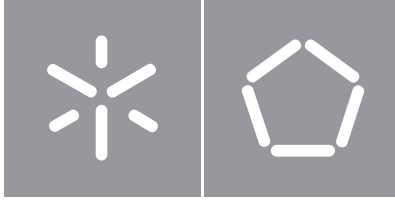


**University of Minho**  
School of Engineering

Diogo Paulo da Costa Pereira

**Adaptive Questionnaires  
using an Workflow Engine**





**University of Minho**  
School of Engineering

Diogo Paulo da Costa Pereira

**Adaptive Questionnaires  
using an Workflow Engine**

Masters Dissertation  
Integrated Master in Informatics Engineering

Dissertation supervised by  
**António Luís Pinto Ferreira Sousa**  
**Helena Fernández López**

# Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

## License granted to users of this work:



### CC BY

<https://creativecommons.org/licenses/by/4.0/> *[Esta licença permite que outros distribuam, remixem, adaptem e criem a partir do seu trabalho, mesmo para fins comerciais, desde que lhe atribuam o devido crédito pela criação original. É a licença mais flexível de todas as licenças disponíveis. É recomendada para maximizar a disseminação e uso dos materiais licenciados.]*

# Acknowledgements

I am truly grateful for my supervisors at University of Minho, António Luís Pinto Ferreira Sousa and Helena Fernández López, for all their support and availability throughout all phases of the project. Their help was fundamental to both writing and developing steps, despite their full schedules they would always try to help.

I would also like to thank my external supervisor at Promptly Health, Ivan Pereira, for sharing his knowledge and guiding me throughout this year.

I would like to express my appreciation to the amazing people I met through my academic journey, for all the moments that have always cheered me up through the course. To all my friends, thank you for your patience and support during this period, your presence and friendship are one of the foundations of my happiness.

I would also like to thank my parents and my brother for all the support and advice in all my decisions. To my family, thank you for all your guidance. Their wisdom has made me the person I am today.

## Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, January 2023

A handwritten signature in black ink that reads "Diogo Paulo da Costa Pereira". The script is cursive and fluid, with the first letters of each word being capitalized and prominent.

Diogo Paulo da Costa Pereira

# Abstract

Nowadays medical monitoring is very important and something that helps us a lot. Some supporting tools are already used for this, such as Patient Reported Outcome Measures (PROMs) and Patient Reported Experience Measures (PREMs), which are standardized medical surveys used to evaluate the quality of care or patient experience from the patient's viewpoint. They can often be transformed into Computerized Adaptive Tests, which aim to determine the best set of questions to ask each patient based on their previous answers. With these tools, it becomes possible for most patients to finish their forms, because they no longer have extensive forms, which increases the response rate to them. This is beneficial for both the patients and the healthcare providers. In the case of the patients, they get a more careful follow-up that helps them to have a better lifestyle and well-being. On the provider side, with additional information from the patient, healthcare providers can provide better, more personalized care and have a larger data set for future research. So, this dissertation is based on these improvements at the medical level and also in other improvements at the business level.

For this, a solution was developed that uses BPMN as a high level language, Camunda as an engine, a user interface and the main component of the solution. This last one connects all the elements and makes it a workflow engine capable of processing forms coded in BPMN diagrams.

**Keywords** Workflow, Engine, PROM, BPMN, Camunda

## Resumo

Nos dias de hoje a monitorização médica é muito importante e algo que nos ajuda bastante. E para isso já são usadas algumas ferramentas de apoio, tais como as Medidas de Resultados do Reportadas pelo Paciente (PROMs) e as Medidas da Experiência Reportadas pelo Paciente (PREMs), que são questionários médicos padronizados utilizados para avaliar a qualidade dos cuidados ou a experiência do paciente do ponto de vista do mesmo. Podem muitas vezes ser transformados em Testes Adaptativos Informatizados, que visam determinar o melhor conjunto de perguntas a fazer a cada paciente com base nas suas respostas anteriores. Com estas ferramentas, passa a ser possível que a maior parte dos pacientes acabem de responder aos seus formulários, pois deixam de ter formulários tão extensos, o que por sua vez aumenta a taxa de resposta aos formulários. Isto é algo benéfico tanto para os pacientes, como para os serviços de saúde, pois no caso dos pacientes, eles passam a ter um acompanhamento mais cuidadoso, o que ajuda a um melhor estilo de vida e bem estar. Do lado do prestador, com informação adicional do paciente, os prestadores de cuidados de saúde podem fornecer melhores e mais personalizados cuidados e ter um conjunto de dados maior para investigação futura. Do lado do prestador, com informação adicional do paciente, os prestadores de cuidados de saúde podem fornecer melhores e mais personalizados cuidados e ter um conjunto de dados maior para investigação futura. Assim sendo esta dissertação tem como base estas melhorias a nível médico e também outras melhorias a nível empresarial.

Para tal, foi desenvolvida uma solução que utiliza BPMN como linguagem de alto nível, Camunda como motor, uma interface para o utilizador e foi implementado um componente da solução que liga todos os elementos e faz com que exista um workflow engine capaz de processar formulários codificados em diagramas de BPMN.

**Palavras-chave** Workflow, Motor, PROM, BPMN, Camunda



# Contents

- 1 Introduction 1**
  - 1.1 Context . . . . . 1
  - 1.2 Motivation . . . . . 2
  - 1.3 Objectives . . . . . 2
  - 1.4 Dissertation Structure . . . . . 2
  
- 2 State of Art 4**
  - 2.1 Forms . . . . . 4
    - 2.1.1 PROM . . . . . 4
    - 2.1.2 PREM . . . . . 5
    - 2.1.3 PROM vs PREM . . . . . 6
    - 2.1.4 Google Forms . . . . . 6
    - 2.1.5 Survey Monkey . . . . . 7
  - 2.2 Workflow Language . . . . . 7
    - 2.2.1 BPMN . . . . . 7
    - 2.2.2 DMN . . . . . 10
    - 2.2.3 YAWL . . . . . 11
    - 2.2.4 CWL . . . . . 12
    - 2.2.5 BPMN vs DMN vs CWL vs YAWL . . . . . 12
  - 2.3 Workflow Engines . . . . . 13
    - 2.3.1 Camunda . . . . . 14
    - 2.3.2 BPMN.io . . . . . 14
  - 2.4 Technologies . . . . . 14
    - 2.4.1 JavaScript . . . . . 14
    - 2.4.2 Node.js . . . . . 15

|          |                                    |           |
|----------|------------------------------------|-----------|
| 2.4.3    | Express                            | 15        |
| 2.4.4    | Pug                                | 15        |
| 2.4.5    | Postman                            | 16        |
| 2.4.6    | REST API                           | 16        |
| <b>3</b> | <b>User Stories</b>                | <b>18</b> |
| <b>4</b> | <b>Development</b>                 | <b>21</b> |
| 4.1      | Camunda Modeler                    | 24        |
| 4.2      | Camunda Engine                     | 25        |
| 4.3      | Implementation                     | 26        |
| <b>5</b> | <b>Proof of Concept</b>            | <b>31</b> |
| <b>6</b> | <b>Conclusions and future work</b> | <b>38</b> |
| 6.1      | Conclusions                        | 38        |
| 6.2      | Prospect for future work           | 39        |

# List of Figures

- 1 Benefits of PREM and PROM together (extracted from [8]). . . . . 6
- 2 BPMN elements (extracted from [19]). . . . . 9
- 3 BPMN diagram example. . . . . 10
- 4 DMN decision table example. . . . . 11
- 5 YAWL example extracted (from [18]). . . . . 12
- 6 Languages comparison. . . . . 13
  
- 7 User Story - Language definition. . . . . 18
- 8 User Story - Coding a form. . . . . 18
- 9 User Story - Export a form. . . . . 19
- 10 User Story - Load a form. . . . . 19
- 11 User Story - Execute a form. . . . . 20
- 12 User Story - Answer a form. . . . . 20
  
- 13 Solution's architecture. . . . . 22
- 14 Solution's architecture flow. . . . . 23
- 15 Camunda Modeler. . . . . 24
- 16 Camunda Modeler example. . . . . 25
- 17 Solution structure. . . . . 27
- 18 Start form. . . . . 28
- 19 GET tasks. . . . . 28
- 20 GET process instances. . . . . 29
- 21 DELETE process instances. . . . . 29
- 22 Create process instance. . . . . 29
- 23 GET question. . . . . 30
- 24 Answer question (Complete task). . . . . 30

25 Home page. . . . . 33

26 Start a form. . . . . 33

27 First question example. . . . . 34

28 Question example. . . . . 35

29 End of a form. . . . . 36

30 List of instances running. . . . . 36

31 List of tasks running. . . . . 37

# Chapter 1

## Introduction

### 1.1 Context

One of the main problems for the medical providers that use Patient Reported Outcome Measures (PROMs) or Patient Reported Experience Measures (PREMs), in other words, standard and validated medical questionnaires, is to encourage their patients to respond to them. If the questionnaire contains many questions, the patient get demotivated, and may not answer all questions. To overcome that problem, it is important to just ask patients the questions that they should answer. There are other problems that demotivate patients, for example, not understanding the motive behind the questionnaire or being afraid to provide information that can affect their care. Another problem is when patients with low digital literacy face digital questionnaires, and are unable to provide answers.

Summarizing all the above information, the problem is basically to motivate patients to answer all the questions that really matter to them and, consequently, improve the response rate and their satisfaction. For this, only the appropriate questions for each case should be presented to them. For example, a young patient may not need to answer a question about his or her flu vaccination, whereas this should be required for an adult or elderly patient. The core functionality provided by the system developed under this dissertation is providing means for professionals to adapt the questionnaires to the patients, taking into account the answers to the questions already answered. For example, if a patient answers "A" to question 1, the questionnaire may jump to question 3, but if the patient gives another answer, for instance "B", the questionnaire may proceed to question 2. It is like building a smart form that adapts to the answers given to it.

## **1.2 Motivation**

One of the main reasons to develop this dissertation is to improve the satisfaction of the patients when replying to health questionnaires known as PROMs and PREMs. One way to do that is to reduce the amount of questions in the questionnaires that patients are asked to fill during/after an appointment, during/after a treatment prescribed by the doctor and so on. To do this, we have to “give some intelligence to the form”, so when a patient chooses one answer out of the others, with that answer, the form will proceed to the next question based on it. The use of PROMs and PREMs is also beneficial to doctors because they can make decisions based on how health care affects patient health and well-being (PROMs) and on patients' experiences of the care they receive (PREMs).

## **1.3 Objectives**

The main goal of this dissertation is to develop a solution based on a workflow engine, capable of creating Computerized Adaptive Tests (CATs), in an intuitive way. To achieve that, it is necessary to study what a workflow engine is, the most appropriate high-level language to be used, techniques to implement it and, afterwards, implement the service capable of generating a workflow engine that will be using a low-level coding language, for people who are not used to the technicalities of it, and will be able to use it to build proper forms, in this case, either a PROM or a PREM.

## **1.4 Dissertation Structure**

This dissertation is structured in seven chapters in total, where the first part contains introductory material (three chapters) and the second part the core of the dissertation (four chapters).

This chapter provides a context of the study as well as its motivations, objectives and how the dissertation is structured.

The next chapter introduces the state of art, which covers related works recently published and introduces the definitions of concepts and technologies used in the dissertation. This chapter is one of the most important, as it contains information regarding PROMs, PREMs and the technologies used to develop the “workflow engine”.

Chapter 3 presents the user stories that will be used to specify this solution.

Chapter 4 addresses the development of specifications, as well as how the problem will be approached, what additional supportive materials will be used, how the solution was implemented and the outcome of

the dissertation with the discussion of it.

Chapter 5 presents a proof of concept regarding the solution developed in the dissertation as well as some real applications of it.

Chapter 6 depicts the conclusion of the dissertation as well as some features or improvements that could be implemented in future work.

## **Chapter 2**

### **State of Art**

This chapter explains concepts regarding the theme of the dissertation and some enlightenment of the technologies that were studied and used to develop the service responsible to create the workflow engine.

#### **2.1 Forms**

A study was conducted on two different forms used to gather data directly from patients, namely PROMs and PREMs, as well as two well-known online platforms used to publish surveys, Google Forms and Survey Monkey.

##### **2.1.1 PROM**

Nowadays, it is possible to provide personalized medical monitoring to patients, which will bring benefits to each individual, which a few years ago was unthinkable. No one ever thought that it would be possible to have such good and evolved medical follow-up of every relevant aspect of a person's overall health, because in the old days people rarely went to the doctor, or never went at all. It also brings benefits to society, because it can extend the working life of the population as well as their lifestyle, and increase longevity with quality of life.

Questionnaires are filled out by patients, informing their health care providers about their health status over time and how they are reacting to the prescribed medication. These types of forms are called PROMs.

By definition, PROMs use the opinions and views of patients to evaluate their health status and well-being. They are validated and standardized self-reporting instruments [7].

There are different types of PROMs that can be [7]:

- generic: measuring health status and quality of life that are common to the most part of the patients;
- disease-specific: used to a specific disease (eg. for a type of diabetes);



- condition-specific: used when applied in a certain condition (eg. applying in a certain population age segment).

To highlight the value that PROMs provide, some positive aspects of their use are presented [7]:

- more personalized and detailed medical follow up regarding each individual;
- if a treatment isn't doing well to a person or the person isn't adapting well to the medicine prescribed, the doctor can easily intervene;
- increase of the population longevity;
- less ill people;
- better lifestyle of the population.

Regarding limitations, there is probably only the subjectivity of the person who is answering the PROM and understanding what the person wants to transmit, because not everyone can communicate so well.

### **2.1.2 PREM**

Patient Reported Experience Measures (PREM) are not far behind from PROMs when it comes to importance, as without them it would not be possible to know what to improve and where. They are used so that doctors can know what to improve and what to do in certain situations, since PREMs are forms in which the patient gives their opinion and view regarding a service provided. For example, when a patient goes to an appointment and at the end fills out a form about his satisfaction or when the patient fills out a form in which he or she would be asked about the experience with a certain medical prescription made by a healthcare provider. PREMs are also self-reporting instruments.

PREMs have the following benefits [7]:

- improves quality of service;
- helps the society to set standards;
- can improve general quality of life.

Regarding limitations, it is the same as PROMs, since it depends on the answers provided by patients and their ability to express themselves.

### 2.1.3 PROM vs PREM

So, as mentioned before in Sections 2.1.1 and 2.1.2, both PROMs and PREMs are answered in the patient's perspective and view giving his opinion in both subjects. PROMs and PREMs can be used together to fully assess quality of care as well as the value of health services delivered. To monitor patients through a patient-centred approach it's essential to have the perspective of their side.

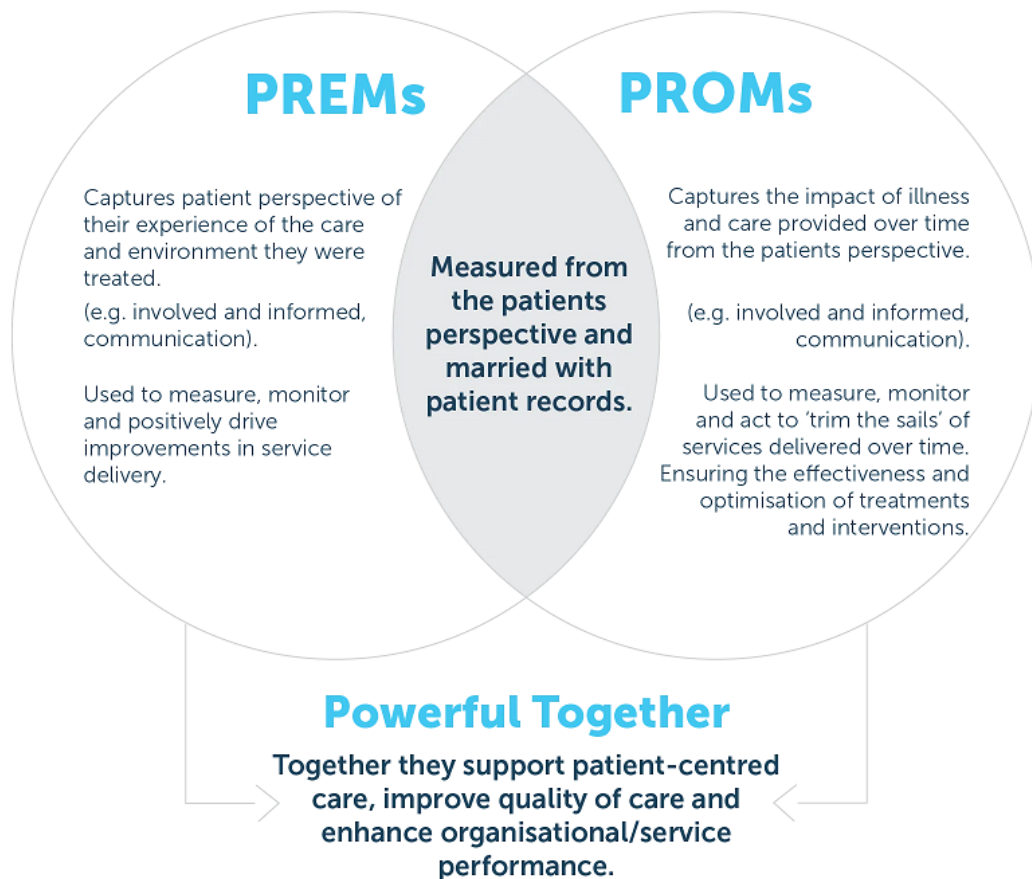


Figure 1: Benefits of PREM and PROM together (extracted from [8]).

### 2.1.4 Google Forms

Google introduced Google Forms around 2018 as a web platform compatible with pretty much all operative systems.

It's a tool where it's possible to build a form from scratch or using a template, to a fully completed and complex form with a lot of attachments, images, different types of possible answers, a board with statistics from the answers given to the person/company that has made that form, to check how the form has been doing and later compare the different answers or, in the case of a test, understand why so many people

got that given question wrong or right. This platform has the capability to create adaptive questionnaires.

Since the data utilized in this subject is sensitive, this platform is ruled out because it stores all the information, which is a liability.

### **2.1.5 Survey Monkey**

Survey Monkey is also a web platform, created in 1999 by Ryan Finley [15]. It's similar to Google Forms, but a big part of the features are locked, because they are paid. It's also a little bit confusing to use compared to Google Forms, at least to the average person and for that it's probably not used as much. Apart from this two facts, if we can pay for it and we really understand it, it can be very useful specially in the analysis of the data given in the answers by the users. With this platform is possible to create conditional questionnaires.

As referred in Section 2.1.4 this platform cannot be used too, because it is a liability when it comes to the sensitive data.

## **2.2 Workflow Language**

To be capable of choosing the most appropriate language to be used, there had to be a careful study and analysis of some languages, namely, BPMN, DMN, YAWL and CWL. Below, an introduction to each language is presented and, after that, in Section 2.2.5, the languages are compared.

### **2.2.1 BPMN**

Firstly, Business Process Model and Notation (BPMN) is a very complete language but when used for the first time without any programming and technology knowledge, the basic concept of the language would be easily understood, because it is a very user-friendly language. Although it has a simple drag and drop system to build the workflows, if an in-depth learning is made about BPMN, it allows the elaboration of considerably more complex solutions with more functionalities, taking longer and making it more challenging. So for pros regarding BPMN they are [19]:

- intuitive;
- well-structured and organized;
- user-friendly;

- semantically strong;
- easy to build an workflow;

All this points contribute to a language that would fit in this problem because the final user will be people who are not that much comfortable using this type of technology so it would be easier for them to have a more friendly language to use when putting the questionnaires together for most of them.

The most important goal of BPMN is to provide a notation that is easy to understand by business users from business analysts who draft the initial concepts of processes, to the technical developers responsible for implementing them, and finally to the business staff who implement and monitor such processes. BPMN was originally released by the Business Process Modeling Initiative in 2004 as a graphical notation, partly inspired by UML Activity Diagrams, to represent the graphical layout of business processes [19]. But nowadays is managed by OMG, Object Management Group.

Some drawbacks are:

- The initial BPMN sketch, usually by the business, normally may need many versions to arrive at a final diagram which allows to go for the implementation;
- It is not straight forward to represent different roles since the usual concept of lanes in pools (lanes and pools are a type of elements in BPMN) might not be enough or lead to huge diagrams.

The majority of components used in BPMN are pretty easy to distinguish and to understand what they do, as it can be seen in the next image that has been extracted from [19].



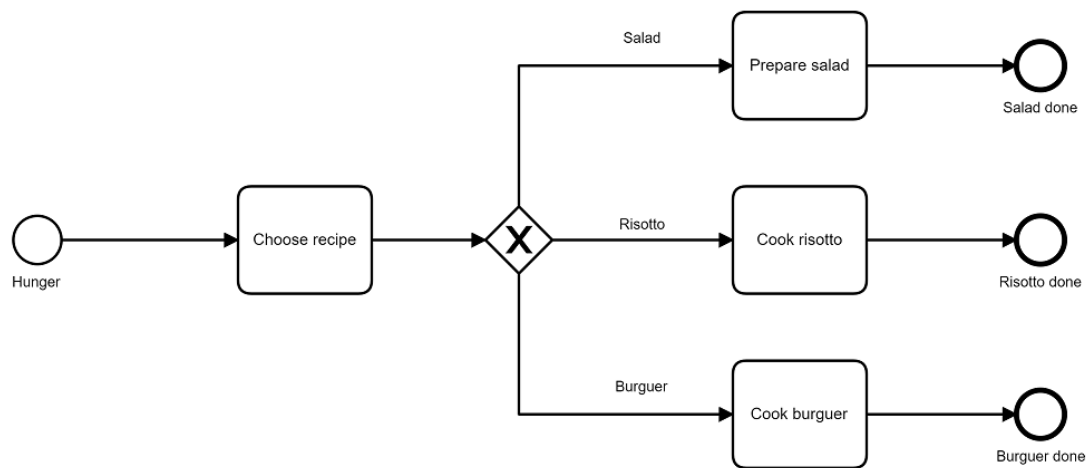


Figure 3: BPMN diagram example.

### 2.2.2 DMN

DMN, also known as Decision Model and Notation, is a language for modeling and notation that specifies precise business decisions and rules. It is also easy to read by the different types of people involved in the decision management. Some of the people that are included are the ones that specify the rules and monitor their application as well as business analysts.

DMN is managed by OMG, the same one that also manages BPMN. DMN was developed to compliment BPMN and CMMN. All the three go side by side, providing a mechanism to model the decision making, connecting with processes and cases. Although they can be used individually, they were carefully designed to be complementary. The three constitute the called “triple crown” of process improvement standards, because when companies used them combined, they can choose which one is the more appropriate to each type of activity modeling.

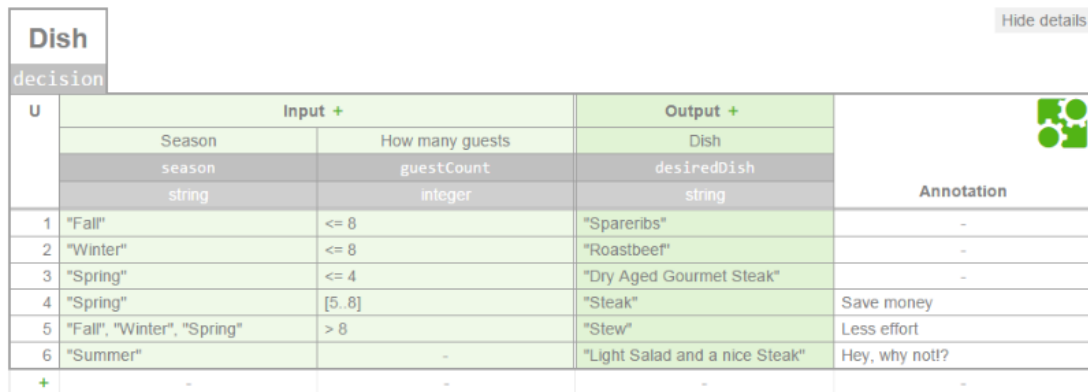
DMN usually is used to define manual decision making, specify the requirements for automated decision making and representing a complete, executable model of decision making. In the addition of the decision requirements, DMN also defines a low code language to the logic that is used in each decision that exists.

Some benefits of using DMN are:

- provides a standard notation for decision tables;
- handles decision making while BPMN and UML do not;
- facilitates IT and analytic roles to have better communication;

- projects improvements due to the use of business rule management systems, that allow quicker changes;

Figure 4 shows an example of a DMN decision table. The table is meant to decide which dish to choose based on two parameters, the season and the number of guests. For example the row number 3, if the season is Spring and the number of guests is lower than 4, then the chosen dish would be Dry Aged Gourmet Steak.



| Dish     |                            | Input +    |                                | Output +       | Annotation |
|----------|----------------------------|------------|--------------------------------|----------------|------------|
| decision |                            | Season     | How many guests                | Dish           |            |
| U        | season                     | guestCount | desiredDish                    |                |            |
|          | string                     | integer    | string                         |                |            |
| 1        | "Fall"                     | <= 8       | "Spare ribs"                   | -              |            |
| 2        | "Winter"                   | <= 8       | "Roast beef"                   | -              |            |
| 3        | "Spring"                   | <= 4       | "Dry Aged Gourmet Steak"       | -              |            |
| 4        | "Spring"                   | [5..8]     | "Steak"                        | Save money     |            |
| 5        | "Fall", "Winter", "Spring" | > 8        | "Stew"                         | Less effort    |            |
| 6        | "Summer"                   | -          | "Light Salad and a nice Steak" | Hey, why not!? |            |
| +        | -                          | -          | -                              | -              |            |

Figure 4: DMN decision table example.

### 2.2.3 YAWL

Yet Another Workflow Language (YAWL), it is a Business Process Management Systems (BPMS) that is used to edit the definition of process specifications and an engine to run them, so the process specifications have everything needed for process automation including control flow, resources and data.

In terms of functionalities, with YAWL we can design, execute and analyze processes, all this due to its modularity, consequently the entire environment can be substituted or modified by users or developers [17].

Figure 5 shows an YAWL example. In this example is created an workflow using YAWL regarding an application to a credit by a client from a bank.

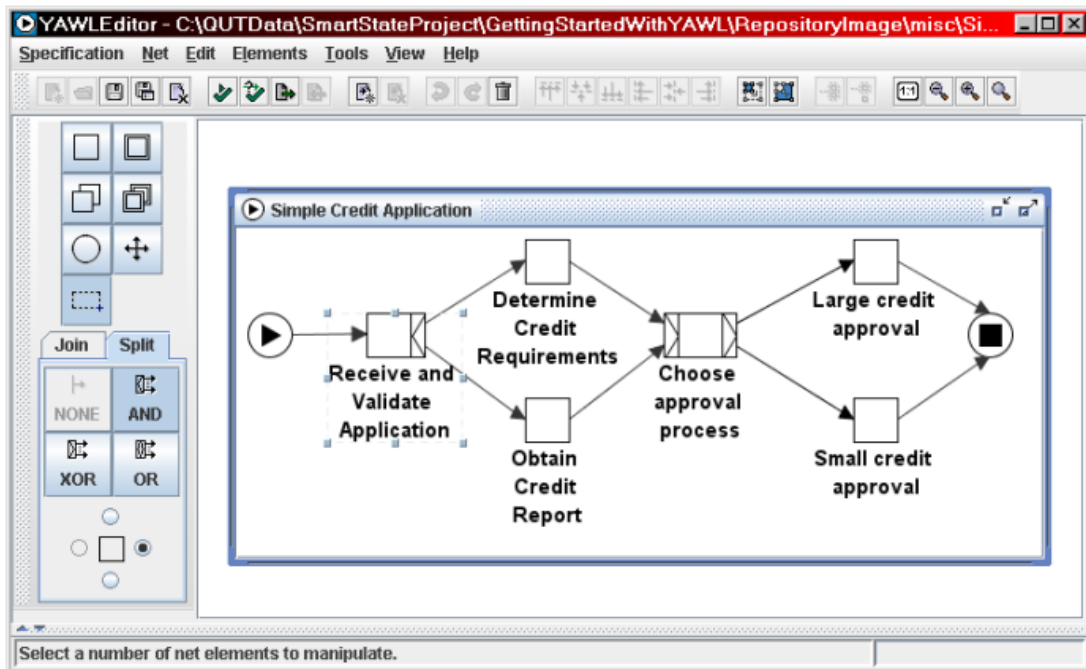


Figure 5: YAWL example extracted (from [18]).

## 2.2.4 CWL

Regarding Common Workflow Language (CWL) that is also a language used in workflow engines, it's a language that is much more hard to understand at the first site, and a bit confusing as well, probably because this one mainly uses the command line to build a workflow with scripts that may be too long. CWL is a way to describe command line tools and connect them together to create workflows, because CWL is a specification and not a specific piece of software. CWL tasks are isolated and the inputs and outputs given must be explicit. The benefit of explicitness are flexibility, portability, and scalability: tools and workflows described with CWL can transparently leverage technologies such as Docker. CWL is well suited for describing large-scale workflows in cluster, cloud and high performance computing environments where tasks are scheduled in parallel across many nodes [2]. CWL also is used to create automated pipelines as in [20].

## 2.2.5 BPMN vs DMN vs CWL vs YAWL

Figure 6 depicts a table comparing the various workflow languages described in this dissertation.



|  | CWL  | BPMN  | DMN   | YAWL   |
|--|--|---|---|--|
| Notation                                       | Common Workflow Language   | Business Process Model and Notation   | Decision Model and Notation   | Yet Another Workflow Language  |
| Code level                                     | Normal code (scripts)  | Low code  | Low code  | Low code   |
| Read/ Understand what's there to the naked eye | Not user friendly  | User friendly   | User friendly   | User friendly  |
| Usability                                      | Used in the command line through scripts.                          | Graphic interface that uses three objects:<br>- flow objects: activities, events, and gateways<br>- connection objects: sequence flows, message flows and association to represent relations<br>- swim lanes. | Uses decision tables with rules regarding a problem to be solved.   | A bit like BPMN, it uses blocks to code, but a little bit harder to understand. Has a higher learning curve. |
| Conclusion                                     | It was excluded due to being confusing and not easy to understand. | It was the language chosen over DMN despite being both very user friendly.  | It was not chosen, only because the graphical interface of BPMN is more perceptible than tables with text/expressions (what the tables mean). | Comparing with BPMN, BPMN is better due to its lower learning curve.   |

Figure 6: Languages comparison.

Therefore, the preferred language to be used in the solution development was BPMN, because it combines the fact that it has a lower learning curve and it is a user friendly language

## 2.3 Workflow Engines

A workflow engine consists on managing and tracking the state of tasks in a workflow, and decides which direction the workflow will take depending on the previous answer. Overall workflow engines simplify and automate some ordinary tasks that once were handmade.

In order to obtain a good and appropriate workflow engine it were studied some different engines, with the aim of comparing them and see which one was the most suitable for this project.

### **2.3.1 Camunda**

Camunda is a company based in Berlin with offices in the US, founded by Jakob Freund and Bernd Rücker. Their main product is a free open workflow and decision automation platform, which is called the Camunda Platform. It comes with tools for creating workflow and decision models, running developed models, and allowing users to perform workflow tasks assigned to them. It is developed in Java and released as an open source software around 2013.

The platform provides two different types of engines, a workflow engine that uses the high-level language BPMN and another that uses a decision engine that uses DMN as a high-level language. Both can be embedded in Java applications and with other languages via REST API [16].

### **2.3.2 BPMN.io**

BPMN.io ended up not being an engine itself, but it is an online platform that can be used to build BPMN and DMN diagrams, which uses an engine called Camunda. So the analysis ends up being short because it is an engine previously covered in one of the previous sections 2.3.1.

The engine that fulfill the most requirements of the problem at hand is Camunda. Some of the aspects that were in it's favour is the documentation available, which is more detailed than the one provided by BPMN.io, and because Camunda is easier to understand how it works in comparison with the other engines, however the documentation of Camunda was not much, so it took a bit longer to understand the basics. More time would be needed to explore some features in more detail.

## **2.4 Technologies**

### **2.4.1 JavaScript**

The main language used in this dissertation is JavaScript. Often shortened to JS, JavaScript appeared around 1995 in the USA, developed by Brendan Eich [5].

It's a compiled programming language with first-class functions, prototype-based, multi-paradigm scripting language, supports imperative, functional and object-oriented being this last one the most used. It's probably known as the language that's used for scripting for Web pages, but it's also used in non-browser environments. JavaScript runs on the client side of the web, which might be utilized to construct/program how the web pages act on the occurrence of an event.

Contrary to popular misconception, JavaScript is not “Interpreted Java”, in a nutshell, JavaScript is a dynamic scripting language supporting prototype based object construction. The fundamental syntax is intentionally identical to Java and C++ to decrease the number of new concepts required to learn the language. So the basic and well-known constructs are if statements, for and while loops, and switch and try-catch blocks have the same function as in these languages.

JavaScript’s dynamic capabilities are run-time object construction, variable parameter lists, function variables, dynamic script creation, object introspection, and source code recovery.

Also, JavaScript has a lot of existing libraries and frameworks that everyone can use, such as jQuery as a library and Node.js (that’s used in this dissertation and will be presented further ahead) for framework [4].

### **2.4.2 Node.js**

Node.js is an open source JavaScript server environment, created in 2009 by Ryan Dahl and in the current days is still one of the most used [6]. It runs on JavaScript Engine, executes the JavaScript code outside a web browser and was designed to build scalable applications.

With Node.js there are multiple ways to arrive to a solution.

### **2.4.3 Express**

Express is a minimal web application framework for Node.js that provides a solid feature set to web and mobile applications and APIs. According to it’s official GitHub repository the first version was deployed in 2010, being TJ Holowaychuk the original author [3].

Most of the time Express, which is used as a back-end component, is put along side with MongoDB as the database and a JavaScript as the front-end framework in popular development stacks.

### **2.4.4 Pug**

In alternative to HTML there is Pug, a template engine implemented with JavaScript for Node.js and browsers. Previously known as JADE, it is an easy-to-code template engine used to code HTML in a more readable fashion, that was heavily influenced by Haml. One upside to PUG is that it equips developers to code reusable HTML documents by pulling data dynamically from the API [9, 10].

## 2.4.5 Postman

Postman is an API platform for building and using APIs, created by Abhinav Asthana, Ankit Sobti and Abhijit Kane around 2012. In 2022, Postman was ranked #28 on the Forbes Cloud 100 list, and climbed up from #54 in the previous year [12].

Postman aims to simplify each step of the API life-cycle and streamline collaboration so someone can create better APIs faster. Additionally, it allows to easily store, catalog, and collaborate around all APIs artifacts on one central platform. Postman can store and manage API specifications, documentation, workflow recipes, test cases and results, metrics, and everything else related to APIs. It also includes a comprehensive set of tools that help accelerate the API life-cycle since the start, testing and documenting your APIs.

Postman aims to let users shift left their development practices, resulting in better-quality APIs, and creates collaboration between developer teams and API design teams. It has work-spaces to help to organize APIs work and collaborate across organization or across the world. There are three different kinds of Postman work-spaces for different needs: personal work-spaces, team work-spaces and public work-spaces.

Finally, Postman integrates with the most important tools in software development to enable API-first practices. The Postman platform is also extensible through the Postman API and through open source technologies [11].

## 2.4.6 REST API

An API is a set of definitions and protocols to building and integrating application software [1]. It's sometimes referred to as a contract between an information provider and an information user, establishing the content required from the consumer (the call) and the content required by the producer (the response). In other words, if you would like to interact with a computer or system to retrieve information or execute a function, an API helps you communicate what you wish to that system so it can perceive and fulfill the request [13].

The REST API, also known as RESTful API, is an application programming interface that conforms to the standards of REST architectural style and allows interaction with RESTful web services. [13]. REST stands for Representational State Transfer and was created by computer scientist Roy Fielding around 2000 [14].

Originally the term REST was referred to as a set of architectural principles, but nowadays is used in

the broadest sense to describe any simple web interface that uses XML (or YAML, JSON, or even plain text) and HTTP, without the additional abstractions of standards-based message exchange protocols.

What makes an API a RESTful API is:

- A client-server architecture with requests via HTTP;
- Stateless client-server communication;
- Uniform interface between the different parts so the information is transferred correctly;
- System based on layers to organize the different types of server that are involved in the information retrieval, while being invisible to the client;
- And, optionally, the possibility of sending executable code from the server side to the client when requested.

In contrast to other protocols like SOAP, REST APIs are faster and light weighted due to REST being a group of guidelines that are implemented when needed, making it perfect for Internet of Things and mobile app development.

# Chapter 3

## User Stories

This chapter presents the user stories that apply to this dissertation.

In Figure 7 is described the first use story, which is the definition of an high level language. To complete this use story it will be necessary to choose a high-level language more adequate for this work from amongst those that exist. The language has to support conditions, actions and, optionally could support graphical representation.

|  |   |              |                                      |               |   |
|--|---|--------------|--------------------------------------|---------------|---|
| <b>Id</b>  | RF-01   | <b>Title</b> | Definition of an high level language |               |   |
| <b>Description</b>   | The service must support at least one high level language for modulation of the form. |              |                                      |               |   |
| <b>Priority</b>  | High  | <b>Risk</b>  | High                                 | <b>Effort</b> | 9 |
| <b>Conditions of acceptance:</b>   |   |              |                                      |               |   |
| <ol style="list-style-type: none"> <li>1. Must support conditions</li> <li>2. Must support actions</li> <li>3. (Optional) Language must support graphic representation.</li> </ol> |   |              |                                      |               |   |

Figure 7: User Story - Language definition.

According to the user story shown in 8, this project should be capable of creating a form question by question using a graphical interface. Making it easy to use for users. In order to complete this use story, it is needed to search if exists any software that can produce that or if it is needed to code it from scratch, having in the end a solution where a user can code a form question by question.

|  |   |              |               |               |   |
|--|---|--------------|---------------|---------------|---|
| <b>Id</b>  | RF-02   | <b>Title</b> | Coding a form |               |   |
| <b>Description</b>   | As a user I want to code a form question by question. |              |               |               |   |
| <b>Priority</b>  | High  | <b>Risk</b>  | High          | <b>Effort</b> | 9 |
| <b>Conditions of acceptance:</b>   |   |              |               |               |   |
| <ul style="list-style-type: none"> <li>• Creation of the form in a graphic interface.</li> </ul> |   |              |               |               |   |

Figure 8: User Story - Coding a form.

The use story in Figure 9 describes that in this project it must be possible to export or save a coded form. So in order to complete it, the solution must export forms.

|                                  |  |              |               |               |   |
|----------------------------------|--|--------------|---------------|---------------|---|
| <b>Id</b>                        | RF-03  | <b>Title</b> | Export a form |               |   |
| <b>Description</b>               | As a user I want to export or save a coded form. |              |               |               |   |
| <b>Priority</b>                  | High   | <b>Risk</b>  | High          | <b>Effort</b> | 2 |
| <b>Conditions of acceptance:</b> |  |              |               |               |   |
|                                  |  |              |               |               |   |

Figure 9: User Story - Export a form.

The use story in Figure 10 describes that in this project it must be possible to load form that was previously exported to service. So in order to complete it, the solution doesn't need to be complex it only could be an action of importing files to a file system and doesn't need to have a graphic interface.

|   |  |              |             |               |   |
|---|--|--------------|-------------|---------------|---|
| <b>Id</b>   | RF-04  | <b>Title</b> | Load a form |               |   |
| <b>Description</b>  | As a user I must be able to load a file with a form that was previously exported to the service. |              |             |               |   |
| <b>Priority</b>   | Low  | <b>Risk</b>  | Low         | <b>Effort</b> | 1 |
| <b>Conditions of acceptance:</b>  |  |              |             |               |   |
| <ul style="list-style-type: none"> <li>• Doesn't need to have a graphic interface for this action.</li> <li>• The files can just be imported to a file system.</li> </ul> |  |              |             |               |   |

Figure 10: User Story - Load a form.

The use story in Figure 11 describes that in this project it must be possible to execute a form. With a given coded form by a user the solution must be able to execute the actions, associating to each step and producing an output. To complete this use story the solution must be able to proceed in the form by being able to execute each individual action, either being that action a call to an API or just a simple question.

|  |  |              |                |               |   |
|--|--|--------------|----------------|---------------|---|
| <b>Id</b>  | RF-05  | <b>Title</b> | Execute a form |               |   |
| <b>Description</b>   | A form must have implemented a set of actions that should be triggered according to the logic coded into the form. |              |                |               |   |
| <b>Priority</b>  | Low  | <b>Risk</b>  | Low            | <b>Effort</b> | 1 |
| <b>Conditions of acceptance:</b>   |  |              |                |               |   |
| <ul style="list-style-type: none"> <li>• Must execute actions(functions) implemented in the code</li> <li>• Exemples: Calling na API HTTP, sending a SMS, insert values into the database, etc.</li> <li>• The action might receive some input and generate na output.</li> <li>• The output can be used on decision "nodes".</li> </ul> |  |              |                |               |   |

Figure 11: User Story - Execute a form.

The use story in Figure 12 describes that in this project it must be possible to answer a form in an interactive way question by question. The questions should be presented one by one until the end of the form. Optionally the solution could have a feature where a user could select the form that he wants to answer. So in order to complete this use story the solution must able to process an answer given by the user when answering a form.

|   |   |              |               |               |   |
|---|---|--------------|---------------|---------------|---|
| <b>Id</b>   | RF-06   | <b>Title</b> | Answer a form |               |   |
| <b>Description</b>  | As a user I want to answer a form in an interactive way question by question. |              |               |               |   |
| <b>Priority</b>   | High  | <b>Risk</b>  | High          | <b>Effort</b> | 9 |
| <b>Conditions of acceptance:</b>  |   |              |               |               |   |
| <ul style="list-style-type: none"> <li>• I should be able to select the form that I want to answer (optional)</li> <li>• Should be presented one question at a time until the end of the form.</li> </ul> |   |              |               |               |   |

Figure 12: User Story - Answer a form.



## Chapter 4

# Development

This Chapter aims to explain the approach to this project, providing some context about each different stage, including the implementation itself.

In order to decide what language to use , it was made a small study regarding different Forms and platforms that can produce them. Afterwards was conducted a research related to the high level languages that can be used by an workflow engine. Being this last part concluded, it was necessary to choose which engine to use if already existed any or if it was needed to develop one from scratch, all this taking into consideration the high-level language that was decided to use (BPMN). The engine chosen to integrate the solution was Camunda. It's briefly explained in section [2.3.1](#) and will be explained in detail ahead in two sections: one section for the modeler [4.1](#) and one section for the engine [4.2](#).

Before exploring Camunda, it was necessary to make a conceptual approach of the solution, represented in Figure [13](#).

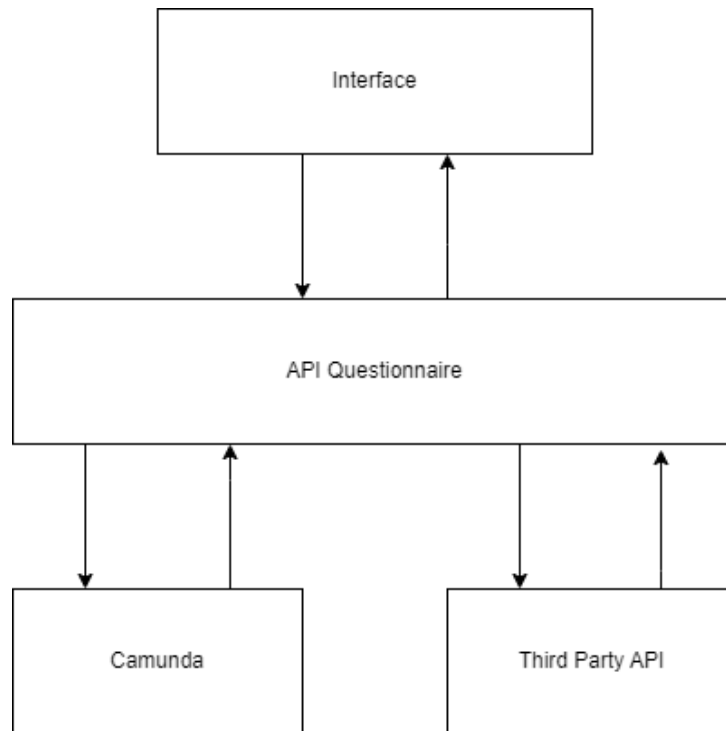


Figure 13: Solution's architecture.

Figure 13 represents the initial architecture for the solution, which includes four different components: the engine itself (Camunda), the client (Interface), the middleware (API Questionnaire) and the plugins (Third Party API). The engine will be responsible for uploading and exporting a form and process the form workflow, while the client will be a web interface with the questions for the user to respond. The API Questionnaire is the essence of this dissertation, because it is the “glue” of every component. The Third Party is a component that uses external APIs to do more specific questions or actions, but wasn't implemented.

An example of the main flow and its alternative flow in this solution is presented in Figure ??:

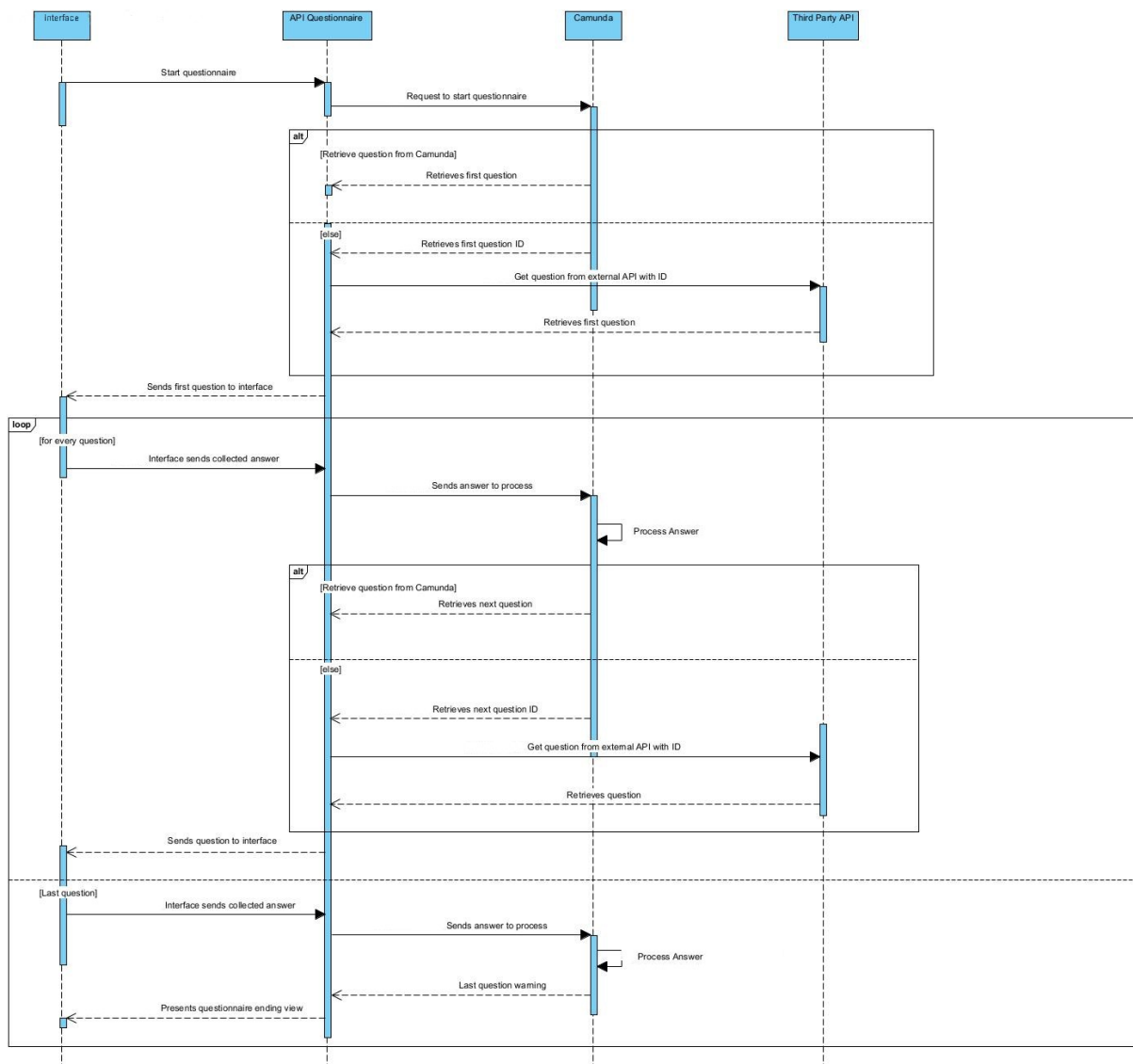


Figure 14: Solution's architecture flow.

Both flows begin with the start of the questionnaire, where the Interface starts the flow of the questionnaire by sending the request to the API Questionnaire component and receives the first question, which it sends to Camunda to be processed. The flow splits into two alternatives: the main one, where Camunda sends the answer back to API Questionnaire, which then sends the question back to the Interface. The alternative flow, Camunda sends the answer ID to the Third Party API, which sends the question back to API Questionnaire, that then sends the question back to the Interface.

Once it has been explained how the first question is obtained in both alternatives, they pass to a conditional loop that while they are not in the last question they will repeat the same process. While in the loop the Interface starts by sending the collected the last answer from the last question to the API Questionnaire, which sends it to Camunda, this one will process it. Once the answer is processed in

Camunda the main flow now will retrieve the question to the API Questionnaire and then sends it to the Interface. On the other hand the alternative flow will retrieve the question ID to the API Questionnaire and then will send it to the Third Party API, that will send the question to the API Questionnaire and afterwards will send it to the Interface.

Finally for both alternatives, if it reaches the last question, the Interface will send the collected answer to the API Questionnaire, which sends it to Camunda for it to process, and then sends an warning to the API Questionnaire that says it was the last question and the API Questionnaire sends the End of Questionnaire view.

## 4.1 Camunda Modeler

Prior to move forward to the implementation, it is necessary to understand how Camunda works and to address what features should be used. Camunda has, as already mentioned, a modulation component. This component exists in two formats, the first is the cloud or online format that can be used to build BPMN or DMN diagrams in a browser, which is the same concept as BPMN.io referred in Section 2.3.2. The other format, which is the one used in this dissertation, consists of the same thing that the other online format does, except that it is a program that runs locally. The only difference resides in the fact that it goes from a browser to an application.

The modeler is in charge of creating diagrams and has at its disposal all the existing elements in the BPMN and DMN language, and also has a very user friendly interface, where the user can drag the components to add to the diagram to the central area. All this contributes to making the tool as complete as possible and easy to use.



Figure 15: Camunda Modeler.

In Figure 15, the black box shows the BPMN elements that can be used by dragging them to the red box, the “Working Area” where the diagram will be. Finally, in the blue box is the properties panel where the diagram can be customized from changing the name of each component or even the whole diagram, to defining the condition expressions.

In Figure 16, there's an example of what a diagram looks like in Camunda Modeler.

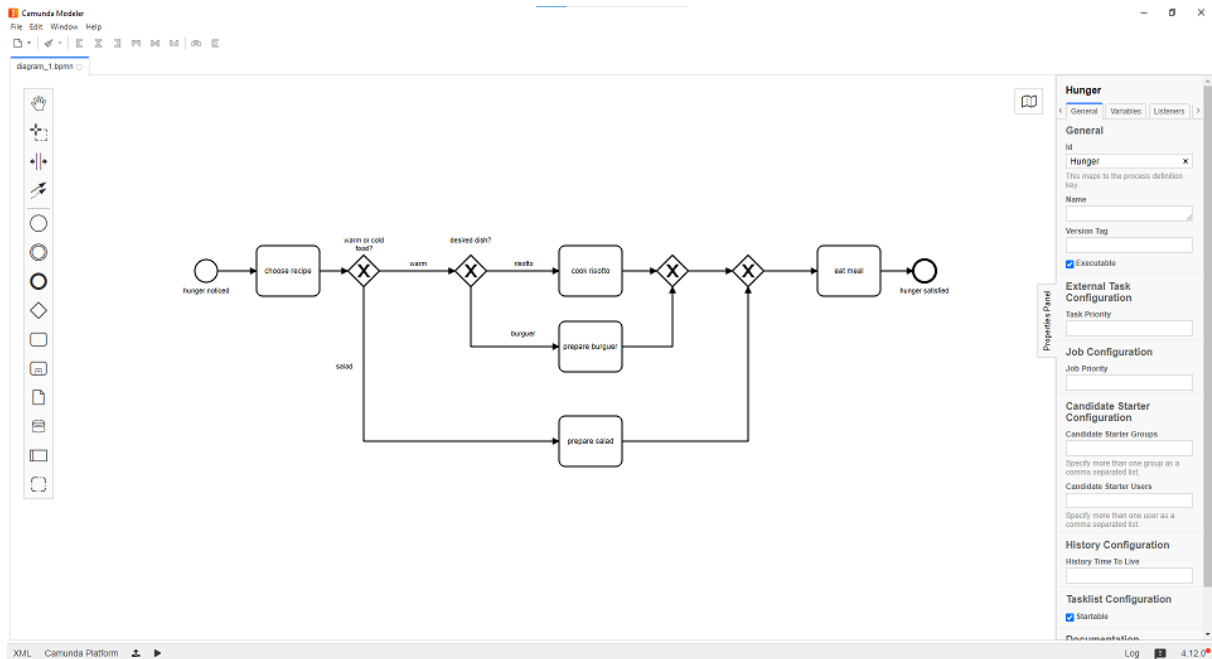


Figure 16: Camunda Modeler example.

## 4.2 Camunda Engine

Postman, presented in Section 2.4.5, was used to verify the available REST API from Camunda and its endpoints.

Next follow some of the calls that are used and others relevant as well in the solution:

- **[POST] /deployment/create**: Creates a deployment.
- **[GET] /process-instance**: Retrieves the list of process instances running, can be made queries for process instances that fulfill given parameters.
- **[DEL] /process-instance/id**: Deletes a process instance.
- **[GET] /task**: Retrieves the list of tasks running, can be made queries for process instances that fulfill given parameters.

- **[POST] /process-definition/key/key/start:** Instantiates a given process definition, starts the latest version of the process definition. The key is the key of the process definition. Process variables and business key may be supplied in the request body.
- **[GET] /task?processInstanceId=id:** Retrieves the list of tasks running of a given process instance.
- **[POST] /task/taskID/complete:** Completes a task and updates process variables. It is required to send the variables needed in the body.
- **[GET] /history/variable-instance?processInstanceId=id:** Retrieves the history of answers of the given process instance.

### 4.3 Implementation

With the language settled and the engine studied, it was time to start the development of the “part that glues” everything together, the component “Questionnaire” presented in Chapter 4 more precisely in Figure 13, making the whole process more autonomous.

To start off, it was created a project using Express. Express organizes the project as shown in Figure 17.

```

- \---project/
  |
  | app.js
  | package-lock.json
  | package.json
  |
  | +---bin
  | |     www
  | |
  | | +---public
  | | |
  | | | +---javascripts
  | | | |     jquery.js
  | | | |     main.js
  | | | |
  | | | \---stylesheets
  | | |     style.css
  | | |
  | | | +---routes
  | | | |     index.js
  | | | |     instances.js
  | | | |     question.js
  | | | |
  | | | \---views/
  | | | | end.pug
  | | | | error.pug
  | | | | index.pug
  | | | | instances.pug
  | | | | layout.pug
  | | | | question.pug
  | | | | start.pug
  | | | | tasks.pug
  | | |
  | |
  |

```

Figure 17: Solution structure.

The solution follows the MVC methodology (Model View Controller), which can be seen in Figure 17, there is the “routes” folder, where the application routes are coded, and in the “views” folder are the interfaces made in Pug that are presented to the user. After that, there is also the “public” folder which contains two other folders, the “javascripts” folder where are the support files in which are coded some actions, using AJAX and a JavaScript library, JQuery, and the other folder named “stylesheets”, which has the CSS customization files. There is also the “bin” folder where there are some dependencies as well as the HTTP ports definition, and the “node\_modules” folder where there are all the temporary files and Node.js related files, such as packages, etc. Finally, the main file “app.js” in which most of the structure is defined.

In the “views” folder there are eight files regarding the multiple views. “end.pug” represents the view when a form ends; “error.pug” the view when any error happens; “index.pug” is the view of the main page, “instances.pug” shows the list of process instances running and each one can be deleted in the delete

button, “layout.pug” this one defines the base layout to all the other views, “question.pug” represents the view of a question and has a text box for the respective answer as well as the submit button, “start.pug” is the view that shows the name of the form that is about to be started and the button to start it and finally “tasks.pug” shows the list of tasks running.

The “routes” folder has three files, “index.js”, “instances.js” and “question.js” where in each one has code related to some part of the solution. The first one “index.js” has to 3 routes, one being the home page render, that renders the view of the home page. Then is the route to start a form, presented in Figure 26. This route only redirects to a page where the actual form can be started by clicking the button start, but this redirect works for every single form that exists. Being the route “localhost:3000/start/form/:key”, it’s possible to choose which form to start simply by changing the last part of the url that is the “:key” with the key of the process definition in question.

```
/* Start a Form */
router.get('/start/questionnaire/:key', function(req, res) {
  res.render('start', { key: req.params.key});
});
```

Figure 18: Start form.

Finally in Figure 19 the route that retrieves the tasks running and then renders the information gathered in the request into the view, in this case “task.pug”. Being the route “localhost:3000/tasks”.

```
/* GET tasks */
router.get('/tasks', function(req, res) {
  axios.get('http://localhost:8080/engine-rest/task')
    .then(dados => {
      res.render('tasks', {title: 'Tasks Running', tasks: dados.data});
    })
    .catch(e => res.render('error', {error:e}))
  console.log(req.body);
});
```

Figure 19: GET tasks.

Next file, “instances.js” has three routes the first one, shown in Figure 20, similar to the last one, but instead of retrieving the tasks, will retrieve the process instances that are currently running and also renders the information gathered in the request into the view, now “instances.pug”. Being the route “localhost:3000/instances”.



```

/* GET process instances */
router.get('/', function(req, res) {
  axios.get('http://localhost:8080/engine-rest/process-instance')
    .then(dados => {
      res.render('instances', {title: 'Instances Running', instances: dados.data});
    })
    .catch(e => res.render('error', {error:e}))
    console.log(req.body);
});

```

Figure 20: GET process instances.

Proceeding to the next route, in Figure 21, that is elimination of a process instance the route is like this “localhost:3000/instances/:id” (the id is the process instance id).

```

// DELETE process instance
router.delete('/:id', async function(req, res) {
  await axios.delete('http://localhost:8080/engine-rest/process-instance/'+req.params.id)
    .then(dados => {
      console.log(dados.data)}
    )
    .catch(e => res.render('error', {error:e}))
    console.log(req.body);
});

```

Figure 21: DELETE process instances.

In Figure 22 is the last route of this file is the one that actually starts a form, it creates a new process instance of given process definition key and after creating that process instance it fetches the first question of the form. This route is used in an AJAX function, that is used on the definition of the action of the start button. Also in this route is stored on the cookies the process definition id, because it will be useful in the future. The route is “localhost:3000/instances/create/:key”, where the “:key” is the key of the process definition in question.

```

//CREATE process instance
router.get('/create/:key', function(req, res) {
  params = {
    businessKey: 'myBusinessKey'
  }
  axios.post('http://localhost:8080/engine-rest/process-definition/key/' + req.params.key + '/start', params)
    .then(response => {
      id = response.data.id
      axios.get('http://localhost:8080/engine-rest/task?processInstanceId='+ id)
        .then(task =>{
          var taskId = task.data[0].id
          res.cookie('processInstanceId',id)
          res.redirect("/question/" + taskId)
        })
        .catch(e => {
          console.log(e)
        })
    })
    .catch(e => {
      console.log(e)
    })
});

```

Figure 22: Create process instance.

Moving on to the last file, “question.js” this one has two routes the first one the route that retrieves the questions, in Figure 23. It retrieves one question at a time, where the question is in the task field “name” and then renders the question in the view “question.pug”. Being the route “localhost:3000/question/:id”, where “:id” is the id of the task in question.

```

/* Question */
router.get('/:id', function(req, res) {
  axios.get('http://localhost:8080/engine-rest/task/' + req.params.id)
    .then(dados => {
      res.render('question', { id: dados.data.id, question:dados.data.name});
    })
    .catch(e => res.render('error', {error:e}))
    console.log(req.body);
});

```

Figure 23: GET question.

To finalize the last route, in Figure 24, this route is in charge of processing the answer provided by the user by completing the task with a request. To complete the task, when the request is sent, on the body of the request it has to have the variables that has the answer in it, and the variable must match with the variable used in the Camunda Modeler to do the split decision. Here is also checked if there are still any tasks running after completing one, and if there aren't any it will be redirected to a page that informs the end of the form, as well as the cookies that had used to store the process instance id will be cleaned. Otherwise the workflow will continue with the next question.

```

/* Answer */
router.post('/:id', function(req, res) {
  params = {
    variables: {
      response: { value:req.body.answer, type:"String"}
    }
  }
  axios.post('http://localhost:8080/engine-rest/task/' + req.params.id + '/complete',params)
    .then(response => {
      id = req.cookies.processInstanceId
      axios.get("http://localhost:8080/engine-rest/task?processInstanceId="+ id)
        .then(task =>{
          if(task.data.length!=0){
            var taskId = task.data[0].id
            res.redirect("/question/" + taskId)}
          else{
            res.clearCookie
            res.render('end')
          }
        })
        .catch(e => {
          console.log(e)
        });
    })
    .catch(e => {
      console.log(e)
    });
});

```

Figure 24: Answer question (Complete task).

## Chapter 5

### Proof of Concept

The main application of this dissertation and also one of its main objectives is the integration of the solution in the medical field, namely for the collection of PREMs and PROMs. The application is expected to have two major impacts. Doctors and other healthcare professionals will get more information about patients because the form completion rate will increase. Moreover, they will get insights on their satisfaction with care. In this way, doctors will be able to provide better care to their patients and will see their satisfaction with the care provided evaluated. These two variants represent the two types of forms that are used, the PROM and the PREM.

Another benefit for doctors is that they no longer receive some information that would be useless in some cases, because the workflow eliminates some questions that are of no interest in some cases and presents the most appropriate and precise ones. It should be noted that all this may vary from form to form and from user to user. On the other hand, the users get shorter forms that are more direct to what the doctors need to know, which saves them from answering long forms that would cause them to lose interest and patience in filling them out, so it ends up being a win-win. This helps everyone improve themselves a lot more.

The second impact on the other ecosystem focuses more on the area of who makes the forms. In companies that apply PROMs and PREMs or other types of medical forms, the work of developing them is on the developers' side, which can end up overloading them. On the other hand, the team that is in contact with hospitals, for example, ends up bringing the form specifications for the developers' team to develop. The benefit of the application of this solution is to relieve the developers so that they are only needed in the last resort or in something more technical and make the whole process of assembling forms more autonomous, and on the other hand that the team that is in contact with hospitals, for example, assembles the questionnaire right away because the solution is made in a way that is simple for people to assemble forms. In other words, the forms are ready on the spot and the developers are not overloaded.

Another possible application could be at the enterprise level both internally and externally. Forms that

measure the happiness or problems that employees of a certain company might have, or a form that deals with customer support or even a form that is presented to a customer after a purchase to find out where the company can improve. These examples all end up being based on the same thing which are forms.

It should be noted that perhaps the examples that would make the most sense would be the complaint and customer support forms, as it saves time for those who are answering by ignoring questions that are not relevant to that particular case and focus on the problems that matter as much to the company as to the customer.

Finally a possible application that might already take more work to put into action. It would be perhaps in the automation of machinery in large production lines or the automation of different types of machines/-electronics, where the workflow would be all the actions of the machine following a certain order and then going into a loop, as for example in the assembly of cars. This application would lead the company to savings despite having an initial investment in the machines if they didn't already have them. On the other hand, it would take away some jobs, provided the company still has manual production lines, otherwise it wouldn't matter.

To better explain the implementation of the developed solution, an example of a possible flow from the view of the user, is presented. To start, it is necessary to have running in the background the Camunda engine and then start the solution. Starting of in the main page of the solution. It's a simple view, it only has the name of the dissertation, the name of the author, and the supervisors and is a static page. As it can be seen in Figure 25.

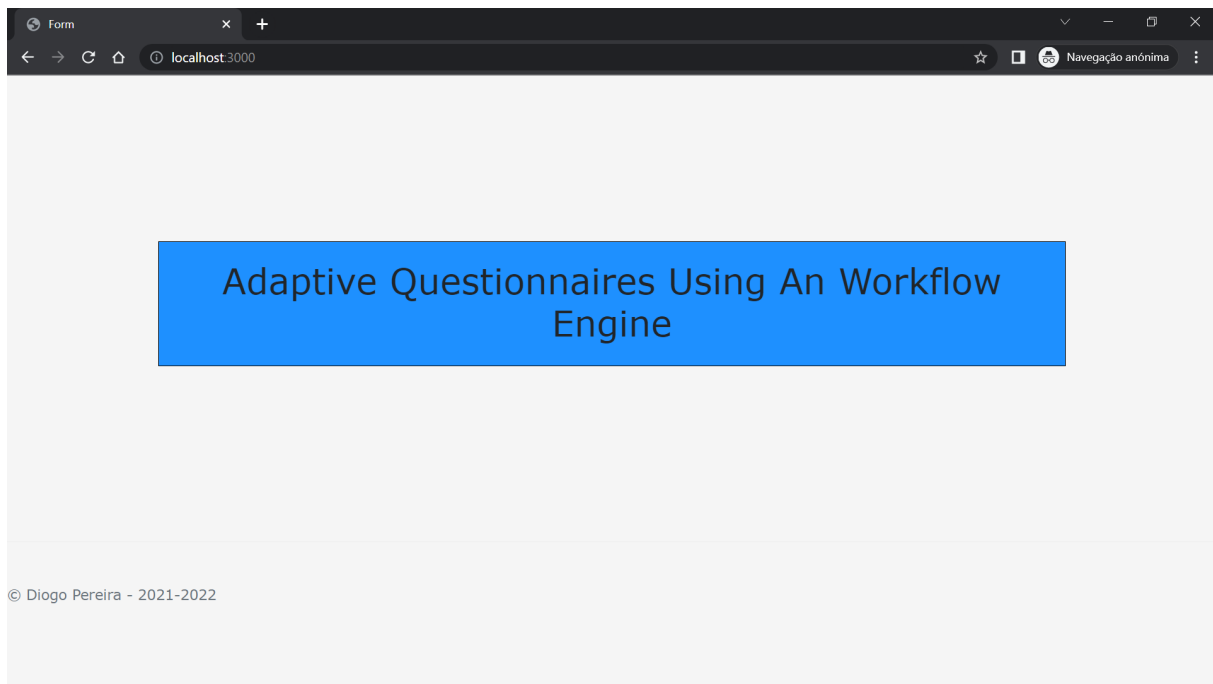


Figure 25: Home page.

So to start a new form, the only thing needed is to set the url to: <https://localhost:3000/start/nameOfForm> . Where it says nameOfForm it's supposed to put the form key that is defined in Camunda, for example <https://localhost:3000/start/Teste1> . This was implemented in a way that could be made different requests for different forms using the same call. After running that url, it should appear a page like the one in Figure 26. That will show the name of the form and a button to start it.

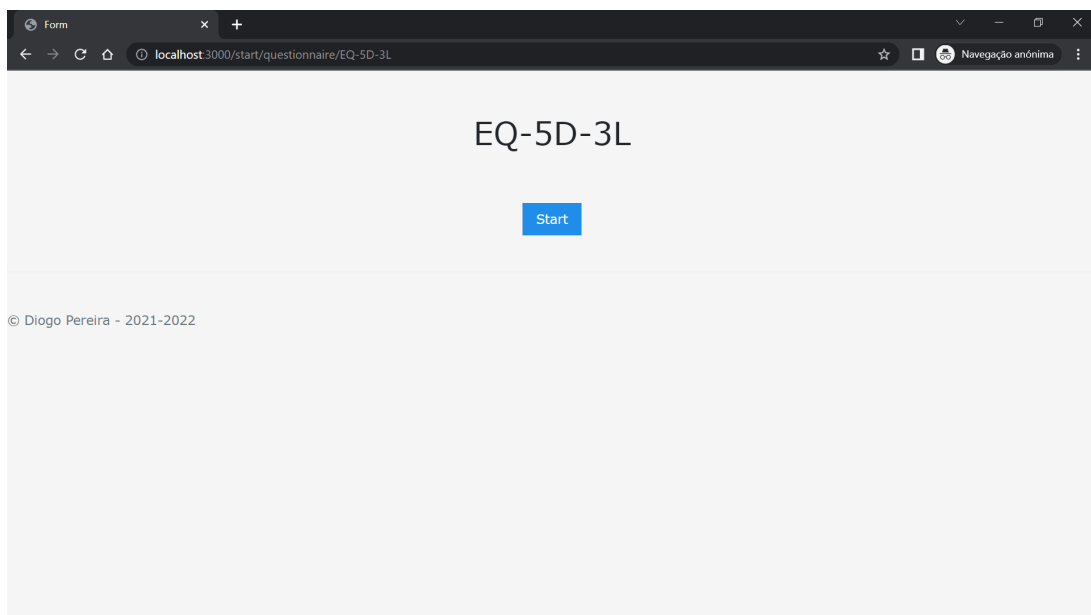
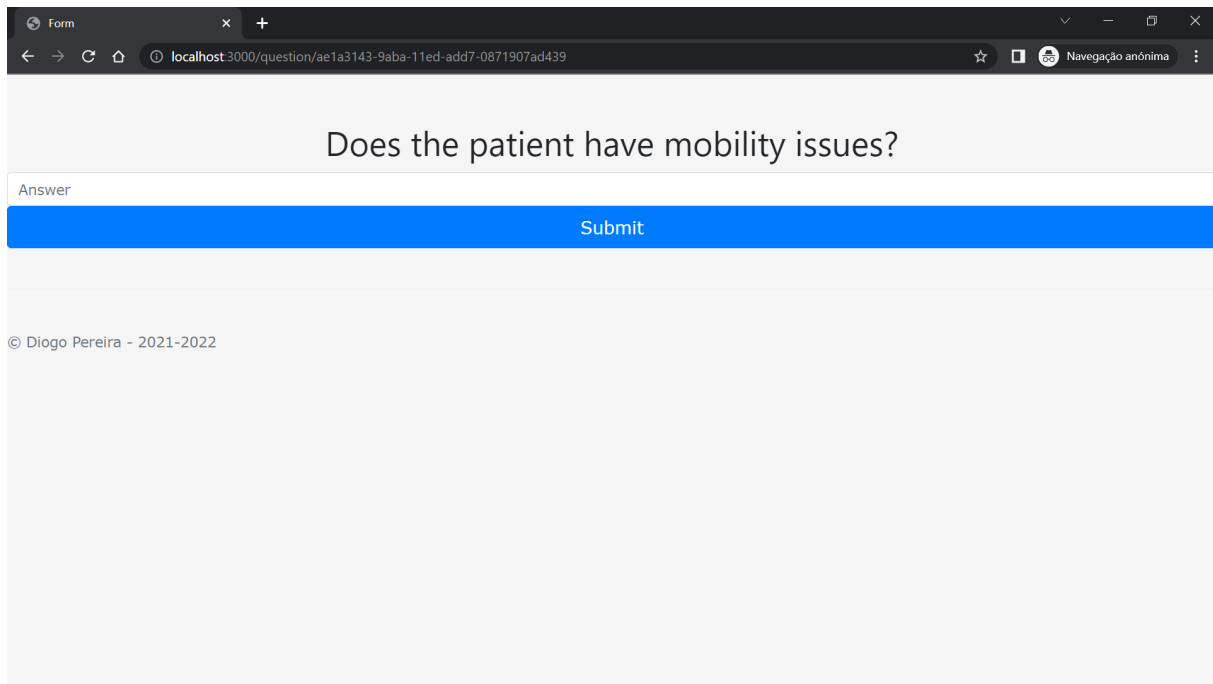


Figure 26: Start a form.

In that page, the only thing available to do is to start the form by pressing the Start button. After the button is pressed, it will be redirected to another page that will show the first question of the form with the respective answer box, as shown in Figure 27. By pressing the button what is happening is that the system is making a request to Camunda to start the form and at the same time retrieving the task that is running, that has the question in the field “name”. The redirection will be to a page where the url will be <https://localhost:3000/question/taskID> where taskID is the Id of the task in question.



The screenshot shows a web browser window with the title 'Form'. The address bar displays the URL 'localhost:3000/question/ae1a3143-9aba-11ed-add7-0871907ad439'. The page content includes the question 'Does the patient have mobility issues?' centered in a light gray box. Below the question is a text input field labeled 'Answer'. A prominent blue button with the text 'Submit' is positioned below the input field. At the bottom left of the page, there is a copyright notice: '© Diogo Pereira - 2021-2022'. The browser's status bar at the bottom indicates 'Navegação anónima'.

Figure 27: First question example.

When the user fills the answer box and then presses submit, the solution will process the answer and retrieve the next question presenting it. So a more detailed explanation of what’s happening is, when the user submits the answer, the Questionnaire will send the answer to Camunda and the answer will be processed. After that, the Questionnaire will have to make another request to get the next question and present it, as is in Figure 28 .

Form

localhost:3000/question/8b5e491c-9aba-11ed-add7-0871907ad439

Navegação anónima

## Does the patient have self care problems?

Answer

Submit

© Diogo Pereira - 2021-2022

Figure 28: Question example.

This will continue until the flow reaches it's final question. When the user submits it's final question, the solution will present a page saying "End of Form!" representing the end of the form, basically what's happening is that it is being verified if there is still any task of that given process instance running. If there is no task running, it is presented the "End of form" page, as shown in Figure 29, otherwise the flow will be resumed with the next question.

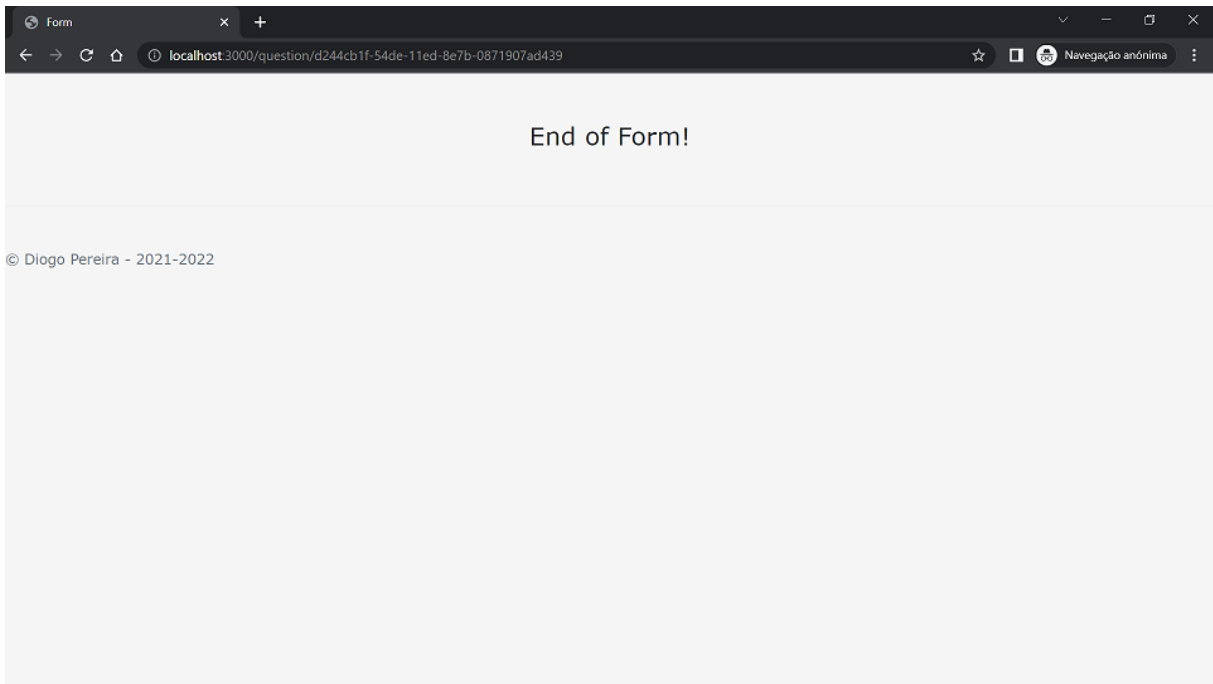


Figure 29: End of a form.

Apart from this steps there is as well some features to the company/hospital, that is a page where can be seen all the instances that are currently running as shown in Figure 30. It is possible to delete an instance using the delete button, also present in the image.

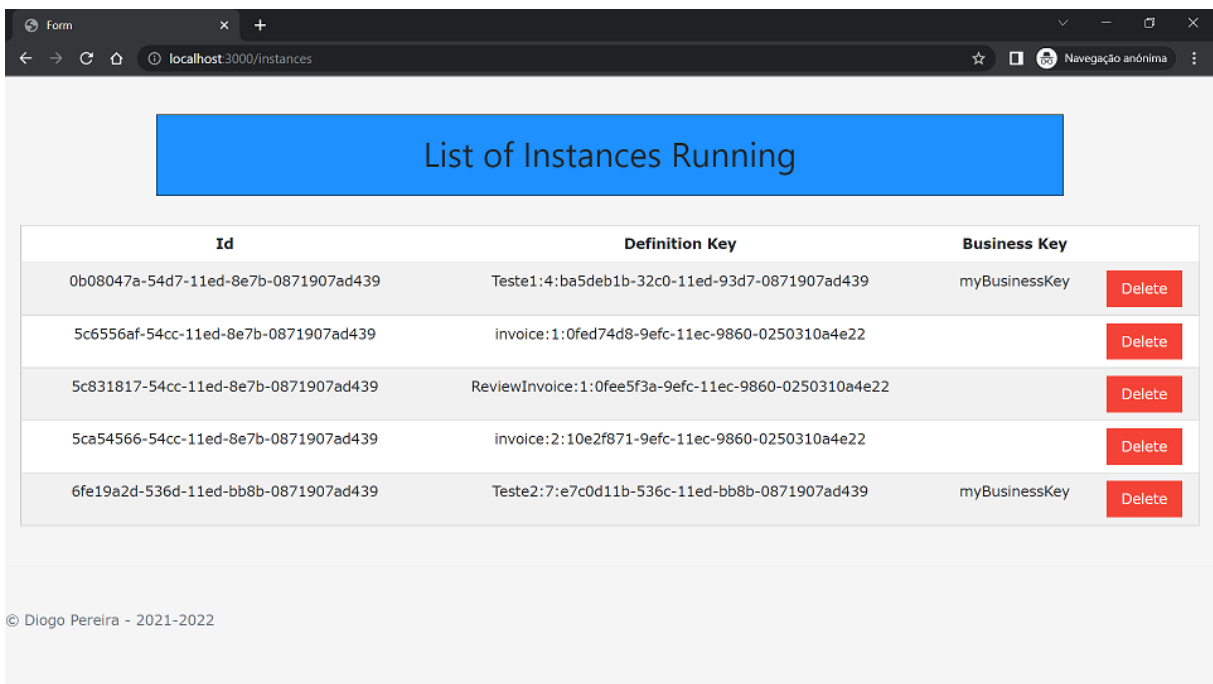


Figure 30: List of instances running.

And another page is where can be seen all the tasks that are currently running, as seen in Figure 31.



| Id                                   | Name                                   | Process Instance Id                  | Description                |
|--------------------------------------|--|--------------------------------------|----------------------------|
| 0b082b8d-54d7-11ed-8e7b-0871907ad439 | Does patient has a Surgery Date on DB? | 0b08047a-54d7-11ed-8e7b-0871907ad439 |                            |
| 5c83b467-54cc-11ed-8e7b-0871907ad439 | Assign Reviewer                        | 5c831817-54cc-11ed-8e7b-0871907ad439 |                            |
| 5cb8cda0-54cc-11ed-8e7b-0871907ad439 | Prepare Bank Transfer                  | 5ca54566-54cc-11ed-8e7b-0871907ad439 | Prepare the bank transfer. |
| 6fe1e851-536d-11ed-bb8b-0871907ad439 | Does patient has a Surgery Date on DB? | 6fe19a2d-536d-11ed-bb8b-0871907ad439 |                            |
| 6fe20f67-536d-11ed-bb8b-0871907ad439 | Patient Timeline                       | 6fe19a2d-536d-11ed-bb8b-0871907ad439 |                            |

© Diogo Pereira - 2021-2022

Figure 31: List of tasks running.

As it will be referred in Section 6.2 could be developed a new page that could show the statistics of the questionnaires and the results obtained.

## **Chapter 6**

# **Conclusions and future work**

### **6.1 Conclusions**

The aim of this dissertation was to provide a tool to facilitate the application of adaptive forms of PROMs and PREMs and, consequently, promote the completion of these questionnaires by patients. Besides increasing the response rate of these forms, the software module developed aims to decrease the workload of the company that applies these forms and collect the answers. This dissertation had as main goal to develop a solution based on a workflow engine to create Computerized Adaptive Tests (CATs) in an intuitive way. To achieve this goal, it was studied what a workflow engine is, the most appropriate high-level language to be used, techniques for its implementation and, subsequently, implemented a service to generate a workflow engine that uses a low-level coding language, allowing people without technical knowledge to also use it to build forms, such as PROMs or PREMs. To do so, a study was made regarding the existing types of medical forms, as well as some platforms for adaptive forms development. Next, a study was conducted regarding high-level languages to select the most appropriate to use and what engine was the most suitable for the given language chosen before. User stories were created to describe how the solution would work. To this end, a conceptual approach to the problem was also conducted to structure the solution, which resulted in the creation of its architecture. After that, the engine was studied to define which functionalities provided should be used. The implementation was done mainly in JavaScript, with the support of Express.js to organize the structure of the solution. Then, the results obtained in this project were presented and discussed, accompanied by their possible applications.

From the work developed, it resulted an application in which it is possible to run one or multiple forms, individually or simultaneously, where the user answers the questions presented and follows a flow of questions until its end. This has several components, one of them being the Camunda Modeler, in which it is possible to load or create BPMN diagrams, which are deployed for later execution, already with the help of Camunda's Engine. When the forms start being executed, the questions are presented to

the users, one by one, so that they can answer each one individually, always following the workflow until, eventually, reaching its end. Two features were also developed, accessible only to those who create and manage the forms, which allow them to access to the list of process instances and the list of tasks that are running.

The solution developed fulfilled the defined objectives. However, some features could have been implemented in an alternative way. Instead of the questions being stored in the task field “name” in Camunda, an identifier could have been attributed in that same field and would retrieve the actual question from another API with that given identifier. The main page could be also be changed and present some kind of menu to choose from the available forms to start one, instead of being via url.

This solution can be used by Promptly Health in their work environment or a just a fraction of the project integrating it with their own product.

This final solution can be applied in different environments, for example, by a hospital or small clinic, to collect PROMs or PREMs to know how the patients are reacting and their satisfaction towards some kind of treatment.

## **6.2 Prospect for future work**

This dissertation still has a lot of room to evolve. To this end, here are some points that can be developed or improved in a future work.

- integration of DMN in BPMN;
- communication with a third party API;
- integration with other softwares, like hospital management softwares;
- different types of answer in the forms, such as selection box, multiple answer, multiple choice, etc;
- improvements in the architecture;
- modules to present statistics and the results of the forms;
- more complete home page, with the option of being able to choose a form to start with, from all the possible ones.

## Bibliography

- [1] Api - wikipedia. URL <https://en.wikipedia.org/wiki/API>.
- [2] 1.1. quick start – common workflow language user guide 0.1 documentation. URL [https://www.commonwl.org/user\\_guide/introduction/quick-start.html](https://www.commonwl.org/user_guide/introduction/quick-start.html).
- [3] Express.js - wikipedia. URL <https://en.wikipedia.org/wiki/Express.js>.
- [4] About javascript - javascript | mdn. URL [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript).
- [5] Javascript – wikipédia, a enciclopédia livre. URL <https://pt.wikipedia.org/wiki/JavaScript>.
- [6] Node.js - wikipedia. URL <https://en.wikipedia.org/wiki/Node.js>.
- [7] Everything you need to know about proms and prems, . URL <https://www.remecare.eu/blog/everything-you-need-to-know-about-proms-and-prems>.
- [8] How do prems and proms work together? - cemplicity, . URL <https://www.cemplicity.com/blog/how-do-prems-and-proms-work-together/>.
- [9] What is pug syntax?, . URL <https://www.educative.io/answers/what-is-pug-syntax>.
- [10] pugjs/pug: Pug – robust, elegant, feature rich template engine for node.js, . URL <https://github.com/pugjs/pug#readme>.
- [11] Postman api platform, . URL <https://www.postman.com/product/what-is-postman/>.
- [12] Postman (software) - wikipedia, . URL [https://en.wikipedia.org/wiki/Postman\\_\(software\)](https://en.wikipedia.org/wiki/Postman_(software)).

- [13] What is a rest api?, . URL <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
- [14] Rest – wikipédia, a enciclopédia livre, . URL <https://pt.wikipedia.org/wiki/REST>.
- [15] Surveymonkey – wikipédia, a enciclopédia livre. URL <https://pt.wikipedia.org/wiki/SurveyMonkey>.
- [16] Camunda - wikipedia. URL <https://en.wikipedia.org/wiki/Camunda>.
- [17] Michael Adams, Andreas V. Hense, and Arthur H.M. ter Hofstede. Yawl: An open source business process management system from science for science. *SoftwareX*, 12, 7 2020. ISSN 23527110. doi: 10.1016/j.softx.2020.100576.
- [18] Lindsay Bradford and Marlon Dumas. Getting started with yawl, 2007. URL [https://sourceforge.net/docman/?group\\_id=114611](https://sourceforge.net/docman/?group_id=114611).
- [19] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards and Interfaces*, 34:124–134, 1 2012. ISSN 09205489. doi: 10.1016/j.csi.2011.06.002.
- [20] Pasi K Korhonen, Ross S Hall, Neil D Young, and Robin B Gasser. Common workflow language (cwl)-based software pipeline for de novo genome assembly from long- and short-read data. *GigaScience*. doi: 10.1093/gigascience/giz014/5368071. URL <https://academic.oup.com/gigascience/advance-article-abstract/doi/10.1093/gigascience/giz014/5368071>.



