ISEL – Instituto Superior de Engenharia de Lisboa

ADEETC – Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

MEIC
Mestrado em Eng. Informática e de Computadores
Progress Report

CASCIFFO - Software Capacitation in Hospital Professor Doutor Fernando Fonseca Research Center Progress Report

Valdemar Palminha Correia Antunes

Advisors

Prof. Fernando Miguel S.L.C. Gamboa

Prof. Pedro Manuel A.C. Vieira

9 of November, 2022

Abstract

This document presents a thesis made on a joint-project named CASCIFFO, between the Lisbon Superior Institute of Engineering - Instituto Superior de Engenharia Lisboa (ISEL), and the Hospital Professor Doutor Fernando Fonseca (HFF). This project was made possible through the funding earned from the Clinical Investigation Agency Award and Biomedical Innovation (AICIB). This thesis aims to develop a platform and provide innovative mechanisms for interoperability with internal or external information systems, allowing, when desired, data synchronization, index search, identification data management and even access to detailed clinical data, the ability to manage and monitor clinical trials as well as their participants within the Clinical Research Center. The main contribution of this thesis will the optimization and simplicity in managing clinical trials in order to facilitate the researchers efforts in the management of clinical trials. We believe this platform to be an important step in the modernization of the HFF and UIC, bringing change in how the researchers view their institute and its importance. This is report aims to specify the functional requisites of in the platform CASCIFFO and situate the current status of development within the expected timeline.

Keywords: Management of Clinical Trials, Clinical Research, UIC, HFF

Index

List of Figures

Chapter 1

Introduction

This chapter reviews the introduction of HFF and its Clinical Research Unit and its mission. It also introduces the platform being development in the scope of this thesis, the motive behind it and the main goals to be achieved. The end of chapter describes the structure of the document.

1.1 Context and Motivation

This thesis is a part of the joint-project CASCIFFO and it's made within the scope of the curricular unit Tese Final de Mestrado (TFM), in the course Mestrado em Engenharia Informática e de Computadores. A strong motivator behind this project is the impact it'll have within the Clinical Research Unit, by facilitating the management and monitoring of clinical trials, CASCIFFO aims to alleviate this burden off the workload of researchers.

1.2 Hospital Professor Doutor Fernando Fonseca

The Hospital Professor Doutor Fernando Fonseca (HFF), first opened in 1995, is a first of the line hospital combining the professionals excelency with the most modern medical practices, searching to answer to a population of over 600.000 inhabitants of the municipalities of Amadora and Sintra [4]. The Institution develops assistance and research activities as well as providing education, pre- and post-graduation training.

1.2.1 HFF's mission

The Hospital's mission is to provide humanized and differentiated health care throughout a person's life cycle, in collaboration with primary and continuing health care, as well as other hospitals in the National Health Service's ("Serviço Nacional de Saúde") network [12].

1.2.2 Clinicial Research Unit

The Clinical Research Unit - Unidade de Investigação Clínica (UIC), created in March 2018, is an internal department within the Hospital that incorporates fundamental concepts of activities in line with the strategic objectives of the institution. The UIC is responsible for managing clinical trials and is characterized by a multidisciplinary team responsible for ensuring accuracy in the scientific planning of the submitted studies, for the fulfillment of clinical best practices by researchers and for the negotiation of contracts for projects financed by external promoters.

1.2.3 UIC's mission

UIC's mission is to promote quality clinical trials in an organized and sustainable manner. It achieves this by following guidelines that value systematic knowledge through the management of interfaces associated with Research, Development, and Innovation. In addition, it also complies with applicable ethical and legal provisions, for the benefit of the Hospital, the Community, the Patients, and the Families/-Caregivers. The goal is to become a reference in the promotion of best practices in hospital clinical research and to consolidate a transversal scientific culture within the institution [12].

1.2.4 Clinical Research

Clinical research is a very important sector to the world of medicine and health care. It's through it that new medicine and new ways of treatment are discovered and tested through strict adherence to scientific accuracy measurements and best practices before being administrated to the general public.

Types of Clinical Research

There are two main types of clinical investigations, without intervention, which includes observational trials ("Estudos Observacionais"), and with intervention, which includes clinical trials ("Ensaios Clínicos"). The distinction between these trials comes down to the type of intervention between the study and the participants. Active intervention occurs when the researchers in a trial introduce any variable, such as a new medicine, that provokes any sort of change within the participant's behavior, health care or mindset. No intervention means that the team of researchers will not intervene in any way with the participants besides only monitoring them. Having stated these differences, we can clearly define observational trials and clinical trials. Observational Trials have no active intervention, they instead contemplate purely the aspect of observation, for example in the evaluation of a potential risk factor. On the other hand, Clinical Trials engage in active intervention, as such they can be characterized as a Clinical Research which involves any intervention that foresees any change, influence or programming of health care, behavior or knowledge of the participating patients or caretakers, with the end goal of discovering the effects it had on the participant's health. Clinical trials consist of a scientific controlled investigation, done on humans (healthy or ill), with the end-goal of establishing or confirming the safety and efficiency of the experimental medicine [13].

Clinical trials have shown to be a vital tool in the development and testing of vaccinations and treatments for the safety of the entire world population, especially now during the present *Novel Coronavirus (SARS-CoV-2)* pandemic. For this reason, the efficiency and simplicity in managing and monitoring the evolution of a clinical trial is essential.

1.3 Project overview and main goals

CASCIFFO is a joint project between HFF and ISEL and was developed through funding by the Clinical Investigation Agency Award and Biomedical Innovation (AICIB). CASCIFFO is a web-app that aligns with the UIC's goals by promoting efficiency and quality in the management of clinical research. CASCIFFO strives to make the visualization, monitoring, and management of clinical Trials as simple and straightforward as possible. It will allow the UIC/HFF to be modernized, bringing a shift in how patients, researchers, and promoters view and value their institution.

1.3.1 Challenges

The current procedure of a clinical investigation relies on e-mail exchanges between the external and internal parties involved, which adds a considerable amount of effort in the management and monitoring of clinical trials. Another concern is with the scheduling and monitoring of Clinical Trials patients, as there is currently no systematic way of distinguishing the types of appointments made for each patient. In this context, the application CASCIFFO, aims to provide a solution in order to enhance the efficiency in the management and monitoring of clinical research.

1.3.2 Main Goals

The application aims to develop and provide innovative mechanisms for interoperability with internal and external information systems, allowing, when desired, data synchronization, index search, identification data management and even access to detailed clinical data. CASCIFFO consists of two core modules, the front-end and the back-end. The front-end supports interaction with users while the back-end connects to an internal database system to the HFF/UIC and which aggregates the total information of this ecosystem. The interaction with users will depend on their role within the platform, displaying the appropriate information to each one.

1.4 Document Structure

This document is divided into three chapters. The first chapter 1 consists of the introduction of the HFF and UIC, their missions and goals, followed with an overview of the project and the goals to achieve. The second chapter 2 consists of the intricacies of CASCIFFO, the infrastructure and the functional requirements. The third chapter 3 describes the implementation details of both modules that make up the platform. Lastly the fourth chapter 4 consists of the conclusion and analysis of the current state of the thesis.

Chapter 2

State of the art

This chapter consists of a detailed view over the concept and functional requirements of the CASCIFFO platform. It is structured with the following sections:

- Related work: Analysis of platforms with similar functionalities.
- Infrastructure: Description of technologies and framework used for the development of CASCIFFO.
- Access control: Identification and categorization of actors and their roles.
- Processes: Identification and detailing of the process flow.
- Functional Requirements: Description of functional requirements.

2.1 Related work

Clinical trial management is an important factor to consider for any health care organization, to be able to track and manage any data concerning clinical trials and studies, from the moment of the proposal, to the clinical trial itself, the patient recruitment and monitoring management. To this effect, after an brief search and analysis of platforms providing these features, two platforms stood out: the Clinical Conductor Clinical Trial Management System (CTMS) [8] by Advarra [1] and RealTime-CTMS. Clinical Conductor CTMS is a premium service scalable to optimize finances, regulatory compliance, and overall clinical research operations such as financial, patient and visit management including patient recruitment. The platform RealTime-CTMS is a leader in cloud-based software solutions for the clinical research industry and is dedicated to solving problems and providing systems that make the research process more efficient and more profitable [9]. CASCIFFO takes a tailored approach in building a straightforward platform that fits the needs of the Hospital Professor Doutor Fernando Fonseca. It features, in similarity to the aforementioned platforms, clinical trial management, from the moment its a proposal to the end of the clinical trial, allowing the entire progress to be tracked and analyzed; time line management in the aspect of tracking progress and reporting the completion date as well as possible overdue target dates; patient monitoring and visit management; financial management of clinical trials and each individual member of the investigation team and a role-based system for users of different internal departments. Furthermore the way CASCIFFO is planned to be developed, will allow for offline use. It is not as robust as a leading cloud-based software solution like RealTime-CTMS, however, it accomplishes its purpose of bringing innovation and simplicity to the management of clinical trials of the Hospital Professor Doutor Fernando Fonseca's Clinical Research Unit (Unidade de Investigação Clínica).

2.2 Infrastructure

The infrastructure of CASCIFFO, as mentioned previously, consists of two core modules, the front-end and the back-end. The front-end runs on a Node.js environment, using React, a JavaScript library for building user interfaces [7], with Typescript to build all front-end functionalities. The dependencies are managed and installed using npm, a software package manager, installer and the worlds largest software library [2]. npm was chosen over yarn [17] due to its larger community and support. The back-end executes on a run-time Java environment, utilizing Spring-boot and Spring-WebFlux as the basis for building the server. Spring-boot facilitates the building and deployment of web applications by removing much of the boilerplate code and configuration associated with web development [10]. Spring-WebFlux, despite being a new technology, was chosen for its non-blocking web stack framework [15]. This technology allows for the use of Spring Data R2DBC [11], a driver that overcomes the inherent blocking limitation of drivers such as JDBC when accessing data in a database. The database connections via R2DBC are non-blocking, returning streams of data in the form of Flux and Mono upon access. CASCIFFO has its own database, utilizing the framework PostgreSQL, a powerful open-source object-relation database system. Postgres was chosen for its earned reputation in its proven architecture, reliability, data integrity and community support [16]. While CASCIFFO has its own databse, the patient and medical staff data will be imported from an internal database, "Admission", within the HFF/UIC. There are restrictions on the amount of queries made on the medical staff information, limiting this procedure to once per day.

2.3 Access control

Within the app CASCIFFO, in order for the management of clinical investigations to progress, it needs to be reviewed by many entities, such as the Administrative Council ("Concelho administrativo", CA), the Finance and Juridical department. Given this nature of CASCIFFO, there needs to be a well-defined structure of access control, so that each entity can contribute to the management of clinical investigations within the scope of their responsibilities. Each involved entity must have a role and a set of permissions. The roles identified are as follows: the UIC role, given to the investigators who can create and edit clinical investigation proposals; the Team Member role, given to investigators belonging to the team conducting the clinical investigation; the Management role, able to approve and reject clinical investigation proposals; the Finance and Juridical roles, given to collaborators who's function belongs within the Finance and Juridical departments, respectfully; and finally the Superuser role, who has complete access to every feature CASCIFFO has to offer. Each user of CASCIFFO can only have one role.

2.4 Processes

This section details the types of processes occurring within the scope of the project. There are three identified processes consisting of the life-cycle of a clinical investigation proposal, the Clinical Trials and the contract addenda.

2.4.1 Clinical Investigation Proposals

From the instant a clinical investigation is kicked-off, it follows through a series of states and protocols that must be adhered to, in order to be completely validated. There are two types of clinical investigations: Clinical Trials and Observational Trials. Each state, except the terminal one, has an entity responsible, ‘owner‘, for advancing the state. The flow of states is as follows:

1. Submitted (”Submetido”), ‘owner=UIC‘;
2. Financial Contract Validation (”Validação do CF”), ‘owner=Finance, Juridical‘;
3. External validation (”Validação externa”), ‘owner=UIC‘;
4. Submission to the CA (”Submissão ao CA”), ‘owner=UIC‘;
5. Internal validation (”Validação interna”), ‘owner=CA‘;
6. Validated (”Validado”).

The enumerated set of states corresponds to the life-cycle of a clinical trial Proposal. An Observational Trial Proposal consists of the enumerated states 1, 4, 5 and 6; it lacks a financial component and a promoter.

Taking the example of the submission of a clinical trial Proposal, an investigator starts by creating and submitting a proposal. Once it's submitted, the CA will be notified, via app and email. This state is described as *Submitted.* When the negotiation of the financial contract begins, the principal Investigator, who belongs to the UIC role, will advance the state to its next step in the proposal's evolution, *Validation of financial contract*, where users with roles of either ‘Finance‘ and/or ‘Juridical‘, which represents the Financial and Juridical internal departments, respectfully, will be notified that a proposal is ready to be validated. The validation will consist of a simple ‘Accept‘ or ‘Refuse‘ with added justification for the choice. In the case of either user with ‘Finance‘ or ‘Juridical‘ role reject the financial contract, both the validations will be reset and the principal investigator will be notified of the occurrence. Once it's accepted by both roles, the proposal will automatically advance into the next state, *External validation.* In this state the UIC will be notified of the change and asked to verify all the documents, including the final version of the financial contract. The UIC to the external promoter, requesting their signatures. When the reply is received via email, the UIC adds the received signatures and possible additional documents to the proposal in the CASCIFFO platform, advancing it to the next stage *Submission to CA*. In the state *Submission to CA*, the principal investigator will be notified both via the platform and email that a proposal requires their signature. Once the principal investigator submits their signature into the platform, he can advance the proposal's state into *Internal Validation.* The progression to the mentioned state will notify users with the 'CA' role stating that a proposal is ready for its final evaluation. Once a user with the role of 'CA' checks the proposal he can either validate it or not. In case it's not validated, the proposal will become 'canceled' with its life-cycle ending there, however, if it is validated, the proposal can become fully validated once the termination of another process is ends successfully. This process, which can be considered a sub-process, is called the validation protocol. It starts in parallel when the proposal is first submitted. The purpose of this protocol is to validate the clinical investigation's ethical and safety values. It consists in the validation of the proposal by an internal agency, the clinical investigations Ethics Comity ("Comissão de Ética para Investigação Clínica", CEIC [3]). The protocol ends when the mentioned agency approves or rejects the proposal. Once it has successfully passed through the described validation protocol, the proposal becomes *Validated* and a Clinical or Observational Trial is automatically created, importing the core information from the proposal. If either process declares the proposal invalid, its state becomes 'canceled', notifying the UIC and showing the root cause of cancellation.

Each proposal is distinguished by six main properties, the principal investigator, the type of investigation, the type of therapeutic service it's integrated into (*i.e.* Oncology), the 'Sigla' which represents the name of the therapeutic or medicine, the partnerships involved in the investigation and the medical team participating in the investigation. Proposals with a financial component must also include the promoter of the investigation, in addition to the properties listed.

2.4.2 Clinical Trials

The life-cycle of a clinical trial is divided into three states: active, completed, and canceled. Starting with the active state, a clinical trial will become available for viewing and editing once its proposal has been accepted. Clinical trials, as a process, consist on the experimentation of new medicine or treatment on a set of participants. These participants can be added to the clinical trial directly from the application. The experimentation requires constant monitoring, through visits, on each participant. These visits can be scheduled either when a participant is added or created afterwards. In addition to monitoring participants, several studies can be made in the scope of the clinical trial, such as scientific articles, presentations, reports, etc.

2.4.3 Addenda to the contract

Throughout the life of a clinical trial, there can be made changes to the study's contract, be it changing the investigator team or other factors that impact the standard run of the study. These changes pass through two entities before being applied; the UIC and the CA. The addenda can only be made once a clinical trial is active, which means its proposal has already been approved. The addenda has five different states: submitted 'Submetido', internal validation by UIC 'Validação interna', internal validation by CA 'Validação interna', the terminal state validated 'Validado' and finally the terminal canceled state 'Indeferido'. The first four mentioned states are sequential, with the last one being an exception state. The sequential flow of states have an entity responsible for advancing their state, the 'owner'. Listed below, in similar fashion to the states presented in the proposal process, is the aforementioned sequence:

1. Submitted ("Submetido"), 'owner=UIC';
2. Internal validation ("Validação interna"), 'owner=UIC';
3. Internal validation ("Validação interna"), 'owner=CA';
4. Validated ("Validado").

2.5 Functional Requirements

This section details the functional requirements and presents a mock user interface (UI) that will satisfy the requirement. The main features of CASCIFFO can be separated into three groups, which are: general functionalities, clinical component and financial component.

1. General features
   - Visualization and management of Clinical Trials as a process;
   - Ability to edit and validate data (edit checks);
   - Access control based on different user profiles;
   - Access by computer, tablet or smartphone;
   - Ability to export information in numerical or graphical mode;
   - Ability to customize the form of visualization.
2. Clinical Component
   - View detailed characteristics and evolution of clinical trials including the tested medicine or technique in question;

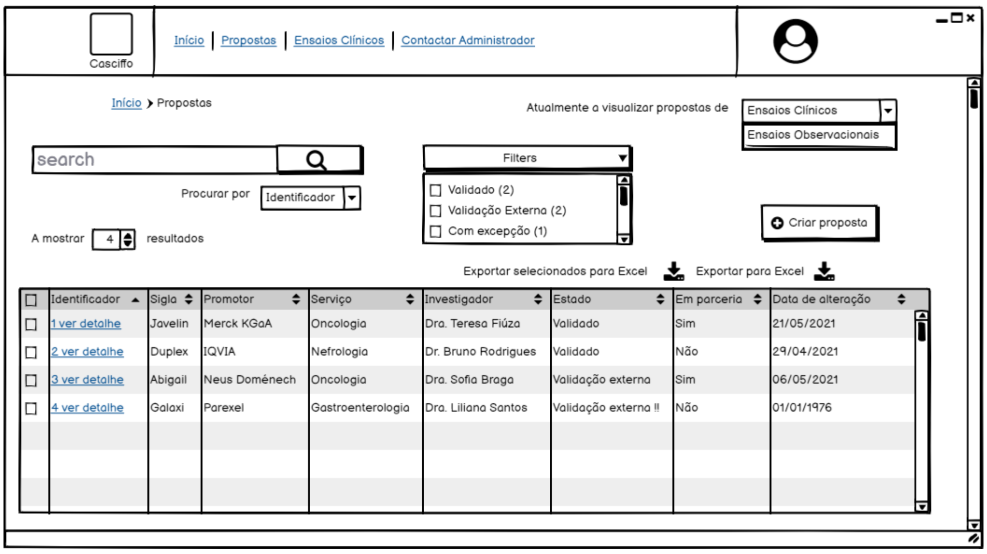
- Monitoring the set of patients included in clinical trials and their characteristics;
- Insertion of patient data in face-to-face or tele-consultation;
- Characteristics of the treatment associated with the clinical trial;
- Monitoring of the patient's behavior under trial and its attendance;
- Monitoring of physical and financial assets;
- Monitoring of visits & recording of adverse events.

2.6 General Features

In this section, the general features mentioned in the document will be described and illustrated through mock-ups.

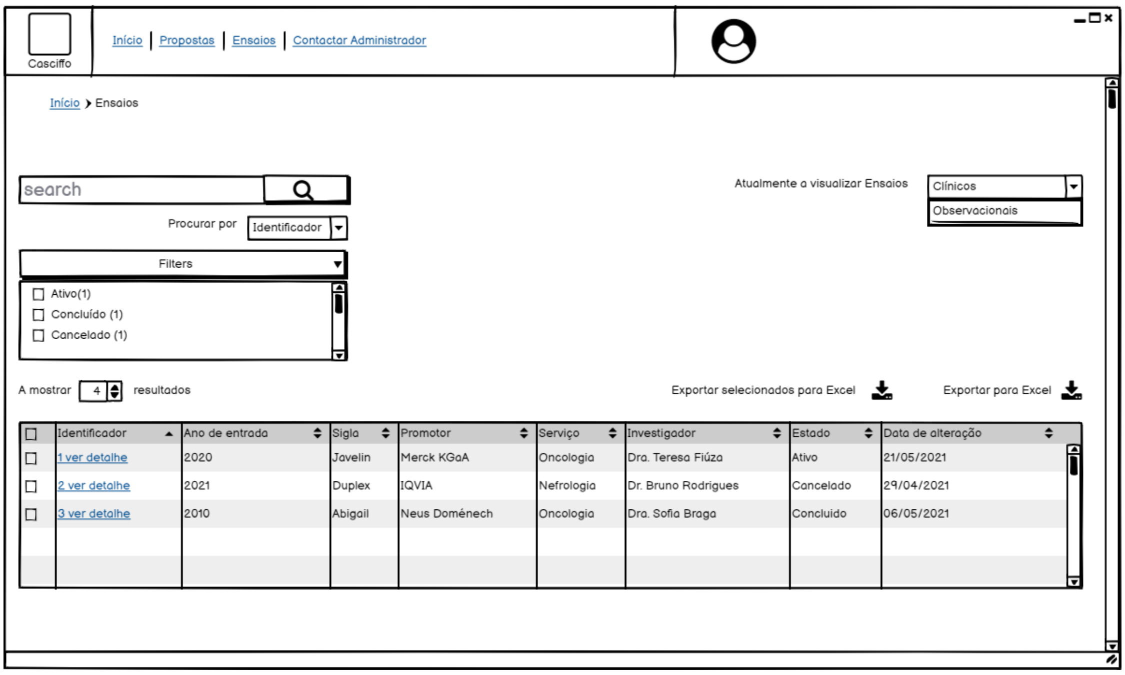
2.6.1 Visualization and Management of Clinical Trials as a Process

To view and manage a clinical trial as a process, a user needs only to view the general overview of Clinical Trials or clinical investigations Proposals. As in figure 2.1 and figure 2.2, the user has an overview of the all submitted proposals and clinical trials with their main characteristics, such as, the identification, current state, last alteration date, the principal investigator and whether it has partnerships or not.



☐	Identificador	Sigla	Promotor	Serviço	Investigador	Estado	Em parceria	Data de alteração
☐	1 ver detalhe	Javelin	Merck KGaA	Oncologia	Dra. Teresa Fiúza	Validado	Sim	21/05/2021
☐	2 ver detalhe	Duplex	IQVIA	Nefrologia	Dr. Bruno Rodrigues	Validado	Não	29/04/2021
☐	3 ver detalhe	Abigail	Neus Doménech	Oncologia	Dra. Sofia Braga	Validação externa	Sim	06/05/2021
☐	4 ver detalhe	Galaxi	Parexel	Gastroenterologia	Dra. Liliana Santos	Validação externa !!	Não	01/01/1976

Figure 2.1: Mock overview of clinical investigations proposals.


Casciffo
Início | Propostas | Ensaios | Contactar Administrador
Início > Ensaios
search
Procurar por Identificador
Filters
Ativo(1)
Concluído (1)
Cancelado (1)
Atualmente a visualizar Ensaios
Clínicos
Observacionais
A mostrar 4 resultados
Exportar selecionados para Excel
Exportar para Excel

	Identificador	Ano de entrada	Sigla	Promotor	Serviço	Investigador	Estado	Data de alteração
☐	1 ver detalhe	2020	Javelin	Merck KGaA	Oncologia	Dra. Teresa Fiúza	Ativo	21/05/2021
☐	2 ver detalhe	2021	Duplex	IQVIA	Nefrologia	Dr. Bruno Rodrigues	Cancelado	29/04/2021
☐	3 ver detalhe	2010	Abigail	Neus Doménech	Oncologia	Dra. Sofia Braga	Concluido	06/05/2021

Figure 2.2: Mock overview of clinical trials.

When a user clicks the link view details (”ver detalhe”), he will be redirected to a screen displaying the details of the target clinical investigation.

2.6.2 Ability to edit and validate data (edit checks)

Within the CASCIFFO platform, the UIC and internal departments can view the details of a certain clinical investigation proposal and edit or validate according to their roles. Users with ‘UIC‘ role will be able to create and edit their own Investigations, whereas the internal departments, with roles of ‘CA‘, ‘Finance‘ and ‘Juridical‘ will only be able to validate the proposals. The creation of a proposal starts in the overview of proposals screen, from there a user can click on Create Proposal (”Criar proposta”) and will be redirected to the screen illustrated in figure 2.3. Here an investigator can choose what type of investigation this proposal corresponds to, either an observational or clinical trial. In the case a clinical trial is selected, more options will be shown since clinical trials have a corresponding financial component. The data fields corresponding to the therapeutic area, the service and the pathology of an investigation are restricted to a set of possible inputs. In addition, the members of a medical team also belong to an already defined database, being validated at the time of creation of the medical team for the investigation. Certain fields cannot be changed once a proposal has been submitted, such as the promoter, the partnerships and the type of investigation can only be defined during the creation of a proposal.

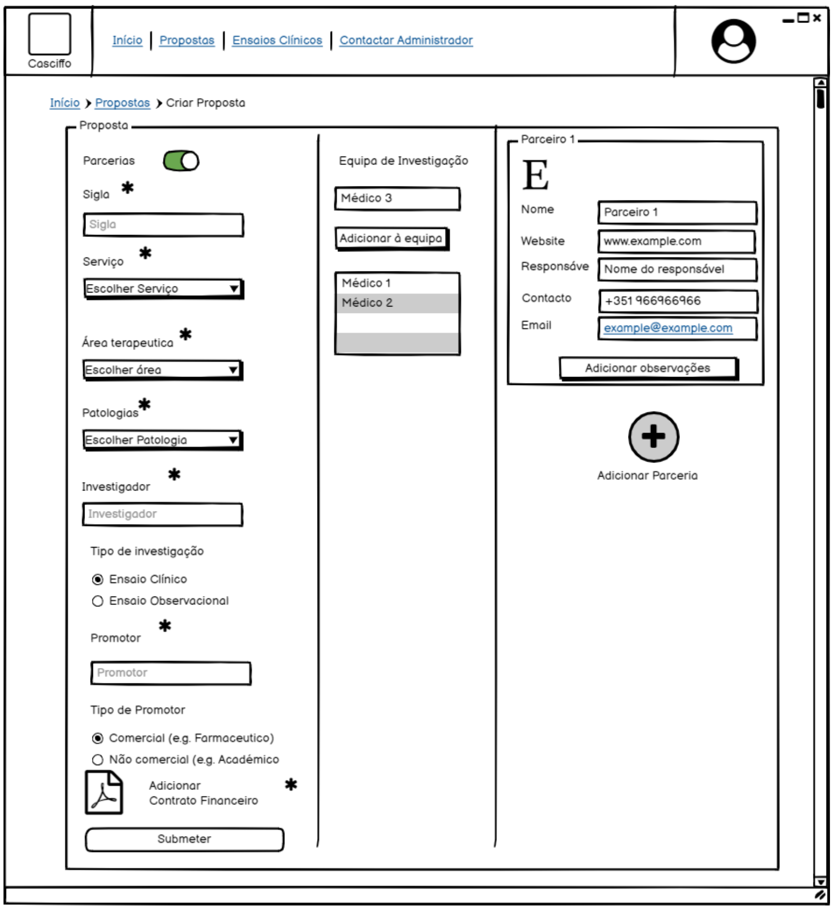


Figure 2.3: Mock creation of a clinical investigation proposal.

2.6.3 Access control based on different user profiles

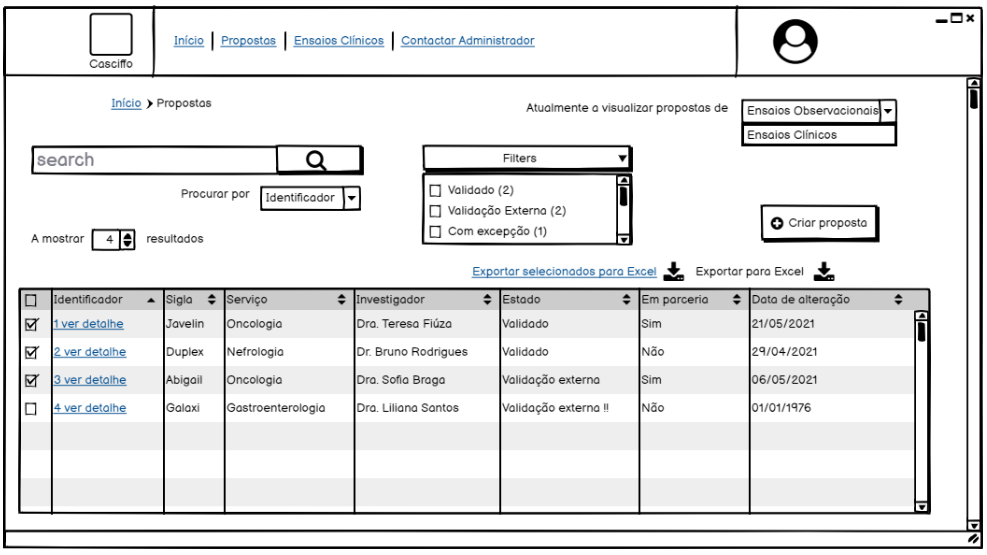
The access control within the CASCIFFO application is based on roles. As described in section 1, there are defined roles for each type of user. A user with the role 'UIC' can manage their own clinical investigation, being able to edit and have a hand in advancing the state. In addition, it also has an overview over all ongoing investigations, not being able to edit those that weren't created by said user. Once this user logs in, a dashboard showing overall statistics of the platform, *i.e.* number of active clinical trials, number of submitted proposals, etc. The role 'Financial' is responsible for validating and updating the financial components of the investigations, hence when a user with this role logs in, an appropriate financial management screen will be displayed, whereas the 'Juridical' component validates the juridical component. The role 'CA' is responsible for giving the final decision in whether an investigation proposal can have the go-ahead to begin their clinical trials or observations. A user of role 'CA' once logged in, will be shown a primary screen displaying the overview of clinical investigation proposals awaiting validation. Finally, we have the 'Superuser' role, which besides having the ability to execute every mentioned action, it can also create new types of services, therapeutic areas and pathologies.

2.6.4 Access by computer, tablet or smartphone

CASCIFFO is a web-application which can be accessed via any device or browser. CASCIFFO offers an extended functionality to browsers that support service workers, since it utilizes the Progressive Web Application (PWA) framework, allowing it to be installed and used as an application. Among the features a PWA allows, the framework was chosen by its ability to let an application run in offline-mode and versatility in that it can be installed via the browser, and used in similarity to a standalone mobile app.

2.6.5 Ability to export information in numerical or graphical mode

CASCIFFO offers the ability to export information via excel, and visualize graphic data within the app. There is a feature, shown in figure 2.4 that allows a user to export data, *e.g.* a selected number of clinical investigation proposals. This feature is present in the screens showing listed data, *e.g.* list of clinical investigation proposals, and the details of each clinical investigation, proposal or trial.



☐	Identificador	Sigla	Serviço	Investigador	Estado	Em parceria	Data de alteração
☑	1 ver detalhe	Javelin	Oncologia	Dra. Teresa Fiúza	Validado	Sim	21/05/2021
☑	2 ver detalhe	Duplex	Nefrologia	Dr. Bruno Rodrigues	Validado	Não	29/04/2021
☑	3 ver detalhe	Abigail	Oncologia	Dra. Sofia Braga	Validação externa	Sim	06/05/2021
☐	4 ver detalhe	Galaxi	Gastroenterologia	Dra. Liliana Santos	Validação externa !!	Não	01/01/1976

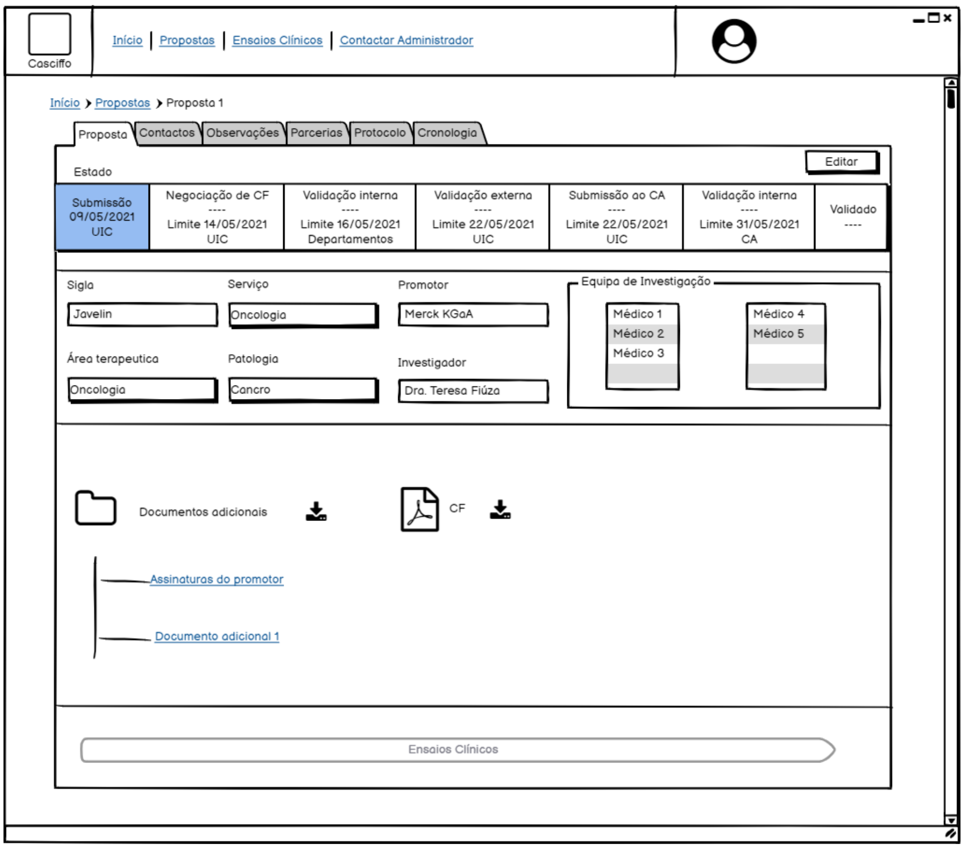
Figure 2.4: Mock screen selecting clinical proposals to export into excel.

2.6.6 Ability to customize the form of visualization

The ability to customize the form of visualization will be available in the initial screen of the app, the Dashboard, where the user will be able to view different types of graphs showing statistics based on the states of clinical investigations.

2.6.7 View detailed Characteristics and evolution of clinical Trials including the tested medicine or technique in question

To view detailed information about a clinical investigation, one needs to first overview the clinical investigations and then click on the details of a desired clinical investigation. Considering this option, the user will be redirected to a screen detailing the study. The evolution of any clinical investigation consists of its proposal followed the trial activity once the proposal has been fully validated. In figure 2.5, the details of a clinical trial proposal can be viewed. The flow of state of a proposal is shown in the form of a bar in a straight forward manner. Each state corresponds to a division, box, in the bar and has three properties: the name of the State; the date it was completed in, if the state has otherwise not been completed, then a sequence of dashes will appear in its place; the deadline at which it should be completed and finally the entity responsible for advancing the state. It is possible to click in the box of a state to display information about the events that occurred during the time the proposal was in that state.



Estado

Submissão 09/05/2021 UIC	Negociação de CF ---- Limite 14/05/2021 UIC	Validação interna ---- Limite 16/05/2021 Departamentos	Validação externa ---- Limite 22/05/2021 UIC	Submissão ao CA ---- Limite 22/05/2021 UIC	Validação interna ---- Limite 31/05/2021 CA	Validado ----

Sigla: Javelin
Serviço: Oncologia
Promotor: Merck KGaA
Área terapeutica: Oncologia
Patologia: Cancro
Investigador: Dra. Teresa Fiúza

Equipa de Investigação: Médico 1, Médico 2, Médico 3 | Médico 4, Médico 5

Assinaturas do promotor
Documento adicional 1

Figure 2.5: Mock overview of a clinical investigation proposal.

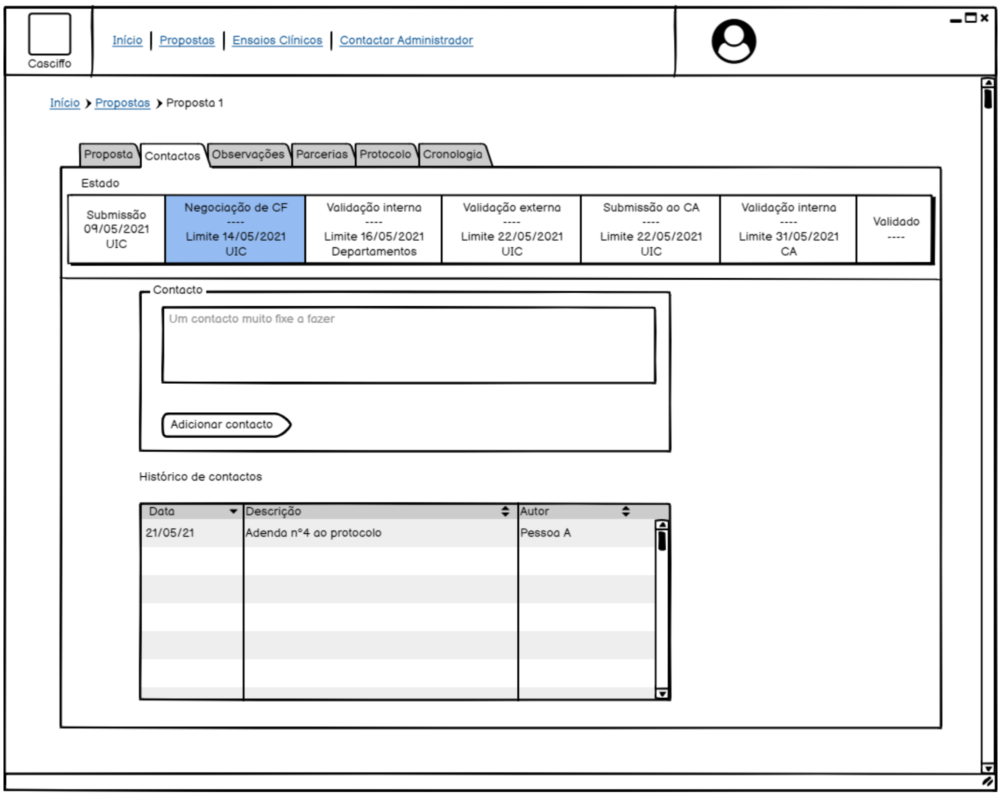
A clinical trial proposal considers five components alongside its principal details displayed as tabs and listed below:

- the Contacts ("Contactos") tab which corresponds to comments related to external communications, viewed in figure 2.6;
- the Observations ("Observações") tab which contains comments made in regard to the proposal, viewed in figure 2.7;
- the Partnerships ("Parcerias") tab, where one can find the partnerships involved in the study proposal, viewed in figure 2.8;

- the validation Protocol ("Protocolo") tab, where it can be viewed the current state of affairs of the process described in section 2.6.1, and in figure 2.9;

- the Chronology ("Cronologia") tab that displays the timeline of events, as in figure 2.10. Events include deadlines introduced by the user and the transition of states. The user will have the ability to specify the scope of the timeline, selecting the time range and type of events to view.

All tabs, except the one pertaining to the Protocol validation, will have present the current state of the proposal.

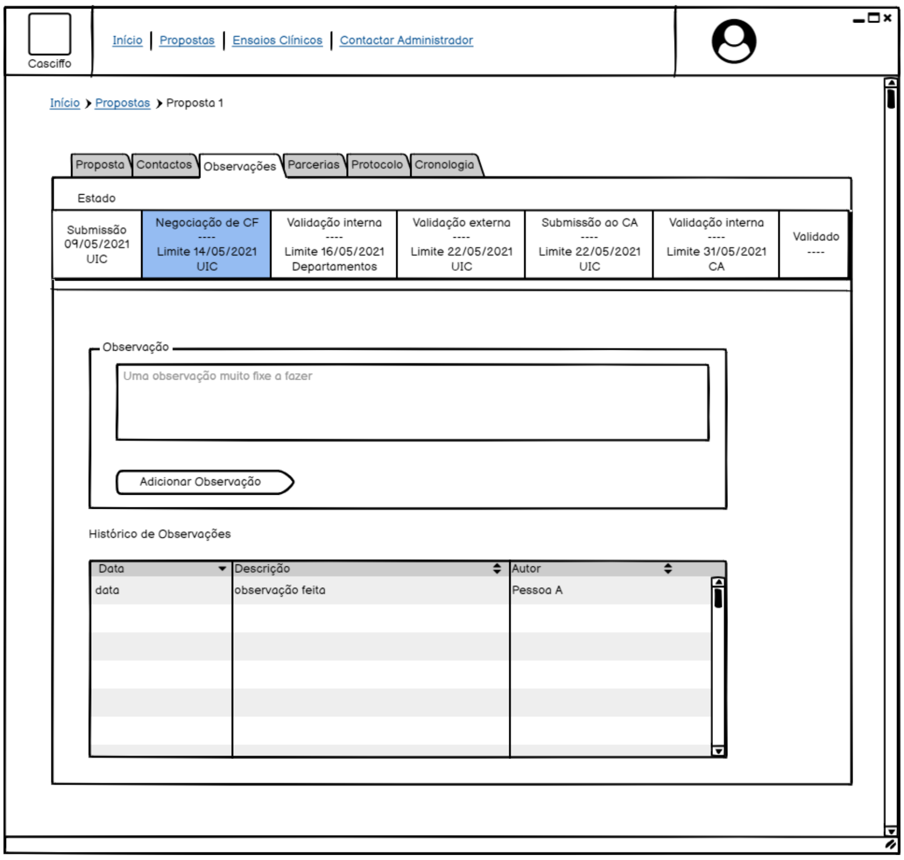


Submissão	Negociação de CF	Validação interna	Validação externa	Submissão ao CA	Validação interna	Validado
09/05/2021	----	----	----	----	----	----
	Limite 14/05/2021	Limite 16/05/2021	Limite 22/05/2021	Limite 22/05/2021	Limite 31/05/2021	
UIC	UIC	Departamentos	UIC	UIC	CA	

Um contacto muito fixe a fazer

Data	Descrição	Autor
21/05/21	Adenda nº4 ao protocolo	Pessoa A

Figure 2.6: Mock overview of the contacts tab.


Casciffo

Início | Propostas | Ensaios Clínicos | Contactar Administrador

Início > Propostas > Proposta 1

Proposta | Contactos | Observações | Parcerias | Protocolo | Cronologia

Estado

Submissão 09/05/2021 UIC	Negociação de CF ---- Limite 14/05/2021 UIC	Validação interna ---- Limite 16/05/2021 Departamentos	Validação externa ---- Limite 22/05/2021 UIC	Submissão ao CA ---- Limite 22/05/2021 UIC	Validação interna ---- Limite 31/05/2021 CA	Validado ----

Observação

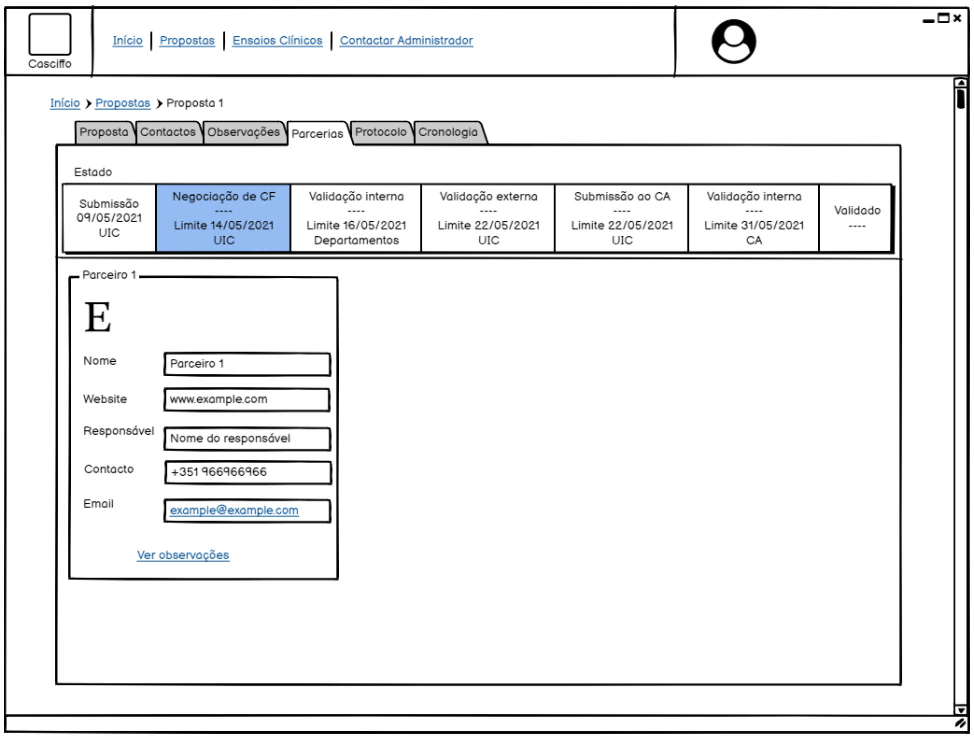
Uma observação muito fixe a fazer

Adicionar Observação

Histórico de Observações

Data	Descrição	Autor
data	observação feita	Pessoa A

Figure 2.7: Mock overview of the observations tab.


Casciffo

Início | Propostas | Ensaios Clínicos | Contactar Administrador

Início > Propostas > Proposta 1

Proposta | Contactos | Observações | Parcerias | Protocolo | Cronologia

Estado

Submissão 09/05/2021 UIC	Negociação de CF ---- Limite 14/05/2021 UIC	Validação interna ---- Limite 16/05/2021 Departamentos	Validação externa ---- Limite 22/05/2021 UIC	Submissão ao CA ---- Limite 22/05/2021 UIC	Validação interna ---- Limite 31/05/2021 CA	Validado ----

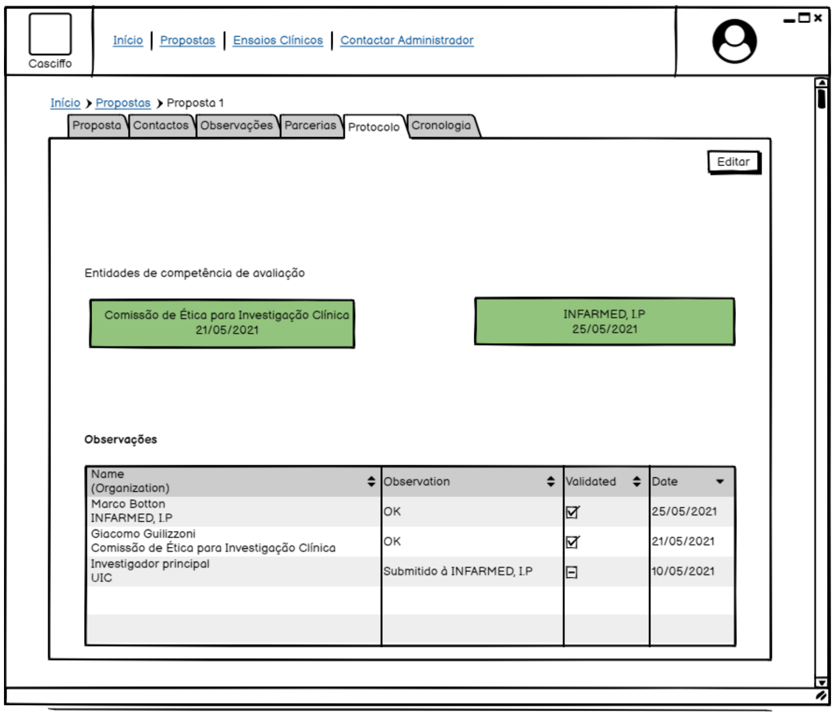
Parceiro 1

E

Nome	Parceiro 1
Website	www.example.com
Responsável	Nome do responsável
Contacto	+351 966966966
Email	example@example.com

Ver observações

Figure 2.8: Mock overview of the partnerships tab.


Casciffo
Início | Propostas | Ensaios Clínicos | Contactar Administrador
Início > Propostas > Proposta 1
Proposta
Contactos
Observações
Parcerias
Protocolo
Cronologia
Editar

Entidades de competência de avaliação

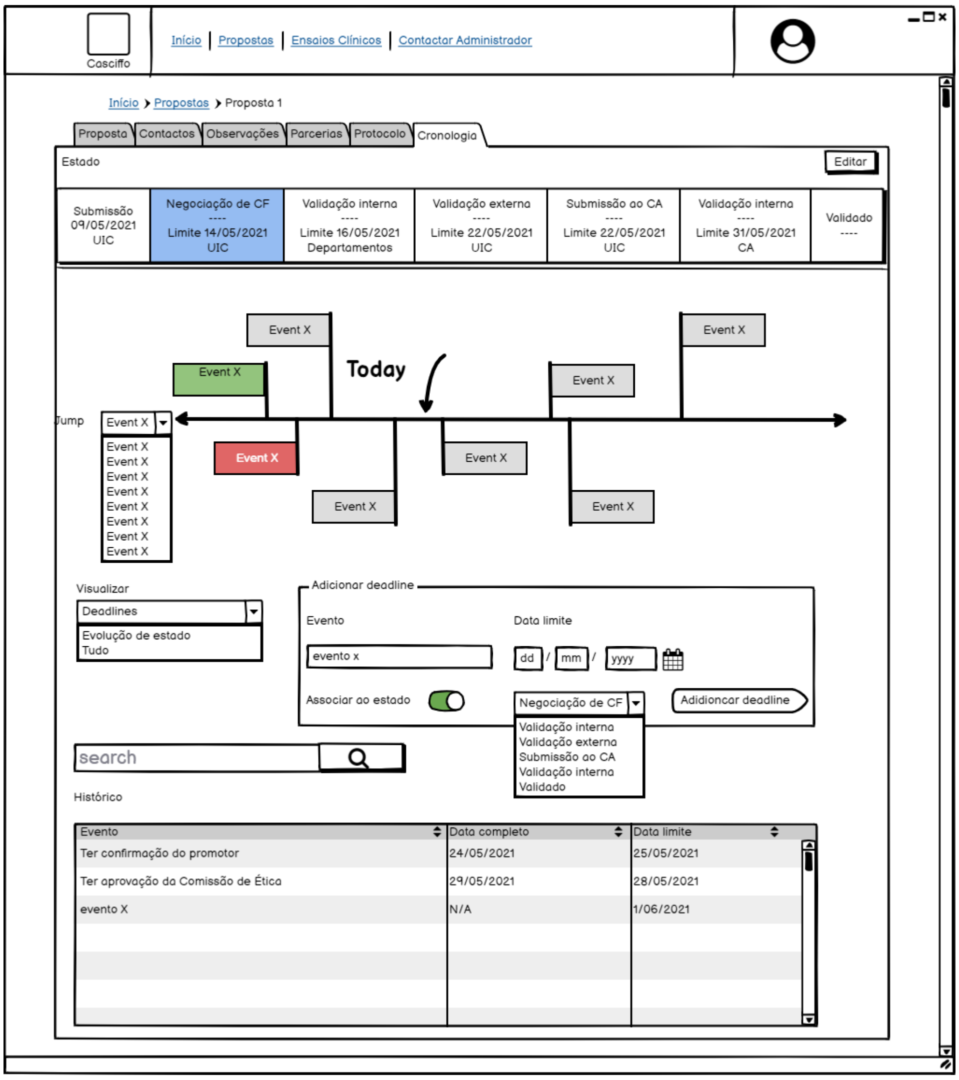
Comissão de Ética para Investigação Clínica
21/05/2021

INFARMED, I.P
25/05/2021

Observações

Name (Organization)	Observation	Validated	Date
Marco Botton INFARMED, I.P	OK	☑	25/05/2021
Giacomo Guilizzoni Comissão de Ética para Investigação Clínica	OK	☑	21/05/2021
Investigador principal UIC	Submitido à INFARMED, I.P	⊟	10/05/2021

Figure 2.9: Mock overview of the protocol tab.



Estado

Submissão 09/05/2021 UIC	Negociação de CF ---- Limite 14/05/2021 UIC	Validação interna ---- Limite 16/05/2021 Departamentos	Validação externa ---- Limite 22/05/2021 UIC	Submissão ao CA ---- Limite 22/05/2021 UIC	Validação interna ---- Limite 31/05/2021 CA	Validado ----

Jump: Event X
Event X
Event X
Event X
Event X
Event X
Event X
Event X
Event X

Visualizar: Deadlines
Evolução de estado
Tudo

Adicionar deadline
Evento: evento x
Data limite: dd / mm / yyyy
Associar ao estado: Negociação de CF
Validação interna
Validação externa
Submissão ao CA
Validação interna
Validado
Adidioncar deadline

Histórico

Evento	Data completo	Data limite
Ter confirmação do promotor	24/05/2021	25/05/2021
Ter aprovação da Comissão de Ética	29/05/2021	28/05/2021
evento X	N/A	1/06/2021

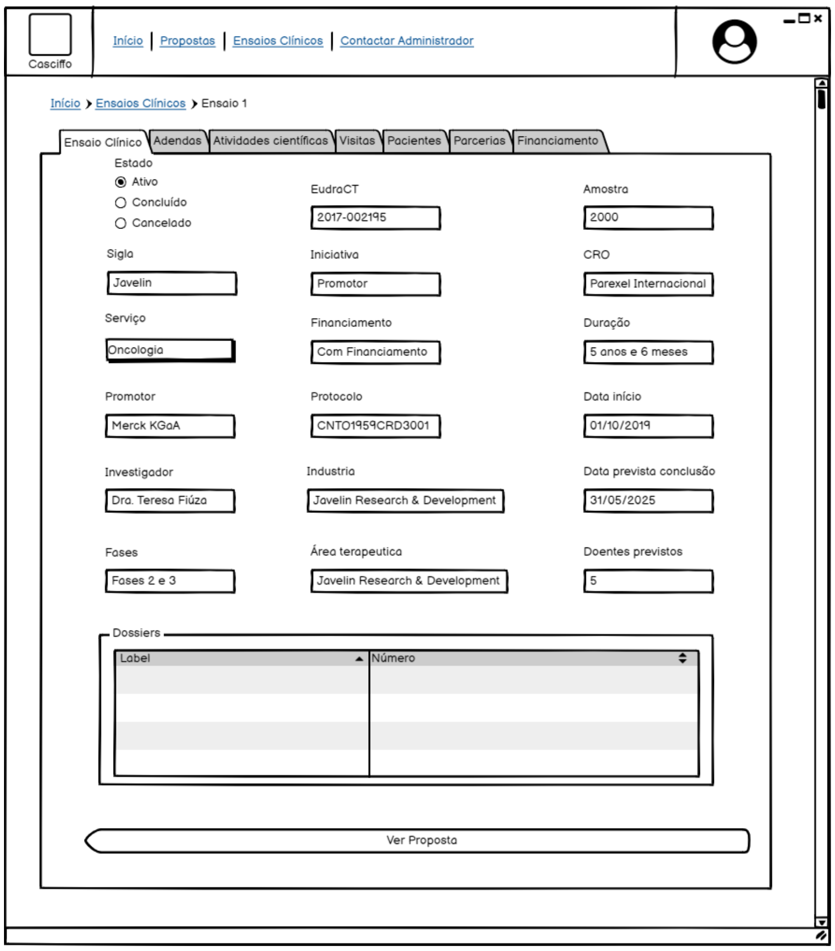
Figure 2.10: Mock overview of the chronology tab.

Once the proposal has reached its successful terminal state, as in section 2.4.1 a clinical trial will be created. To access the newly created trial, the user can either click the button "Ensaio Clínico" from the proposal details screen, as in figure 2.5, or from the overview of clinical trials click on the details of one. In the screen dedicated to the clinical trial and presented in figure 2.11, there are five different tabs:

- the clinical trial tab ("Ensaio Clínico") that displays the characteristics of the study;
- the scientific activities tab ("Atividades científicas"), viewed in figure 2.12, which includes scientific work in the scope of the study, such as articles, thesis, reports, etc.;
- the visits tab ("Visitas") where the investigator team can have an overview of the history and scheduled visits;
- the patients tab ("Pacientes") with the purpose of showing an overview of the participants included in the trial;

- the partnerships tab (”Parcerias”), similar to the proposal’s version of partnerships tab, displays the partnerships involved in the study;

- the financial management (”Financiamento”), where one can view the flow of monetary gain from visits and the partition between the investigator team.



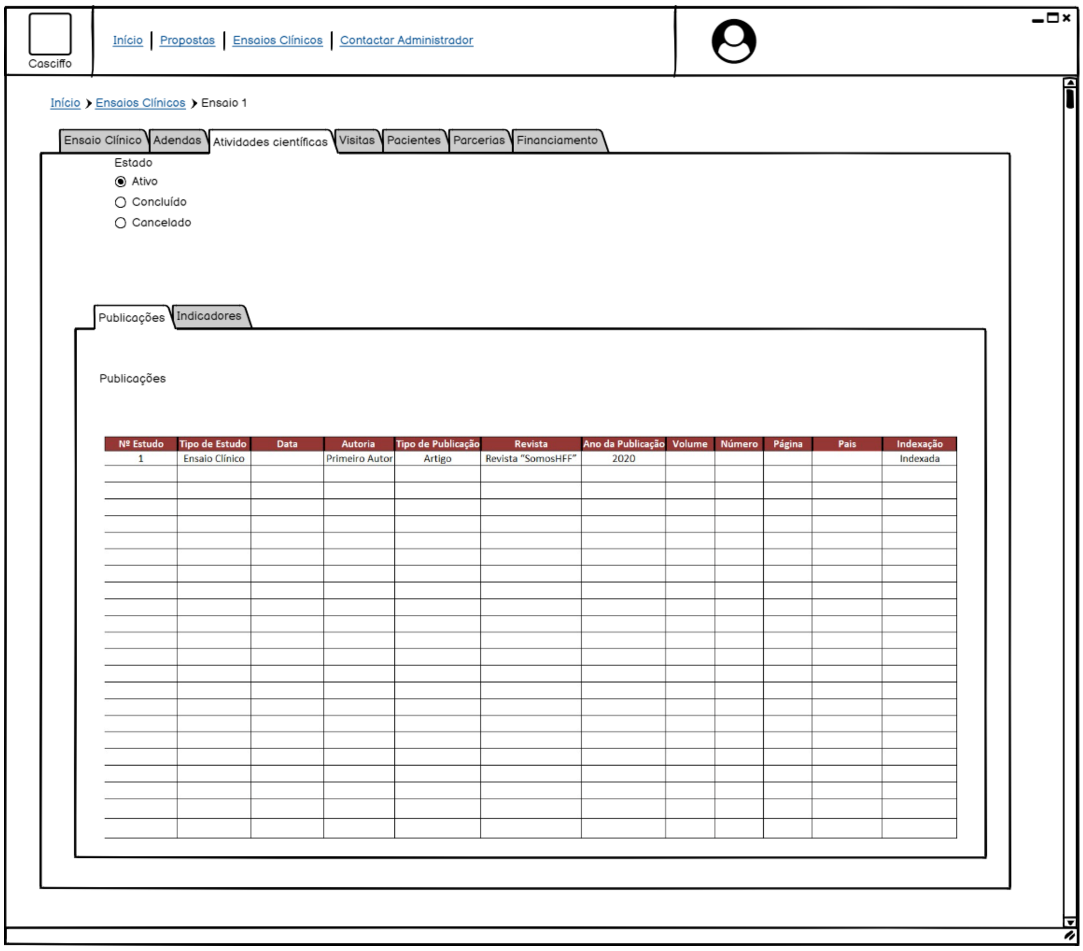
Estado
[✓ Ativo]
Concluído
Cancelado

EudraCT: 2017-002195	Amostra: 2000	
Sigla: Javelin	Iniciativa: Promotor	CRO: Parexel Internacional
Serviço: Oncologia	Financiamento: Com Financiamento	Duração: 5 anos e 6 meses
Promotor: Merck KGaA	Protocolo: CNTO1959CRD3001	Data início: 01/10/2019
Investigador: Dra. Teresa Fiúza	Industria: Javelin Research & Development	Data prevista conclusão: 31/05/2025
Fases: Fases 2 e 3	Área terapeutica: Javelin Research & Development	Doentes previstos: 5

Dossiers

Label	Número

Figure 2.11: Mock overview of the details of a clinical trial.

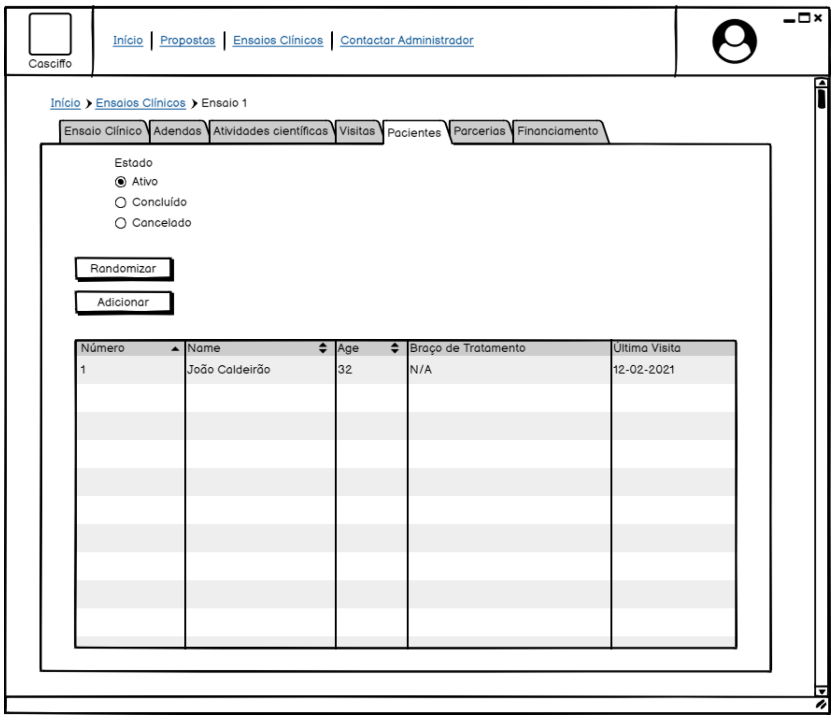


Nº Estudo	Tipo de Estudo	Data	Autoria	Tipo de Publicação	Revista	Ano da Publicação	Volume	Número	Página	Pais	Indexação
1	Ensaio Clínico		Primeiro Autor	Artigo	Revista "SomosHFF"	2020					Indexada

Figure 2.12: Mock overview of scientific activities made within the scope of the investigation.

2.6.8 Monitoring the set of patients included in clinical Trials and their characteristics

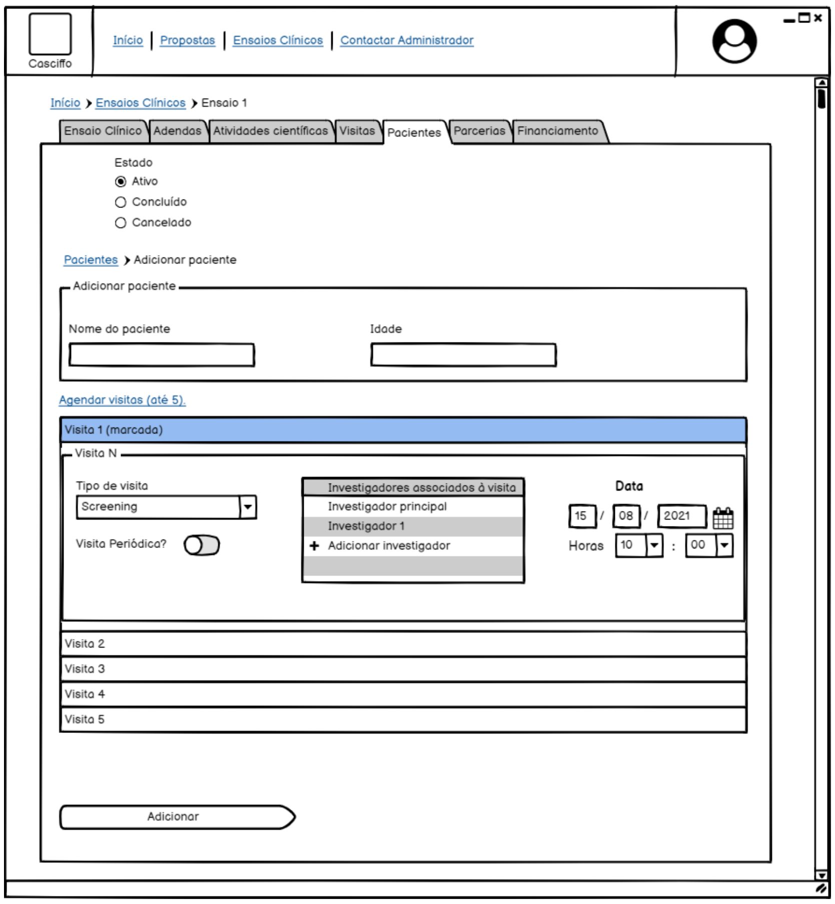
The details of the set of patients involved in a clinical trial will be displayed when viewing the details of said clinical trial, under the patients (”Pacientes”) tab , illustrated in figure 2.13. This tab displays the set of current participants undergoing the clinical trial and information such as the participant number, used to identify the participant throughout the study, the name of the participant, their age, the treatment branch they were assigned to within the scope of the study, their last and the closest upcoming visit. Included in this screen are the buttons Randomize (”Randomizar”) and Add (”Adicionar”). The button ”randomize” will randomly assign participants to treatment branches within the study, this one-time procedure is to be used once all the participants have been added to the clinical trial. The button ”add”, will begin the procedure to add a patient. Upon clicking this button, the user will be shown a small box, as illustrated in figure 2.14, where he can input the name of the participant and then is also given the chance to immediately schedule several visits.



Estado
- [✓ Ativo]
- Concluído
- Cancelado

Número	Name	Age	Braço de Tratamento	Última Visita
1	João Caldeirão	32	N/A	12-02-2021

Figure 2.13: Mock overview of all participants included in the study.



Estado
- [✓ Ativo]
- Concluído
- Cancelado

Adicionar paciente

Nome do paciente:

Idade:

Visita N

Tipo de visita: Screening

Visita Periódica?

Investigadores associados à visita
Investigador principal
Investigador 1

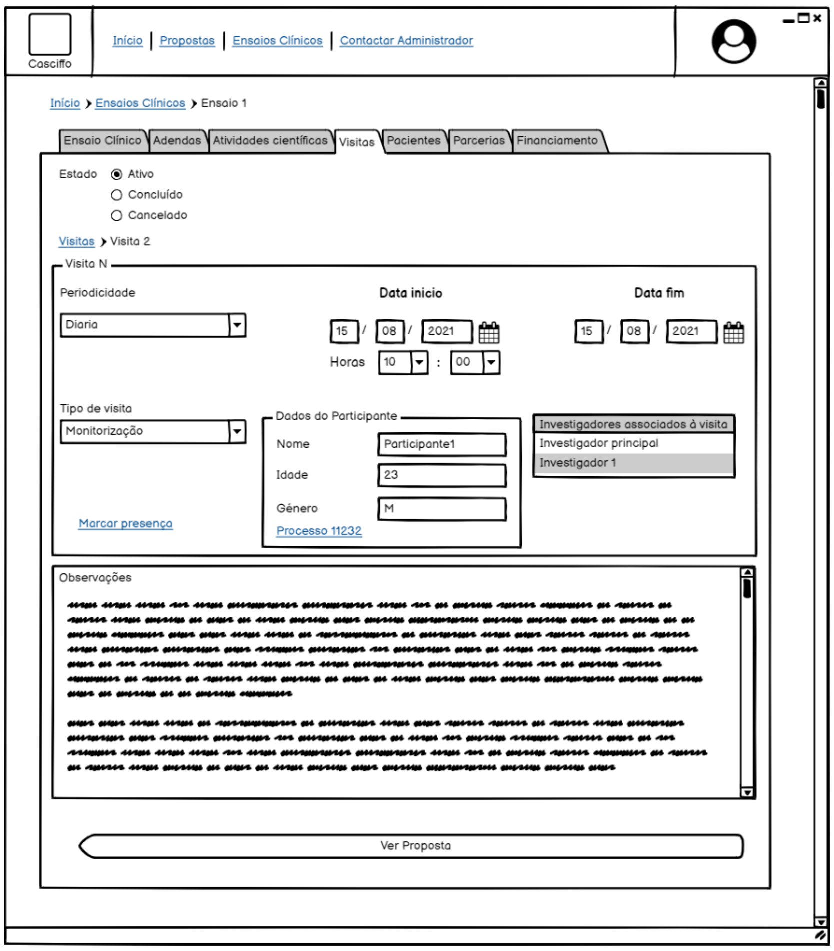
Data: 15 / 08 / 2021

Horas: 10 : 00

Figure 2.14: Mock screen of adding a new patient as a participant in the study.

2.6.9 Insertion of patient data in face-to-face or tele-consultation

The insertion of patient data in the context of a visit is made by the investigator associated to the mentioned visit. In order to do this, the investigator has to navigate to the details of the visit, from the overview of clinical trials, to the details of the trial followed by the overview of visits and finally the details of the considered visit. Here the investigator can input observational data into the field "Observações" which will represent the observations made throughout the visit or tele-consultation. He is also asked to mark the attendance of the participant by clicking on a simple button "Marcar presença", as illustrated in figure 2.15.



Estado [✓ Ativo] Concluído Cancelado

Periodicidade: Diaria

Data inicio: 15 / 08 / 2021 Horas 10 : 00

Data fim: 15 / 08 / 2021

Tipo de visita: Monitorização

Dados do Participante

Nome	Participante1
Idade	23
Género	M

Processo 11232

Investigadores associados à visita
Investigador principal
Investigador 1

Observações

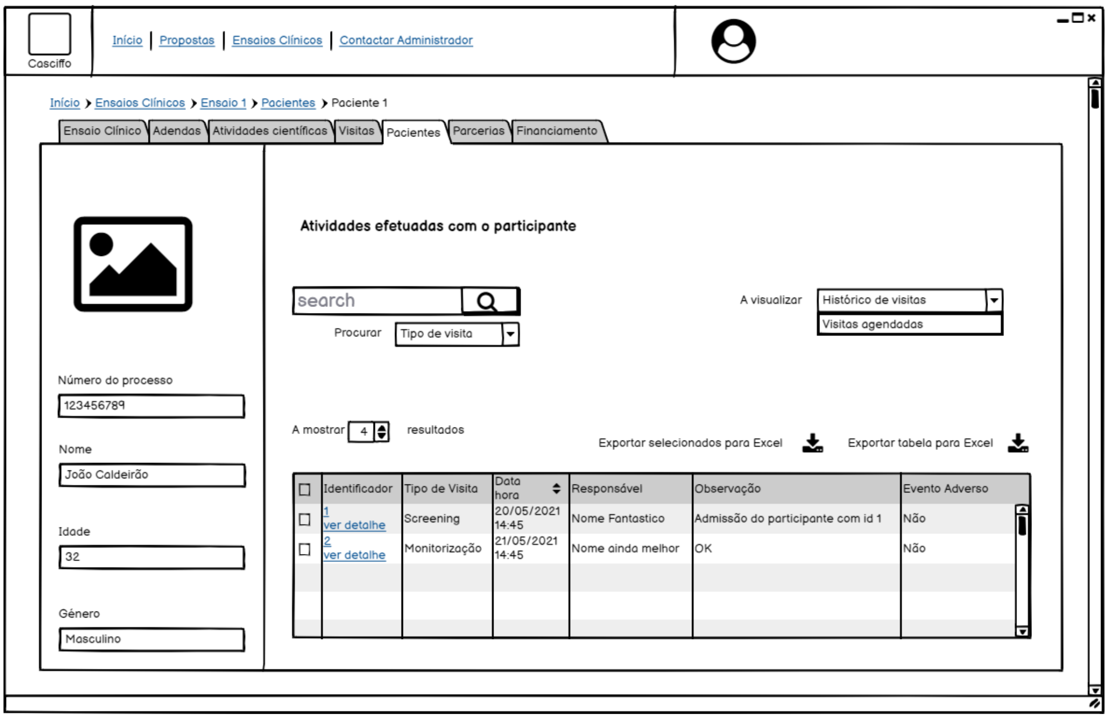
Figure 2.15: Mock overview of a visit's details.

2.6.10 Characteristics of the treatment associated with the clinical trial

The characteristics of the treatment associated with clinical trials consists of three main factors, the service area it is within, the therapeutic field and the pathology the clinical trial aims to investigate/treat. These details can all be found in the detailed view of a clinical trial under the tab "Ensaio Clínico".

2.6.11 Monitoring of the patient's behavior under trial and its attendance

The monitoring of the patient's behavior and its attendance, can be tracked via the visits and in twofold. The first is to filter the visits, under the visits tab, to show only the visits related to the participant in question. The second is to view the details of this participant in particular, from the overview of total participants in the study, clicking on the details of the participant and then checking the tab "Attendance", as in figure 2.16.


Casciffo
Início | Propostas | Ensaios Clínicos | Contactar Administrador
Início › Ensaios Clínicos › Ensaio 1 › Pacientes › Paciente 1
Ensaio Clínico
Adendas
Atividades científicas
Visitas
Pacientes
Parcerias
Financiamento
search
Exportar selecionados para Excel
Exportar tabela para Excel

Atividades efetuadas com o participante

Procurar: Tipo de visita

A visualizar: Histórico de visitas / Visitas agendadas

Número do processo: 123456789

Nome: João Caldeirão

Idade: 32

Género: Masculino

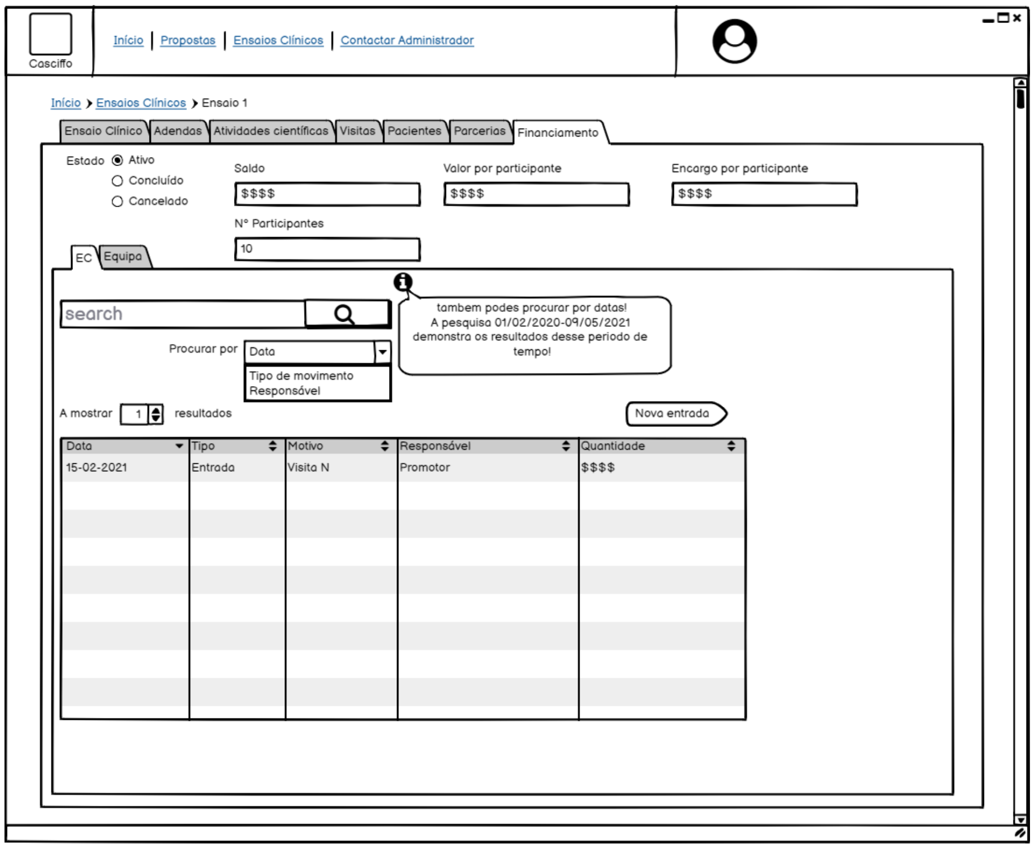
A mostrar 4 resultados

☐	Identificador	Tipo de Visita	Data hora	Responsável	Observação	Evento Adverso
☐	1 ver detalhe	Screening	20/05/2021 14:45	Nome Fantastico	Admissão do participante com id 1	Não
☐	2 ver detalhe	Monitorização	21/05/2021 14:45	Nome ainda melhor	OK	Não

Figure 2.16: Mock overview of patient details.

2.6.12 Monitoring of physical and financial assets

In regard to the monitoring of physical assets, these can be viewed under the tab "Ensaio Clínico" while in the detail page of the clinical trial in question. The assets to be monitored are documentation archives, consisting of three fields: the Volume, a Label and the total Number. CASCIFFO also offers another type of monitoring, the monetary flow. As described in section 2.4.2, each clinical trial has a financial management section that discriminates the earnings made by visit and the partitions of each team member involved in the study. Under the tab "Financiamento", the general monetary information, such as, total balance ("saldo"), amount per participant, etc., is displayed on a top section of the page. Below this section it can be observed another two tabs, differentiating the income made from the investigation team and the clinical trial itself. The first tab "EC", shown in figure 2.17, shows the income made according to the visits done, whereas the second tab "Equipa", illustrated in figure 2.18, shows the income by team member.



Estado (● Ativo) ○ Concluído ○ Cancelado

Saldo	Valor por participante	Encargo por participante	Nº Participantes
$$$$	$$$$	$$$$	10

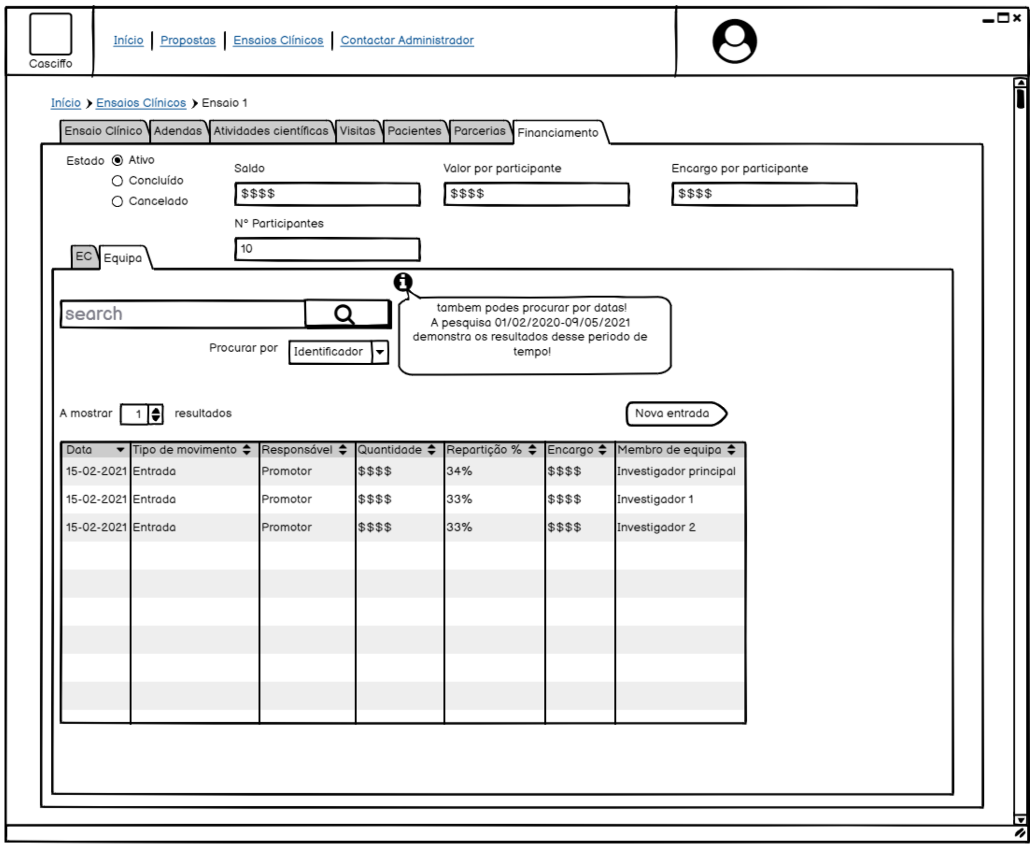
tambem podes procurar por datas!
A pesquisa 01/02/2020-09/05/2021 demonstra os resultados desse periodo de tempo!

Procurar por: Data
Tipo de movimento
Responsável

A mostrar 1 resultados

Data	Tipo	Motivo	Responsável	Quantidade
15-02-2021	Entrada	Visita N	Promotor	$$$$

Figure 2.17: Mock detailed view of the income flow made in the investigation.



Estado (● Ativo) ○ Concluído ○ Cancelado

Saldo	Valor por participante	Encargo por participante	Nº Participantes
$$$$	$$$$	$$$$	10

tambem podes procurar por datas!
A pesquisa 01/02/2020-09/05/2021 demonstra os resultados desse periodo de tempo!

Procurar por: Identificador

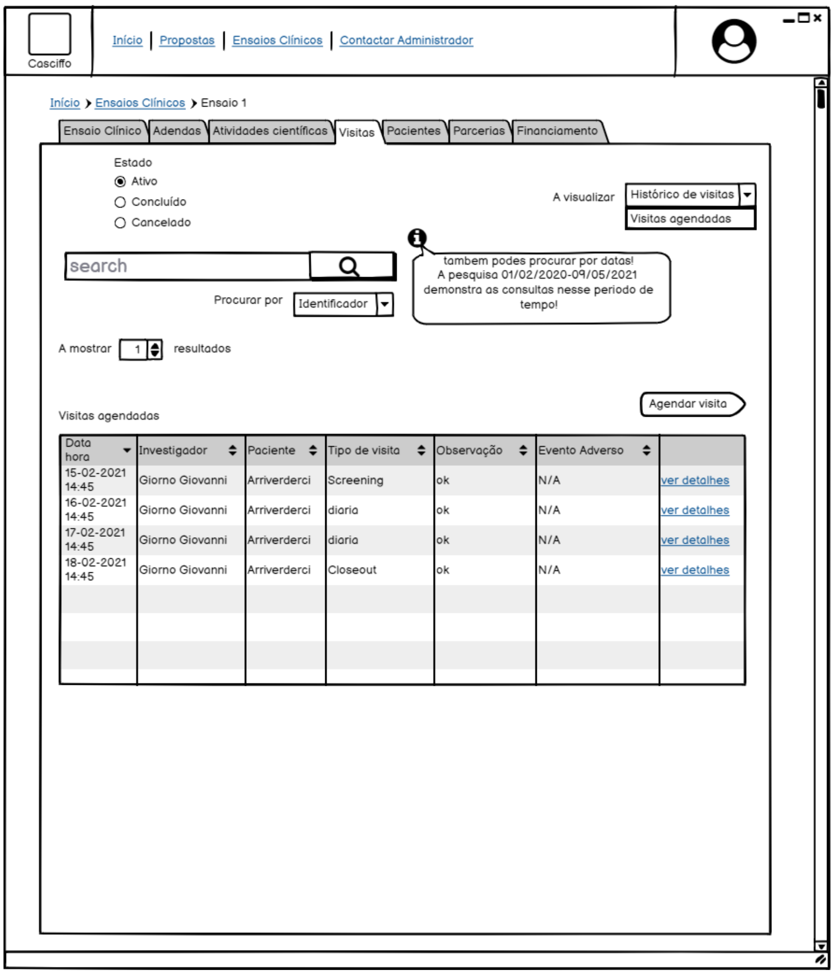
A mostrar 1 resultados

Data	Tipo de movimento	Responsável	Quantidade	Repartição %	Encargo	Membro de equipa
15-02-2021	Entrada	Promotor	$$$$	34%	$$$$	Investigador principal
15-02-2021	Entrada	Promotor	$$$$	33%	$$$$	Investigador 1
15-02-2021	Entrada	Promotor	$$$$	33%	$$$$	Investigador 2

Figure 2.18: Mock detailed view of income flow divided by the investigator team.

2.6.13 Monitoring of visits & recording of adverse events

The monitoring of visits during a clinical trial can be tracked and viewed under the tab "Visitas", in the detailed screen of a clinical trial. In this setting, any investigator (associated to the clinical trial) is able to view and track the past and scheduled visits made by the team. However, only the investigators associated to a visit are able to manipulate them. When creating a visit, that visit will automatically be associated to its creator, giving also the option to associate other investigators to the same visit, allowing them to freely edit the visit details. Along with this option, the investigator is required to fill in the fields depicted in figure 2.19. These fields are as follows: the periodicity ("Períodicidade") that can be turned ON in case the visit is a reoccurring one with the ability to schedule at a custom interval in days; the type of visit ("Tipo de visita"), that has three possible inputs, Screening, indicating the first visit, Monitoring ("Monitorização") indicating a motoring visit and finally a Closeout visit, which indicates the final visit done to a participant. In case the visit is periodic, the field of ‘end date‘ ("Data fim") becomes mandatory to fill.



Estado
(● Ativo) ○ Concluído ○ Cancelado

A visualizar: Histórico de visitas / Visitas agendadas

tambem podes procurar por datas! A pesquisa 01/02/2020-09/05/2021 demonstra as consultas nesse periodo de tempo!

Procurar por: Identificador

A mostrar 1 resultados

Visitas agendadas

Data hora	Investigador	Paciente	Tipo de visita	Observação	Evento Adverso	
15-02-2021 14:45	Giorno Giovanni	Arriverderci	Screening	ok	N/A	ver detalhes
16-02-2021 14:45	Giorno Giovanni	Arriverderci	diaria	ok	N/A	ver detalhes
17-02-2021 14:45	Giorno Giovanni	Arriverderci	diaria	ok	N/A	ver detalhes
18-02-2021 14:45	Giorno Giovanni	Arriverderci	Closeout	ok	N/A	ver detalhes

Figure 2.19: Mock overview of visits scheduled in the clinical trial.

When a visit occurs, the associated investigators will have access to the observations ("Observações") field and adverse event ("Evento adverso") warning. In addition to this, the field "Marcar presença" is now available to mark attendance.

Chapter 3

Implementation

This chapter presents the solution developed to create CASCIFFO given the functional requirements detailed previously. It describes the techniques used and choices made throughout the implementation of the solution.

It is structured with the following sections:

- Architecture: Architecture and chosen methodology.
- Data modeling: Describes the data model used.
- Back-end: Describes the back-end module.
- Front-end: Describes the front-end module.
- Continuous development: Usage of cloud-based platform Heroku for continuous development.

3.1 Architecture

The initial approach taken towards CASCIFFO was made following the monolithic architecture. A monolithic architecture consists in the development of software components within a single container. These components are often divided into three main layers, the UI, domain and database layer. From an operating system's point of view, these components operate within one single application. The advantages brought by this design are centered around simplicity, the application is easier to deploy, test, debug and monitor. Bringing this architecture into context, the first layer - database layer - developed in *PostgreSQL*, contains the data created and manipulated by the platform CASCIFFO; the second layer - domain layer - corresponds to the Back-end module, developed in Kotlin, using the framework*Spring Webflux* [15] and responsible for security and business processes within the platform; lastly the third layer - UI layer - corresponds to the Front-end module, implemented using the framework *ReactJs* and written in *TypeScript*.

One drawback of this architecture is scalability and maintenance; the need of redeploying the entire application when a change is made, for example, a change in the Front-end module should not imply an redeployment of the Back-end module as well. To solve this, the modules were separated, the front-end module and back-end module can run independently of each other. In addition, this separation favors separate deployments.

3.2 Data modeling

The data model created to satisfy the needs of the platform CASCIFFO is centered around two main entities, firstly the Proposal entity which contains information regarding the submission and progress of a proposal of a clinical investigation. Secondly the Clinical Research entity which contains information pertaining to a clinical trial or clinical observation that has had its proposal submitted, accepted and validated.

A Proposal or Clinical Research entity can refer to either a clinical trial or observation study, depending on the field *research_type*. Although a clinical trial and an observational study are different, the information held by both is identical, differentiating only on existence of a financial component inherent to clinical trials. Therefore, these investigations can be stored regardless of their type in the mentioned entities. Aside from the identical data, another reason for choosing only one entity for representing clinical trials and observational studies in each stage, proposal submission and the active stage, is the amount of data stored, which is expected to be hundreds, so it makes it easier to do queries aimed at analyzing statistics.

3.2.1 Proposal Entity

Associated to a proposal entity is the type of service it's included in, the therapeutic area and type of pathology, these three fields correspond to a foreign key to their own entities. Each of these entities consist of two fields, an identifier and a name. They were made into entities rather than simple fields to allow for better consistency in the data model, since it makes use of built-in foreign-key validations and also facilitates creation and deletion of said fields. In addition to these fields, a proposal entity also has a principal investigator responsible for submitting the proposal, an investigation team with all the investigators that can edit the proposal, a set of Proposal Comments made within the scope of the proposal, a Financial Component in case its a proposal for a clinical trial, a set of timeline events and the files associated to the Proposal.

3.2.2 Clinical Research

A clinical research has associated to it the proposal for which allowed for it to become active, a state indicating its current status, a list of visits that occur in the context of the investigation, a team of investigators, equal to the team submitted during its proposal stage and in case its a clinical trial it also has a financial component associated to it. It is to note that the financial component associated to a clinical research is different than a proposal's financial component. In addition, a clinical research can also have a list of several addenda.

3.2.3 Addenda Entity

The addenda entity, associated to a clinical research, can have a file associated to it pertaining to the change being made, a state since it must undergo a validation process before it has an affect on the clinical research and can have a list of comments.

3.2.4 States

The proposal, clinical research and addenda entity require state to track their evolution over time, thus the State entity was created, this entity holds all possible states for any proposal, clinical investigation or addenda. To differentiate between them an entity was created with the goal of interconnecting state chains. A state chain describes the flow from an initial state to a terminal state, i.e Proposal Submitted to Proposal Validated.

Next Possible States

This entity, called, Next Possible States, represents the possible transitions from one state to another, it contains two important fields that describe the chain the current state transition - `state_type` - belongs to (i.e proposal, research or addenda), as well as the general order of the transition in the chain, initial for the first possible transition, progress for any in-between and terminal indicating the end of the chain. To record the transitions, the entity State Transition was created, recording the previous state, new state, the date of transition, the type of transition (proposal, research or addenda) and finally the associated reference to what transitioned state. This reference is not a foreign key by choice, since it facilitates recording transitions in one entity rather than three different tables specifying the foreign relationship.

Files

In addition to the need of tracking state, each of these entities can have their own resources in the form of files. The files are stored on the operating system of the back-end module to avoid bloating the database with files, and only information about the full-path, file size, and name is stored in the database.

3.3 Back-end module

This section describes the summarized implementation choices of the Back-end module. It first introduces the development of the back-end module and follows into more details regarding the repositories, services, controllers, mappers and authentication respectively.

The back-end module was developed using *Spring Webflux [15]* over Spring Web due to the reactive nature. *textitSpring Webflux [15]* brings a fully non-blocking programming environment and provides asynchronous calls to the database through the *R2DBC* driver. This allows for a completely non-blocking experience from the moment an *HTTPS* request is received.

This module consists of three main layers, the Repository layer, responsible for database access, the Service layer, responsible for business logic and finally the Controller layer where the requests are received and consumed. Each layer communicates only with the layer below, not allowing any circular dependencies. The controller layer communicates with the service layer, and the latter with the repository layer. Across these layers there are exception handlers to handle any foreseen and unforeseen exceptions.

3.3.1 Repositories

The repository layer is responsible for directly accessing and manipulating data in the database.

The repositories are interfaces annotated with Spring's `@Repository`, indicating they behave as a repository, each repository should implement an appropriate public interface, in the reactive context, it can either be `ReactiveCrudRepository` or *ReactiveSortingRepository* followed by the type of entity and type of identifier (such as `ReactiveCrudRepository<Entity, EntityIdType`), where the former allows for basic CRUD operations (Create, Read, Update, Delete) without needing to implement further functionality, while the latter allows for built-in sorting functionalities. Spring also allows custom functions to be implemented either through naming conventions, such as `findAllByIdAndFieldIs(id, field)`, and through custom queries. For more complex queries another option is available, by annotating a method with `@Query(sql: String)` and passing a `string` argument with the SQL statement defined at design time. All repository methods should return a type encapsulated by either `Mono` or `Flux`. The type `Mono` should be returned when the method is expected to return between zero and a single result, while `Flux` should be used a result varying between zero and n elements are expected.

The entities used by repositories, in Kotlin, can be represented by data classes since they only contain data about the entity and have no behavior.

A key factor to let the *Spring Webflux*ring [15] framework know a data class is an entity, for the main purposes of saving and updating, is annotating said class with `@Table("table_name")`, and the identifier field with `@Id`. It is worth noting that the identifier of a database table must be numeric. Text identifiers and compound keys don't function properly with this framework, so each database table has its own unique numeric identifier.

When creating an entity, the property annotated with `@Id` must a value of `null`, otherwise, the framework will attempt to update an nonexistent entity. To work around this, we can either implement a custom repository and override the `save()` method or design the entities used to have an automatically generated identifier. The approach taken while building CASCIFFO's data model was the latter.

It should also be noted that naming conventions are taken into account between the database and the entity, the *Snake Case* naming convention used by *PostgreSQL*, is automatically converted into *Camel Case* in *Spring*, for example a table field named `created_date` is automatically converted into `createdDate`.

One downside of using *Spring Webflux* [15] and R2DBC, is that it doesn't provide mechanism of fetching and automatically building complex relationships. For example given two entities, A and B, with a relationship of 1-to-1, when fetching entity B, there isn't an automatic mechanism to build the associated entity A, after the fetch, the data will come as-is in the database, as opposed to *Spring Web* and *JPA* where such relationships can be modeled and built by the *Spring Framework*. While nested entities can exist in a data class representing a database table, since they can't be loaded through automatic means, they must be annotated with `@Transient` alongside `@Value(defaultValue)`. The former annotation tells *Spring*'s framework to ignore the annotated property when saving a new entity while the latter annotation comes into play during the mapping after a fetch, this annotation defines the default value to assign to the annotated property, in the development context, the value assigned was always `@Value("null")`.

This means that for complex entities with multiple foreign keys, with each foreign key corresponding to a nested object that needs to be manually instantiated, in order to fetch the data to build these objects would require to make that as database calls as there are nested entities which leads to performance issues. To counter this, entities were created to serve as containers of data, by aggregating all the fields required to build not only the base entity but also the nested ones. In conjunction with this, for each of these containers a repository was created, however, since the data that we want is split between tables, a custom query is needed to fetch the desired results. By using this approach we can specify all the necessary data manually. On a side note the aggregate entity doesn't need to be annotated with `@Table()` since it will exclusively used for fetching data.

3.3.2 Services

The service layer is where all business logic is handled. The approach taken to implement this layer was with interfaces together with Spring's dependency injection, all services have an interface that describes their operations. The implementation of these services must then be annotated with @Component so Spring can easily scan and auto inject the implementations to where they're used. This facilities maintenance and rearranging of code, as well as building and swapping existing implementations of a service.

Each service can communicate with other services and with their associated repositories, in addition to business logic, the request parameter validation and other exceptions are thrown and handled in this layer.

3.3.3 Controllers

The controller layer is where the *HTTPS* requests are handled, the routing of requests is handled by Spring's framework with the help of annotations that distinguish what controller handles which route and operation. To build a controller class, the class needs to be annotated with `@Controller` or in this case of a *REST API* module, `@RestController`. The main difference between the two annotations is that

`@RestController` indicates that the class is a controller where `@RequestMapping` methods assume `@ResponseBody` semantics by default. The `@RequestMapping` accepts a route path to serve as the base of the controller class, `@ResponseBody` indicates the type of response, in this case, `JSON` is used in all responses.

When building the operations of a controller class, the methods that handle route requests, depending on the type of operation, need to be annotated with one of the following `@Post`, `@Put`, `@Get`, `@Patch` or `@Delete`, each of these annotations take in an optional argument that represents the path of the route taking into account the base path if specified in `@RequestMapping(path)` on the class.

Each of the methods in a controller class can return an object or a `ResponseEntity<Type>` for more flexibility in the structure of the response. In the reactive environment that*Spring Webflux* [15] provides, each return type should be inside a `Mono` or `Flux` stream. The former represents a stream of one object, whereas the latter a stream of objects, from 0 to N objects. `Mono` and `Flux` represent streams of data using *Reactor*, these data types can be chained together like building blocks in a declarative manner of programming. First the developer explicitly declares the behavior of the chain, with functions such as `map(), filter(), groupBy(), flatMap()`. The execution of this chain only begins once the stream is subscribed to.

Returning a `Mono` or `Flux` lets the Spring framework handle when its subscribed and structures the response data, whenever a conversion to `JSON` happens the streams are automatically subscribed to. While `Mono` and `Flux` are *Spring Webflux* [15]'s standard since it operates over *Reactor*, *Kotlin* also offers another way of dealing with streams, while implementing the back-end module *Kotlin Coroutines* [5] were preferred over `Mono` and `Flux` due to its `async-await` nature and ease of use. To use *Kotlin Co-routines* the functions need to have the `suspend` keyword, i.e `suspend fun functionName(params)`, in this way, instead of returning a `Mono<Type>` we can return the `Type` directly. When dealing with `Flux` data types, *Kotlin* offers an alternative called `Flow<Type>`, the type `Flow` has a similar set of operations and behavior to `Flux`. To convert a stream of type `Flux` to a `Flow`, a simple call to the method `asFlow()` applied to the `Flux` stream will do the job.

A caveat of `Flux` and `Flow` occurs when returning objects with nested streams, (i.e returning a `Flow` of object A that has a nested `Flow` of object B in its instance), during *Spring*'s automatic conversion from object to `JSON`, the nested streams aren't represented properly, showing only a fetch status instead of the stream itself. In order to address this issue, a data transfer object (DTO) representation of each entity model was created to hold lists instead of reactive streams. In conjunction with this, several mappers were created for the conversion of entity models to their respective DTO representations.

Mono/Flux syntax vs Suspend function syntax

A *Kotlin coroutines* is an instance of suspendable computation. It is conceptually similar to a thread, in the sense that it takes a block of code to run that works concurrently with the rest of the code. However, a coroutine is not bound to any particular thread. It may suspend its execution in one thread and resume in another one. [6]

The main difference between coroutines and threads can boil down to their execution schedule. Coroutines are sequential within their scope (e.g. several coroutines can run in the same thread one after another or concurrently in a fixed thread pool, for example). Threads can run concurrently with other threads, while coroutines are sequential within their scope (e.g. several threads can run in the same thread one after another, for example). With this, we can say that while threads focus on preemptively multitasking, coroutines focus on cooperative multitasking. Furthermore, unlike threads which are usually controlled by the operating system, coroutines are managed by the user in conjunction with the programming language.

To define a coroutine function, the `suspend` keyword must precede its declaration makes it a suspending function, meaning that it can run other coroutines inside to suspend the execution of code.

On the topic of syntax choice, *Kotlin Coroutines* [5] was favored due to the simplicity and clarity of the written code. To illustrate the difference in syntax between the usage of *Spring*'s `Mono`/`Flux` and *Kotlin coroutines* we'll use a small portion of CASCIFFO's data model focusing on a proposal, its principal investigator and the states of a proposal. The entities in play will be `Proposal`, `User` - representing investigators - and `Comment`. Following the context given, a proposal has one principal investigator and can have several comments made by other investigators. The data of a proposal is split in the database, when wanting to get the full details of a proposal, a query must also be made to fetch the `User` and another to fetch the `Comment` associated to the proposal. Assuming a repository exists for each of these entities their names are their respective entities suffixed with `Repo` we can write the following code to query the database and fetch the entities needed to build a proposal with nested entity of investigator and a list of comments. Below is the representation of the method `getProposalDetails` using `Mono`/`Flux` in listing 1 and using a *Kotlin Coroutine* in listing 2.

Listing 1: Loading nested entities in an object using Mono/Flux syntax

```
fun getProposalDetails(pId: Int): Mono<Proposal> {
    return personRepo
        //findById returns Mono<Proposal>
        .findById(pId)
        // zipWith returns Mono<Tuple2<Proposal, User>>
        .zipWith(userRepo.findByProposalId(pId))
        .flatMap{
            it.t1.pInvestigator = it.t2.get()
            it.t1 //Proposal
        }
        // zipWith returns Mono<Tuple2<Proposal, (Mutable)List<Comment>>
        .zipWith(commentsRepo.findAllByProposalId(pId).collectList())
        .flatMap {
            it.t1.comments = it.t2.get().map{e -> commentsMapper.mapToDto(e) }
            it.t1
        }
}
```

Listing 2: Loading nested entities in an object using suspend function syntax

```
suspend fun getProposalDetails(pId: Int): Person {
    val proposal = proposalRepo.findById(pId).awaitSingle()
    person.pInvestigator = userRepo.findByProposalId(pId).awaitSingle()
    //here comments was changed from type Flux<Comment> to Flow<Comment>
    person.comments = commentsRepo.findAllByPersonId(pId).asFlow()
    return proposal
}
```

3.3.4 Mappers

Mappers are crucial tool in any application, allowing the conversion of one data type into another and hiding the complexity of such operation. Using Spring's powerful dependency injection a single generic `Mapper<ModelType, DTOType>` interface was created with the methods `mapDTOtoModel()` and `mapModelToDTO()`, both methods are suspend functions. During the mapping phase, nested objects of type `Flux` or `Flow` are subscribed to and subsequently transformed into lists that can be represented in *JSON*.

Each mapping conversion from DTO to Model and vice versa are called within the controllers.

3.3.5 Authentication

On top of these layers lies Spring's own framework, and in it the Web Filters. Web Filters are methods that run during an incoming request. Through Spring's framework, they intercept and perform operations with the request data before control is given to a controller handler. These Web Filters are especially useful to perform authentication operations on the incoming requests.

The authentication was implemented with *JSON Web Tokens* (JWTs), each request that requires authentication must have a valid JWT in the header 'Authentication' of the request. This JWT in addition to the standard fields, also holds user information, namely the user email that is used to uniquely identify each user. To further security, a role-based access control was defined as mentioned previously in the document. This can be implemented in a few steps, first and foremost is to implement*Spring Webflux* [15]'s `ReactiveUserDetailsService`, which is responsible for fetching a user based on a passed argument `username`, this method will be called during the interception of a request that requires authentication. The next step is to create an authentication manager that implements `ReactiveAuthenticationManager` and overriding the method `authenticate`, this manager is responsible for extracting and validating the information from a JWT. Note that both the classes that implement these interfaces need to be annotated with `@Component`. The need for the annotations comes from a useful feature of *Spring*'s framework, dependency injection, by annotating these classes with `@Component` or methods with `@Bean` allows *Spring* to scan and detect the classes and auto inject them into other parts of the code where they are passed as arguments for methods or constructors. *Spring* taking control of this aspect of the code, can be referred as *Inversion to Control* (IoC) where a framework takes control of a portion of the program or code, instead of the programmer explicitly controlling it.

To make use of this newly created components we need to add the authentication manager to the `WebFilter` chain that can be accessed and configurable from a class annotated with `@Configuration` and implementing the method `springSecurityFilterChain`, annotated with `@Bean`. This method receives an argument `http` of type `ServerHttpSecurity` and should also receive the created authentication manager component as an argument. Finally to add it to the `WebFilter` chain, the method `addFilterAt()` is used, applied to the argument `http`.

This describes how a JWT's information is extracted and validated, to implement a role-based access control, one can choose Spring's annotation-based authentication, by annotating the required methods or controller classes with `@PreAuthorize(string)`. The passed argument is a function name alongside its arguments, in this context, the most adequate ones would be `hasAuthority()` or `hasAnyAuthority()`. Another possible way of implementing this requirement is by utilizing the `http` argument mentioned previously inside the method `springSecurityFilterChain` and programmatically add these rules to each route, as shown in the coding listing 3. By utilizing the methods `authorizeExchange()` followed by `pathMatchers(route)` and applying `hasAnyAuthority(authorities)` or `hasAuthority(authority)`. An authority is simply a constant with a string value specifying a role, for context, some of the authorities used were `SUPERUSER_AUTHORITY` and `UIC_AUTHORITY`.

Listing 3: Configuration of spring security filter chain

```kotlin
@Bean
fun springSecurityFilterChain(
    converter: JwtServerAuthenticationConverter,
    http: ServerHttpSecurity,
    authManager: JwtAuthenticationManager
): SecurityWebFilterChain {

    val jwtFilter = AuthenticationWebFilter(authManager)
    jwtFilter.setServerAuthenticationConverter(converter)

    http
        .addFilterAt(jwtFilter, SecurityWebFiltersOrder.AUTHENTICATION)

    //example of routing configuration
    http
        .authorizeExchange()
        .pathMatchers(HttpMethod.GET, ENDPOINTS_URL)
            .permitAll()
        .pathMatchers(HttpMethod.POST, ENDPOINTS_URL)
            .hasAnyAuthority(SUPERUSER_AUTHORITY, UIC_AUTHORITY)
        .pathMatchers(HttpMethod.DELETE, ENDPOINTS_URL)
            .hasAuthority(SUPERUSER_AUTHORITY)

    //other http configurations
    ...

    return http.build()
}
```

Both the discussed approaches were used, having settled with programmatically adding new rules to the routes themselves. This choice was made based on performance issues with the annotation-based authentication, the annotation `@PreAuthorize()` was adding an overhead of up to 1 second per request, whereas programmatically this issue was not observed. The cause can be due to*Spring Webflux* [15], since the annotation `@PreAuthorize` is more commonly used with Spring Web.

3.3.6 Exception handling

On the topic of business rules, once one is broken, an exception is raised with information regarding the cause. Business rules are defined in the service layer and conversely it is where incoming requests are validated before performing the requested process. If the information passed along the request isn't valid, an exception is manually thrown with an appropriate `HTTP status` and a reason, since the back-end module exposes an API through http requests.

With exceptions being raised whenever there's invalid data on incoming requests, the response must come with appropriate information on why the request failed or was rejected. For example, when requesting a resource that requires authentication yet the request sent none, the response will provide sufficient information so the one that made the request knows that for accessing that specific resource, it is also required to send authentication and how to send it.

To handle these exceptions and others that can occur during run-time, such as a database integrity violation or in case the database is unreachable among other possible run-time exceptions we can make use of great features provided by *Spring*'s framework. We first looked at utilizing custom exceptions annotated with `@ResponseStatus`, by annotating a class that extends `Exception`, we can define the response status code and the reason that caused the exception to be thrown, as illustrated by the listing 4.

Listing 4: Example of an exception annotated with @ResponseStatus

```
@ResponseStatus(
    code = HttpStatus.NOT_FOUND,
    reason = "Specified resource Id doesn't exist."
)
class ResourceNotFoundException(): Exception()
```

This approach is simple and makes use of *Spring*'s automatic converter to *JSON*. The issue lies in its simplicity, it doesn't allow us to change anything other than reason and status code, the body itself is handled by *Spring*, an example of an answer can be observed in the code listing 5. We would also need to create several exception classes for each edge case that requires an exception to be thrown.

Listing 5: Example response using *Spring*'s automatic response converter.

```
"{
    "timestamp": "2022-11-09T12:24:21.609+00:00",
    "path": "/proposals/999",
    "status": 404,
    "error": "Not Found",
    "message": "Specified resource Id doesn't exist.",
    "requestId": "c0d35eab-2"
}"
```

Another possible solution is to create specific exception handlers, using the annotation `@ExceptionHandler(SampleException::class)` on a method and passing the class type of exception to catch as its argument, we can process specific exceptions locally within the controller classes themselves. This resolves issue of the first approach in the sense that we can make use of the incoming request and outgoing response to write our own defined response body. The trade offs, however, come at a cost of maintenance, the exception handlers will be tightly coupled to their controllers and one exception handler can only process one type of exception.

Another analyzed approach was the implementation of `ErrorWebExceptionHandler` and overriding the method `override fun handle(ServerWebExchange, Throwable): Mono<Void>`. This method will catch any exceptions not handled by the exception handlers mentioned earlier, here we have access to the argument of type `ServerWebExchange` which includes in its properties the request, the response and other useful information about the this particular connection that resulted in an exception. Here we are also able to access and customize the body property of the response as we wish.

Yet another studied approach was the usage of classes annotated with `@RestControllerAdvice`, these classes can intercept exceptions when controllers, respectively annotated with `@RestController` raise exceptions. With this class we can centralize all exception handlers in one place, decoupling the controllers of their exception handlers and moving the latter here.

The last studied solution was the usage of the exception class `ResponseStatusException`, which on the contrary to the first mentioned approach, we define raise the exceptions and set their status and reason locally when the exception is thrown. This reduces the amount of exception classes since one type of exception can be used for any issue which great for prototyping and testing. A trade off of this approach will be the increased difficulty of code management and duplicated code inside the code blocks that raise exceptions.

The final solution taken was a mix and match of the described approaches. A global exception handler was created by utilizing the creation of a class annotated with `@RestControllerAdvice` and implementing `ErrorWebExceptionHandler` to catch any unexpected exception during run-time. For specific exceptions, the usage of a method annotated with `@ExceptionHandler` and accepting the argument `ResponseStatusException::class` facilitated in manipulating the response body. With this solution, we benefit from global exception handling and decoupled controllers.

3.3.7 Testing

To assure the quality and that each layer was functioning properly several integration tests where made using *Spring Unit* [14] testing framework, the integration tests used an in-memory database based on H2 so that they can run without affecting the production database. The tests made verify each single layer, starting from the repositories, followed by the services together with repositories and finally the full layer chain of controller - service - repository. The final layer was tested with the Postman tool to verify the responses and facilitate the ability to manipulate the header requests as well as viewing the response headers.

3.4 Front-end module

The front-end module was developed using *ReactJs* [7], version 18, mainly due to its popularity and ease of use alongside a super set of *JavaScript*; *TypeScript*, version 4. The module is built using a node package manager, the popular *npm* [2], and deployed with *npx webpack* during development.

This module was developed to be a single page application (SPA) and consists of three main components, the View, Model and Service.

Starting from the service component, this is where data is requested via *HTTPS* to the back-end module. All services are stateless and responsible for requesting data.

The model component is where all model entities are held.

Finally the view component is where the magic happens, the business logic of the Front-end module is near the client and occurs within each Component depending on its responsibility. In *ReactJs* [7], the UI is built with components, each Component can be viewed as a building block of a page, it is reusable and provides ways to handle state and logic.

Moving to security in the front-end module, role based access-control is a requirement, in order to satisfy this, a login system was developed using *Higher Order Components* (HOC), these components are the same as any other, however, they have the responsibility to perform validations before rendering other components. To implement this functionality, any page that requires a user being authenticated or having a specific role will be chained with a HOC, first the HOC is rendered, which performs validations on the user info available on the browser session storage and makes a decision on whether to render the page requested or display an error. Upon a successful login, the back-end module sends a valid JWT, the user identifier and the roles associated to the logged in user that is then stored within the browser's session storage.

According to the functional requirements the screens were developed based on the mock ups detailed in the report. In addition, the module follows the standard of a *Progressive Web App* (PWA), allowing the application to be downloaded from the browser, to work as a mobile app and offline functionality albeit limited. The dashboard screen currently displays several useful stats through graphs and tables.

Each page corresponds to a route that is handled by *React.Router*, however, during development an issue arouse which was when accessing an `URL` other than base one first such as, `[base-path]/propostas` instead of `[base-path]`, it displayed the error `Cannot GET [URL]`, this issue was due to how *React.Router* interacts with the `URL`. *React.Router* is only loaded when the base `URL` is first called, after which it takes control of the route handling. To resolve this, a simple line of code was needed to be added to the server start up file, which redirects any all `URL` requests to the base path first and foremost. This way, *React.Router* is always loaded and becomes capable of handling requests as expected.

3.5 Continuous development

The approach to the development of CASCIFFO involved a technique of continuous development and testing, this was possible by deploying both modules to the cloud platform Heroku and having them available for testing. With each new feature a new deployment was released without compromising having the this testing environment go down while working on new features.

Chapter 4

Conclusions

The management and monitoring of clinical trial data is a arduous task for the researchers within the UIC, who have to contribute a considerable amount of effort and time into organizing the overview of clinical trials. To decrease this burden and optimize the management of clinical trials, the platform CASCIFFO has been proposed.

In this report, the functional requirements as well as the current development status of the platform CASCIFFO .

4.1 Current status

The front-end module has all screens finished, and is currently under quality testing.

4.2 Future work

The current plans for the future work in the development of CASCIFFO is a quality review and having the it fully deployed on premises.

Bibliography

[1] *About Advarra.* Mar. 2022. URL: https://www.advarra.com/about/.

[2] *About npm.* Jan. 2022. URL: https://docs.npmjs.com/about-npm.

[3] Comissão de Ética para a Investigação Clínica. *Apresentação.* Jan. 2022. URL: https://www.ceic.pt/missao.

[4] Hospital Professor Doutor Fernando Fonseca. *Introduction of the Hospital Professor Doutor Fernando Fonseca.* Jan. 2022. URL: https://hff.min-saude.pt/.

[5] JetBrains. *Coroutines.* Nov. 2022. URL: https://kotlinlang.org/docs/coroutines-overview.html.

[6] JetBrains. *Coroutines Basics.* Nov. 2022. URL: https://kotlinlang.org/docs/coroutines-basics.html#your-first-coroutine.

[7] *React.* Jan. 2022. URL: https://reactjs.org/.

[8] *Revolutionize the Way You Conduct Clinical Research.* Mar. 2022. URL: https://info.advarra.com/clinical-conductor-demo-request-capterra.html.

[9] *Revolutionize the Way You Conduct Clinical Research.* Mar. 2022. URL: https://realtime-ctms.com/clinical-research-software/.

[10] *Spring boot.* Mar. 2022. URL: https://spring.io/projects/spring-boot.

[11] *Spring Data R2DBC.* Mar. 2022. URL: https://spring.io/projects/spring-data-r2dbc.

[12] Portugal Clinical Trials. *Hospital Professor Doutor Fernando Fonseca.* Jan. 2022. URL: https://portugalclinicaltrials.com/pt/centros-de-investigacao-clinica/hospital-professor-doutor-fernando-fonseca/.

[13] Portugal Clinical Trials. *O que é um estudo clínico?* Jan. 2022. URL: https://portugalclinicaltrials.com/pt/porque-e-que-os-estudos-clinicos-sao-importantes/.

[14] *Unit Testing.* Mar. 2022. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html#unit-testing.

[15] *Web on Reactive Stack.* Mar. 2022. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#webflux-new-framework.

[16] *What is PostgreSQL?* Jan. 2022. URL: https://www.postgresql.org/about/.

[17] *Yarn.* Jan. 2022. URL: https://yarnpkg.com/getting-started.