps taken by the participants. The 'Questionnaire responses' section gives information about the response

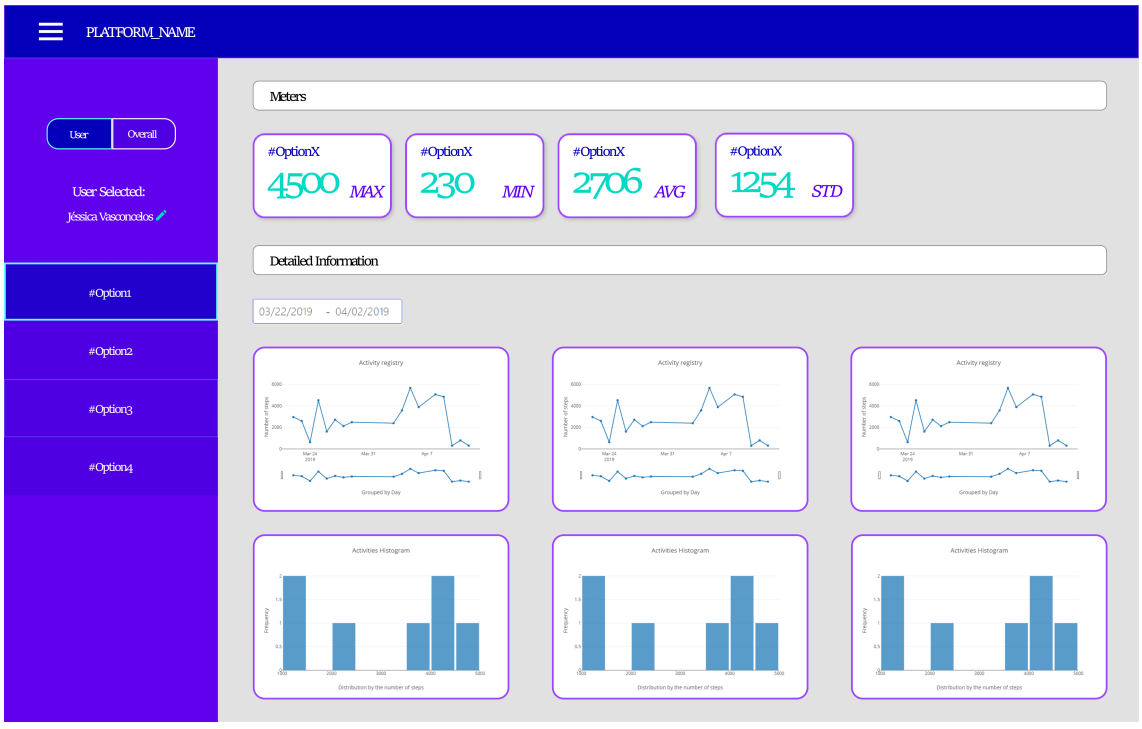
given by the participants to the different questionnaires. The 'Swimming Activity' section, containing information on swimming practice and, finally, 'Heart Measurements' containing heart rate records and their irregularity as well as systolic and diastolic pressure.

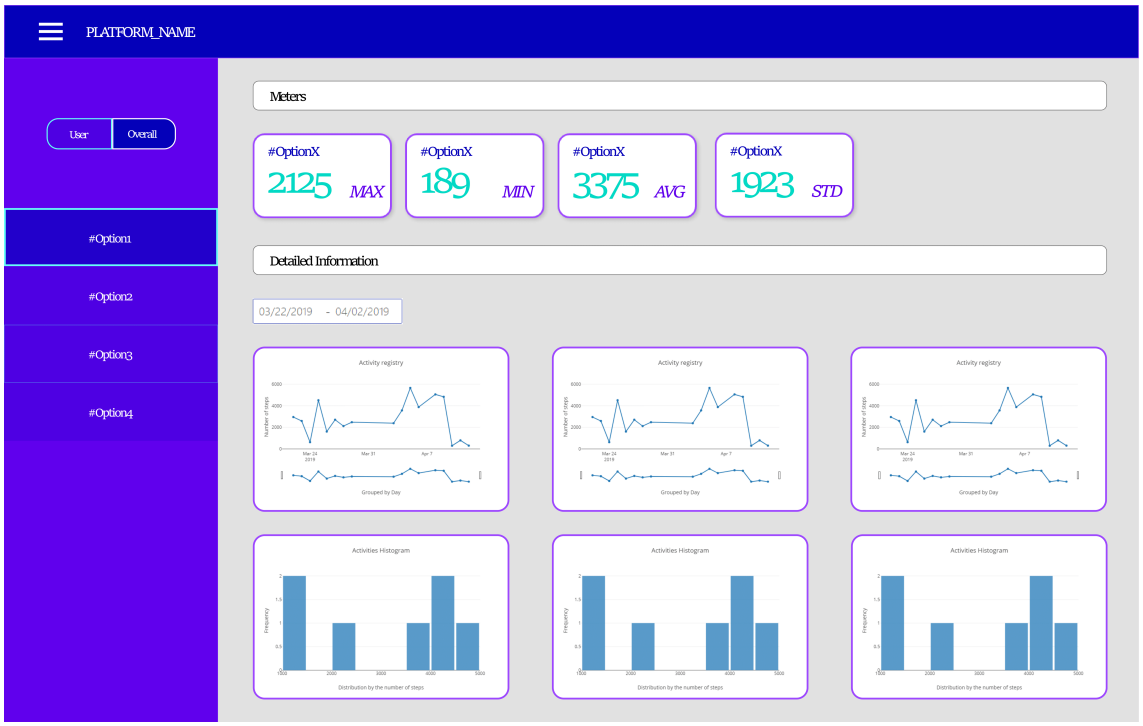
Instructions on how participants were to interact with any other persons present, including how they were to ask for assistance or interact with other: Participants could either call the tester or raise their hand for assistance.

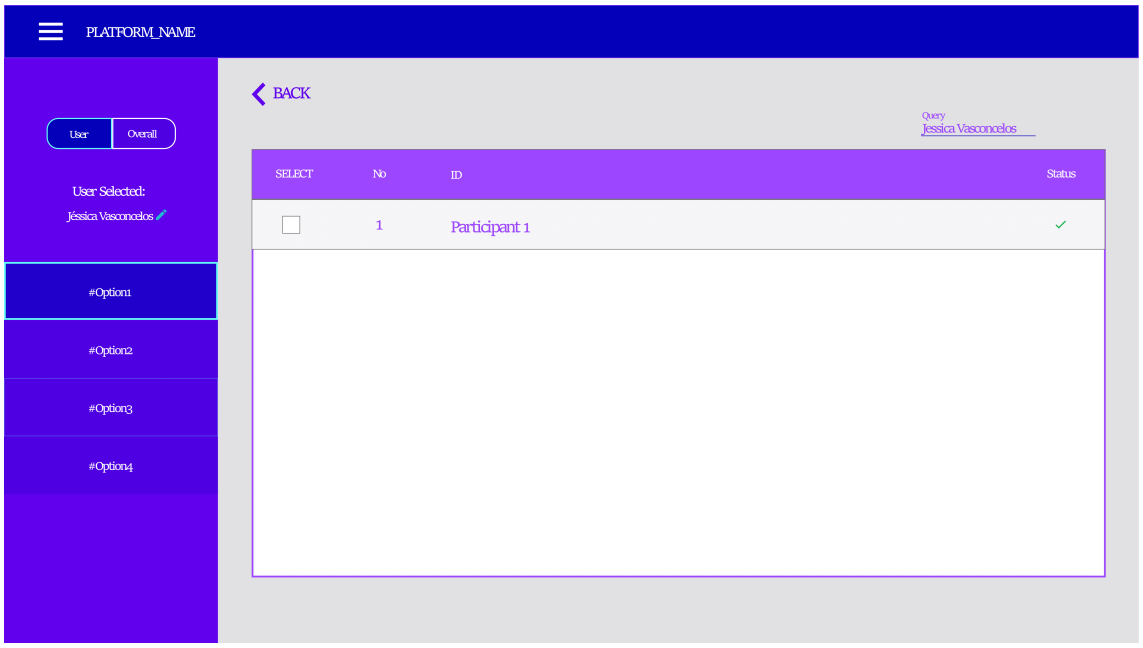
Appendix E

Mock ups

E.1 Version 1.0




PLATFORM_NAME
User
Overall
#Option1
#Option2
#Option3
#Option4
Meters
#OptionX
2125 MAX
#OptionX
189 MIN
#OptionX
3375 AVG
#OptionX
1923 STD
Detailed Information
03/22/2019 - 04/02/2019
Activity registry
Activities Histogram


PLATFORM_NAME
User
Overall
User Selected:
Jéssica Vasconcelos
#Option1
#Option2
#Option3
#Option4
BACK
Query
Jessica Vasconcelos
SELECT
No
ID
Status
1
Participant 1

E.2 Version 2.0