ProntApp: A mobile question answering application for medicines

Luís Miguel Mancilha de Almeida Martins de Jesus

Thesis to obtain the Master of Science Degree in Information Systems and Software Engineering

Supervisors: Prof. Helena Isabel de Jesus Galhardas
            Prof. Paulo Jorge Pires Ferreira

Examination Committee
Chairperson: Prof. Miguel Nuno Dias Alves Pupo Correia
Supervisor: Prof. Helena Isabel de Jesus Galhardas
Member of the Committee: Prof. Ricardo Jorge Feliciano Lopes Pereira

November 2015
Dedicated to my parents Veríssimo and Luísa, my sister Maria and her husband Marco, my nephew Afonso, my girlfriend Elisabete and my grandparents Veríssimo, Felismina, Henrique and Maria José
Acknowledgments

Firstly, I would like to express my sincere gratitude to my Professors Helena Galhardas and Paulo Ferreira for their continuous patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I thank my fellow colleagues Rodrigo, Pedro Costa, Pedro Matos, Filipe and Rui that accompanied me for all these years, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last years.

Also, I would like to thank my family: my parents Veríssimo and Luísa, my sister Maria and her husband Marco, my nephew Afonso and my grandparents Veríssimo, Felismina, Henrique and Maria José for supporting me spiritually throughout writing this thesis and my life in general.

Last but not the least, I would like to thank my girlfriend Elisabete for never allowing me to give up and always helping me and standing by my side when I needed, and for being my best friend.
**Resumo**

Há cada vez mais informação sobre medicamentos disponível online. Essa informação é acedida não só por pessoas com formação no ramo da medicina mas também por pacientes, e pelo público no geral. No entanto, com tanta informação disponível, tornou-se cada vez mais difícil procurar pela informação que necessitamos numa situação específica. O ProntApp é uma aplicação móvel para Android e iPhone com o intuito de colmatar esta falha com um sistema de question-answering em língua Portuguesa para medicamentos. Neste documento descrevemos as principais funcionalidades do ProntApp, o estado da arte em termos de sistemas de question-answering no ramo da medicina, sistemas web e aplicações móveis deste ramo, a sua arquitectura e decisões de implementação. Por fim, descrevemos a avaliação experimental que foi feita para validar a aplicação ProntApp, bem como os resultados obtidos, e algumas conclusões acerca do trabalho.

**Palavras-chave:** aplicação móvel, question-answering system, medicamentos, reconhecimento de voz, android, iphone
Abstract

We have been witnessing an increased amount of medical data that is available online. Such data is accessed not only by medical staff but also by patients, and the public in general. However, with that much data, it has become increasingly difficult to search for the correct answers we are looking for in a specific situation. ProntApp is an Android and iPhone mobile application that aims at fulfilling this gap with a question-answering system in Portuguese for medicines. In this document, we describe the main features of ProntApp, we present the current state-of-the-art of the medical domain question-answering systems, web-based systems and mobile applications for medicines, its architecture and the implementation decisions. Finally, we describe the experimental evaluation that we conducted to validate ProntApp and the results obtained, and draw some final conclusions.

Keywords: mobile application, question-answering system, medicines, voice recognition, android, iphone
# Contents

List of Tables .............................................................................................................. xv
List of Figures ............................................................................................................... xviii

1 Introduction .............................................................................................................. 1
  1.1 Objective ............................................................................................................. 2
  1.2 Requirements ..................................................................................................... 2
  1.3 Proposed Solution ............................................................................................... 3
  1.4 Contributions ...................................................................................................... 3
  1.5 Typical Architectures of Mobile Applications .................................................... 4
    1.5.1 Online Architecture ..................................................................................... 4
    1.5.2 Offline Architecture ..................................................................................... 5
  1.6 Thesis Outline ..................................................................................................... 6

2 Related Work ............................................................................................................. 7
  2.1 MedicineAsk ....................................................................................................... 7
    2.1.1 Information Extraction Module .................................................................... 7
    2.1.2 Natural Language Interface Module ............................................................ 9
  2.2 MEANS .............................................................................................................. 9
  2.3 Web-Based Medical Systems .............................................................................. 11
  2.4 Medical Mobile Applications ............................................................................ 11
    2.4.1 Epocrates’ Mobile Application ................................................................... 12
    2.4.2 Medscape’s Mobile Application ................................................................. 12
    2.4.3 Drugs.com’s Mobile Application ................................................................ 13
    2.4.4 eMed.pt ...................................................................................................... 13
  2.5 Voice Recognition Systems ............................................................................... 14
  2.6 Comparison of question-answering systems ....................................................... 14
  2.7 Comparison of voice recognition systems .......................................................... 15
  2.8 Comparison of existing medical mobile applications ....................................... 16
  2.9 Discussion ......................................................................................................... 16

3 ProntAPP: Architecture .......................................................................................... 19
  3.1 Features ............................................................................................................. 19
List of Tables

2.1 Comparison between MedicineAsk and MEANS. ......................... 15
2.2 Comparison between Google’s voice recognition system, CMU Sphinx and Nuance Dragon. 15
2.3 Comparison between Epocrates, Medscape, Drugs.com and eMed.pt. ..................... 16

5.1 Results of Natural Language Server Load Tests (5 simultaneous requests) .............. 56
5.2 Results of Keyword Server Load Tests (5 simultaneous requests) ........................ 56
5.3 Results of Natural Language Server Load Tests (10 simultaneous requests) ............ 56
5.4 Results of Keyword Server Load Tests (10 simultaneous requests) ........................ 56
List of Figures

1.1 Online Architecture. .............................................. 4
1.2 Offline Architecture ................................................. 5

2.1 Example of a user question and how it is processed internally by MedicineAsk. ........ 8
2.2 Architecture of the Information Extraction module. ........................................... 8
2.3 Architecture of the Natural Language processing module. .................................... 9

3.1 The architecture of ProntApp. ........................................................................ 20
3.2 ProntAPP's User Interface Mockup .................................................................... 24

4.1 Android Application Architecture ................................................................... 28
4.2 iPhone Application Architecture .................................................................... 29
4.3 Delegation pattern implemented in MedicineAsk. ........................................... 31
4.4 Extensions to the Database ............................................................................. 34
4.5 Content of the "alarm_quantity_type" table .................................................. 34
4.6 Content of the "alarm_quantity_option" table ................................................ 35
4.7 Main Screen before performing a question ...................................................... 41
4.8 Main Screen after performing a question .......................................................... 42
4.9 Medicine Detail Screen .................................................................................... 43
4.10 Add Alarm Screen ......................................................................................... 44
4.11 Alarms List Screen ......................................................................................... 45

5.1 Gender Distribution of Users. ........................................................................ 48
5.2 Age Distribution of Users. .............................................................................. 48
5.3 Distribution of the Area of Expertise of Users. ................................................ 49
5.4 Distribution of the Versions of the Android Devices of Users. ........................... 49
5.5 Distribution of the Versions of the iPhone Devices of Users. ............................. 49
5.6 Years of Expertise of Users. ............................................................................. 50
5.7 Mobile App Use Frequency of Users. ............................................................... 50
5.8 Main Screen Appearance Rating. ..................................................................... 51
5.9 Voice Question Rating. .................................................................................... 51
5.10 Written Question Rating. ................................................................................ 51
Chapter 1

Introduction

More and more medical information is becoming available on the web (e.g., Medscape\(^1\), Epocrates\(^2\), etc.). Both medical staff and common users need to access this information efficiently. However, with that much information, it has become increasingly difficult to search for the correct answers we are looking for in a specific situation. For example, when a user searches on Google “What are the contra-indications of Vastarel?”\(^3\), the answer consists of a list of search hits, meaning that the answer is not answered immediately. There are several web-based medical systems that present detailed medicine information for certain medical questions (e.g., Infarmed\(^3\), Medscape). However the questions are usually keyword-based, therefore they must be posed with limited search terms so that the system is able to answer. Medical professionals are used to search for medicines with very specific keywords and terms. Contrarily, the common user is not that familiar with such terms. In addition, the common user may not be a computer expert, and may have some difficulties using such keyword-based interfaces. These interfaces should be available for users in a more convenient and ever-present way, taking advantage of the new technologies and gadgets, and therefore enabling more users to access the information they need, exactly when it is convenient for them.

Currently, the most popular gadget in the world is clearly the smartphone; as a matter of fact, such devices are becoming more and more popular. It is estimated that, by 2017, at least 34% of the world’s population will own a smartphone, a fast increase with respect to 9.6% in 2011\(^4\). In terms of the Portuguese market, 55% of the phones sold in the country’s are smartphones, \(^5\) and this number is rapidly increasing. We believe that it is of utmost relevance to have information about medicines available on a smartphone. This enables to offer access to such information anywhere, in a convenient way, and taking advantage of many of the smartphone’s unique features.

\(^1\)http://www.medscape.com/
\(^2\)http://www.epocrates.com/
\(^3\)http://www.infarmed.pt/
1.1 Objective

The main goal of this thesis is to develop a mobile application, named ProntAPP, to answer questions about medicines in Portuguese. This application is intended to be used by the common user, including elder people, as well as medical staff. However, elder people are not the most proficient smartphone users. Additionally, we had the opportunity to gather some feedback with some medical doctors to find out how they would use such an application, and the most important features that the application should have, in addition to answering medical questions. From their feedback, we came to the conclusion that medical doctors deal with the same set of medicines on a regular basis, and would benefit from having them easily and rapidly accessible in the application. In addition, they proposed that, since the application is also targeted to the common user and elder people, there should be some kind of reminder to take the medicines in due time, with options to control the type of dosage, amount, and several hours of the day.

1.2 Requirements

Taking the target users and their limitations into account, and also the feedback that we gathered from medical doctors, the application must fulfill the following requirements:

- R1. Users must be able to pose natural language questions about medicines in Portuguese, and thus should not be limited to keyword-based questions.
- R2. The information about medicines that is returned as an answer must be as detailed as possible.
- R3. Users must be able to search for what they need with a simple interface (e.g., using a speech recognition interface), and in a small amount of steps.
- R4. The application should provide easy access to frequently used medicines.
- R5. The application should remind the user to take their medicines in due time with the ability to choose the type of dosage, amount, and several times a day.
- R6. The application must be available in a large number of mobile devices, ranging from low-end to high-end devices, and running different operating systems.

Currently, medical mobile applications available from Google Play (Android) and the App Store (iPhone) do not fulfill this set of requirements. In particular, most applications are only available in English, and do not offer a question-answering functionality (i.e., a natural language interface). Epocrates, Medscape and Drugs.com only offer their applications in English and with a keyword-based search. The only application available in Portuguese, eMed.pt, also does not have natural language interface. None of these applications offer a voice recognition functionality, and only eMed.pt offers an alarm functionality, however a very basic one (without the ability to set multiple hours per day). We provide more details about such solutions in Chapter 2.
1.3 Proposed Solution

In order to satisfy the set of requirements mentioned above we developed two mobile applications, one for Android and one for iPhone, which present the following set of features:

- **Data about medicines**: ProntAPP provides its users with vast information about medicines, such as contra-indications, pricing and dosage.

- **Question-Answering for medicines in Portuguese**: ProntAPP recognizes Portuguese natural language questions. It also lets users search for the information they need by posing spoken questions to the system, and by searching for keywords.

- **Online server communication**: ProntAPP leverages on the computational power of online servers to perform highly intensive tasks, like voice recognition and natural language processing.

- **Voice recognition**: To further facilitate the user’s input, ProntAPP provides a voice recognition interface.

- **Favorites**: ProntAPP has a favorite medicines list, so they are easily accessible. The user can add a medicine to their favorites list, and also remove it.

- **Alarm**: ProntAPP has the ability to set alarms for medicines, with the ability to choose dosage type, amount and multiple hours per day. This alarm system reminds the user to take his medicines in due time.

- **Large Number of Supported Devices**: The developed mobile applications support 99.8% of all Android devices\(^6\) and 84.2% of all iPhone devices\(^7\)

1.4 Contributions

This section describes the main contributions of this thesis. As a result of this thesis, two native mobile applications were developed: one for Android and one for iPhone. The applications will be available for download on Google Play and AppStore. The main features of these applications are the ability to answer natural languages in Portuguese about medicines, voice recognition using Google’s voice recognition and Nuance Dragon, the ability to add a medicine as a favorite and the ability to add a repeating alarm for several hours of the day for a given medicine. This application is the only one currently available that has this functionalities, being innovative in its field.

Furthermore, the question-answering module of the application was implemented using MedicineAsk. To do this, we had to do some major code refactoring in the online version of MedicineAsk in order to develop an API so that the question-answering functionality could be accessed from outside clients (Android and iPhone). This API was developed with Jersey\(^8\) which supports the development of RESTful web-services in Java.

\(^6\)https://developer.android.com/about/dashboards/index.html
\(^7\)https://david-smith.org/iosversionstats/
\(^8\)https://jersey.java.net/
1.5 Typical Architectures of Mobile Applications

This section presents some background aspects. In particular, we describe the two main architectures that are usually found in mobile applications: online and offline. The first one considers that the mobile device is always connected to the network with obvious drawbacks in terms of mobile data usage (e.g. 3G data plans), connectivity (i.e. it is only available if the mobile device is connected to the network) and battery usage (i.e. the mobile device uses more battery if it is connected to the network). The second option, offline architecture, enables the mobile device to run the application without the need to be always connected to the network, and has a lower battery consumption, however it can not leverage on the speed of an outside machine performing demanding tasks (i.e., voice recognition).

1.5.1 Online Architecture

The *online architecture* assumes that the *client* application is always connected to the Internet. This architecture is illustrated in Figure 1.1. The three main components of the architecture are: the *client*, the *server* and the *database/repository*.

![Online Architecture Diagram](image)

The *client* (e.g. smartphone, desktop, smartwatch) provides a user interface (through form inputs, buttons, etc.) where the output is shown (through tables, text, etc.). The *server* contains all the business logic in terms of data processing and retrieval, and is able to answer requests sent by the *client*. The *database* stores all the required data to answer the *client*’s requests.

The *client* (running on a desktop computer or a smartphone, for example) communicates with the *server* via HTTP requests. When the *server* receives a request, it sends the corresponding query to the *database*. After retrieving the requested data from the *database*, the *server* encodes the answer into a specific format (typically, JSON or XML). The *client* receives the HTTP response, which contains
the JSON or XML structures with the requested data, processes it, and presents it through the user interface.

This kind of architecture has several positive aspects. All the business logic in terms of data processing, retrieval and database communication are performed by the server. This means that any future upgrades to the system are available to all clients seamlessly, independently of the operating system in use. Furthermore, the clients can leverage on the speed of an outside machine performing demanding tasks, which may be a requirement for highly computational intensive tasks (i.e., speech recognition).

The negative aspects of this architecture are as follows: (1): the application requires the client to be always online to fetch information (which may not always be possible), and (2): there must be an existing server available online.

1.5.2 Offline Architecture

The offline architecture enables the client application to access the data even without being connected to the Internet. This architecture is illustrated in Figure 1.2. It has two main components: the client and the server.

![Offline Architecture Diagram]

Figure 1.2: Offline Architecture

The client, similarly to the online architecture, provides a user interface (through form inputs, buttons, etc.) where the output is shown (through tables, text, etc.). Additionally, all the business logic, data processing and database communication is the responsibility of the client application. Therefore, it is required to store a copy of the database locally in the client, so it can be accessed without network requests.
In order to have the database available locally, there are essentially two approaches. First, the database can be packaged directly with the application, so that, when a user downloads the application, the database is also downloaded. In the rest of the document, we will refer to this approach as the packaged database approach. Second, the application can be downloaded without the database. In the rest of the document, we will refer to this approach as the downloaded database approach.

The downloaded database approach requires an existing server to handle the database update logic (i.e., making available a new copy of the database when required). Then, when the application is first launched it downloads the most recent version of the database from a server. This operation of checking for database updates can also be performed every time the client launches the application and is connected to the Internet, to ensure the most up-to-date database.

The main benefit of the packaged database approach is that the client does not need to be connected to the Internet when first using the application. The downloaded database approach requires that the client has Internet connectivity at least when using the application for the first time. Furthermore, the packaged database approach implies that database updates are only possible when a new version of the application is released. In the downloaded database approach, database updates are more seamless and do not require any application update.

The offline architecture has the obvious advantage of being independent from network connectivity (i.e., it works whenever the client is online or offline). However, the business logic, data processing and database communication must be implemented in the client. This means that these components must be re-implemented if a new client on a different operating system is needed. As an example, if the components are implemented in Java (for an Android client), they must be re-implemented in Objective-C (for an iPhone client). Furthermore, it cannot leverage on the speed of an outside machine performing demanding tasks (i.e., an online server), which may be a requirement for highly computational intensive tasks (i.e., speech recognition). Another downside of the offline architecture is that the whole database would have to be downloaded, even if the user did not want to search for all data, implying costs in terms of data transfer to both the client and the server.

1.6 Thesis Outline

This document is organized as follows. In Chapter 2, we review the most relevant question-answering systems for medicines, existing Android and iOS medical mobile applications and state-of-the-art of voice recognition systems. Chapter 3 describes ProntAPP’s architecture and its main components. In Chapter 4 we present the implementation of ProntAPP: the question-answering module, online server API, database, voice recognition module, alarms and favorites functionality, the software libraries used and the user interface. Chapter 5 describes the experiments conducted to validate ProntAPP and the results obtained. In Chapter 6, we conclude and present directions for future work.
Chapter 2

Related Work

This section presents several systems that are relevant and related to ProntApp. First, we describe MedicineAsk and MEANS, two question-answering systems that support Natural Language questions (the first one in Portuguese and the second one in English, see Sections 2.1 and 2.2). Second, we present Epocrates, Medscape, Drugs.com and Infarmed which are medical web-based systems that accept keyword-based questions. We also present the eMed.pt mobile application, which is the only mobile application currently available in Portuguese, as far as we know, that is able to provide information about medicines. Third, we present the state of the art in terms of voice recognition systems for mobile applications. In particular, we discuss Google’s voice recognition system, CMU Sphinx, developed at the Carnegie Mellon University and Nuance Dragon. Finally, we present a comparison of all the systems described.

2.1 MedicineAsk

The MedicineAsk system [12][6] is a question-answering prototype for the medical field, which answers questions about medicines and active substances posed in Natural Language (Portuguese). It extracts data regarding medicines and active substances from the Infarmed’s Pharmacists’ Schedule\(^1\) (Prontuário Terapêutico) website, and stores the data extracted in a relational database. Given a user question in Portuguese, MedicineAsk interprets and translates it into an SQL query, which is posed to the relational database that provides the answer. Figure 2.1 shows an example of a user question and how it is processed by MedicineAsk.

The MedicineAsk architecture contains two main modules: (1) the Information Extraction module and (2) the Natural Language Interface module.

2.1.1 Information Extraction Module

This module aims at extracting data from the Infarmed’s Prontuário Terapêutico website, processing and storing it into a relational database. This module is further decomposed into four components: web data

\(^1\) http://www.infarmed.pt/prontuario/index.php
Figure 2.1: Example of a user question and how it is processed internally by MedicineAsk.

![Figure 2.1](image)

Figure 2.2: Architecture of the Information Extraction module.

![Figure 2.2](image)

extraction, processing of entity references, annotation and the relational database.

The web data extraction component is responsible for parsing the Infarmed’s website hierarchical structure, which is divided into chapters, sub-chapters and active substances. When the user is visualizing a page with data about a specific medicine in Infarmed’s website, sometimes there are references to other entities, which link to another page with more data about them. The processing of entity references component replaces these references with the data present on the linked entity page (i.e., all the data that the user would see if he navigated to the link), so that the current, initial page has all the important data, without having any links.

When visualizing the data regarding an active substance in Infarmed’s website, the description is often regular text, rather than a comma separated list of words (i.e., A typical text is: “Indications: This medicine should be taken in cases of fever and also in case of headaches” instead of “Indications: fever, headache”). In order to answer questions like “What medicines treat fever?”, the active substance data must be first annotated (i.e., some notes are added to the words). The annotation module analyses the words of that sentence and tags the active substance based on that list (i.e., the active substance would be annotated with the words “fever” and “headache”).

The relational database stores the following main types of data: the Infarmed’s website hierarchy
structure, the chapter data for each chapter of the Infarmed website, and the data corresponding to each active substance, such as its name, retail price, laboratory, package size, pharmaceutical form and comparticipation.

2.1.2 Natural Language Interface Module

This module is responsible for analyzing and processing a natural language question that is posed to the system. It is composed of the following three components: question type identification, question decomposition and question translation. The question type identification component identifies what the question is about, i.e., if a question is about adverse reactions or about the correct dosage of a medicine. The question decomposition module finds the relevant entities mentioned in the question, e.g., which active substance does the user want to know the adverse reactions of. The third component, question translation module, converts the natural language question into an SQL query that is posed to the relational database.

2.2 MEANS

MEANS [1] is a question-answering system for the medical domain. Its purpose is to answer Natural Language questions about medicine in general using medical data obtained from publicly available resources. The question analysis steps in MEANS are as follows:

- **Question Type Identification**: Questions are divided into YES/NO or WH questions. WH questions start with HOW, WHAT, WHICH or WHEN and have an answer with more information than a simple YES or NO. The question type is determined by applying a simple set of rules to the input questions (e.g., if the question begins with a HOW, a WHAT, a WHICH, or a WHEN then it is a WH question).

- **Expected Answer Type Identification (EAT)**: This step is necessary when dealing with WH questions to discover what type of answer will be returned. The possible answer types were defined
by the authors and are the following: MEDICAL PROBLEM, TREATMENT, MEDICAL TEST, SIGN OR SYMPTOM, DRUG, FOOD, and PATIENT. The EAT is determined using lexical patterns previously built by hand for each question type. WH questions are matched against these pre-built question patterns to find its answer type. Multiple EATs for a single question are saved in order to answer multiple focus questions such as "What are the symptoms and treatments for pneumonia?". In this particular case, the EATs are: SIGN OR SYMPTOM or TREATMENT.

• **Question Simplification:** This step applies a simplification to the question to improve the analysis. The main goals of this step are to replace interrogative pronouns by the ANSWER keyword (e.g., turning "What is the..." into ANSWER) and to transform the question into its affirmative form, by removing the question mark. As an example, the question "What is the best treatment for pneumonia?" would be processed in two steps. First, the "What is the best treatment" would be replaced with ANSWER, becoming "ANSWER for pneumonia?". Then, it is turned into its affirmative form, becoming "ANSWER for pneumonia". This ANSWER keyword is a special word that the system will ignore in later steps.

• **Medical Entity Recognition:** This step focuses on detecting and classifying medical terms into seven different categories, which match the possible EAT: PROBLEM, TREATMENT, TEST, SIGN OR SYMPTOM, DRUG, FOOD, and PATIENT. To find medical entities, a rule-based method called MetaMap+ [3] and a machine learning method that uses Conditional Random Fields [9] are used.

• **Relation Extraction:** A relation identifies a relationship between two medical entities in a sentence and is very important to determine answers to medical questions. Seven relations are extracted: TREATS, COMPLICATES, PREVENTS, CAUSES, DIAGNOSES, DH(Drug has dose) and PHSS (Problem has signs or symptoms). In the example "Aspirin cures headaches", we have the relation TREATS between the concepts "aspirin" and "headache". The relation extraction step uses a machine learning method called Support Vector Machine (SVM) [5]. A rule-based method is used whenever there are not enough examples in the training corpus to properly train the machine learning method [2].

• **SPARQL Query Construction:** This step constructs a SPARQL query [4]. Initially, a Resource Description Framework (RDF) [7] graph is built. After that, medical entities and relations are extracted from the medical texts, with a process similar to the one used to analyse the questions. The extracted data is then annotated in RDF and inserted into an online RDF database. In this step, natural language questions are translated into a SPARQL query that is then posed to the RDF database.

Analogously to MedicineAsk, MEANS is able to provide an answer to Natural Language questions about medicines and active substances, but it is also able to answer broader questions about Medicine. MEANS supports questions in English, while MedicineAsk supports questions in Portuguese.
2.3 Web-Based Medical Systems

In this section, we present the three most widely used web-based systems that accept keyword-based questions in English: Epocrates\(^2\), Medscape\(^3\) and Drugs.com\(^4\). We also discuss the most widely used web-based system available in Portuguese: Infarmed\(^5\).

All the four mentioned systems can be used as quick references for data about medicines. The four systems enable the user to make a search by disease, and all support medicine *look-up by name*. A *look-up by name* is a mode where the user inputs the medicine name to search, and all relevant information regarding that medicine is returned.

Epocrates, Drugs.com and Infarmed, besides supporting the *look-up by name* also support another search mode, named *look-up by class*. A *look-up by class* is a mode where the user can navigate through a set of classes and subclasses of medicines, selecting the desired drug inside the class (e.g., the "Dermatologic" class has a subclass named "Acne, Systemic", which contains the drug "Isotretinoin"). This mode is particularly useful when the user does not know the name of a medicine, but knows its purpose. Drugs.com also supports a medicine search by medical conditions (e.g. abdominal pain, fever, etc.), where the output are the medicines that handle the input medical condition.

The four systems return information about dosage, contra-indications and adverse reactions of each medicine. Even though the same data about medicines is returned by all the four systems, it is presented differently. Drugs.com presents all the information in a non-structured way, almost as free text. Medscape presents this information (i.e., information about dosage, contra-indications and adverse reactions of each medicine) in a more organized way, without a clear distinction between titles and text. Infarmed presents this information in Portuguese similarly to Medscape, organized in tables.

The Epocrates’ website search box provides help mechanisms when writing the medicine name (auto-complete), and presents the information in a structured way. Epocrates makes a clear distinction between pediatric dosage and adult dosage, it presents the information about drug interactions divided by severity and both contra-indications and adverse reactions are cleanly listed. Only Drugs.com enables phonetic and wild-card search in order to help identifying the correct medicine whenever the spelling of a medicine’s name is unknown and only the pronunciation is well-known.

Epocrates, Medscape and Drugs.com also have a drug interaction checker. This feature allows the user to compare interactions between a list of drugs. The output are all the medication combinations that have interactions between them.

2.4 Medical Mobile Applications

The four web based systems presented in Section 2.3 (Epocrates, Medscape, Drugs.com and Infarmed) have Android and iOS versions. This section presents details about the mobile applications of these systems.

\(^2\)http://www.epocrates.com/
\(^3\)http://www.medscape.com/
\(^4\)http://www.drugs.com/
\(^5\)https://www.infarmed.pt/prontuario/index.php
2.4.1 Epocrates’ Mobile Application

The Epocrates’ Mobile Application is available as both an Android application\(^6\) and an iOS application\(^7\). It has several features: (i) Drug and Medicines Information, (ii) Clinical Practice Guidelines, (iii) Interaction Check, (iv) Pill ID and (v) Notes.

In (i) Drug and Medicines Information, a user can check for: adult and pediatric dosage for FDA-approved and off-label indications (off-label use is the use of pharmaceutical medicines for an unapproved indication or in an unapproved age group, unapproved dosage, or unapproved form of administration); black box warnings (featured in the labeling of medicines associated with serious adverse reactions), contra-indications, adverse reactions, and medicine interactions; safety/monitoring (pregnancy risk categories, lactation safety ratings, monitoring parameters, and similar medicine names); pharmacology (metabolism, excretion, subclass, and mechanism of action); manufacturer, DEA/FDA status, and approximate retail price. In Alternative Medicine Content, concise medicine monographs (i.e. a statement that specifies the kinds and amounts of ingredients a medicine or class of medicines may contain, the directions for the medicine’s use, the conditions in which it may be used, and the contra-indications to its use) for alternative medicines are presented. A user can also access reported uses, reported doses for various conditions, cautions, adverse reactions, a list of synonyms, and other helpful information.

In (ii) Clinical Practice Guidelines, a user can access evidence-based, patient-specific guidelines from national specialty societies. In (iii) Interaction Check, potentially harmful medicine-medicine interactions between up to 30 (brand, generic, OTC, or alternative) medicines at a time can be accessed. OTC medicines are over-the-counter medicines that do not need a prescription. An overview is presented detailing the interaction profile for each active ingredient of the medicine product. In (iv) Pill ID, a user can identify a pill based on its imprint code and/or physical characteristics. Finally, in (v) Notes, a user can take notes about specific medicines.

The use of Epocrates’ mobile application requires user registration in order to access its contents, and it is entirely used offline. It does not feature any Natural Language processing, the only queries available are keyword-based.

The Android mobile application is available free of charge on Google Play. It has over 1 million downloads, and the current devices it supports are Android smartphones with version 2.3 or above. This application is not developed in Java (the native Android programming language). It is developed in HTML, which hurts the performance (i.e. it runs slower on Android than an application developed in Java), thus making the application slow and unresponsive to use.

The database that stores data is not packaged with the application since it occupies 13MB. Upon entering the application for the first time the database is downloaded, so that offline use is possible.

2.4.2 Medscape’s Mobile Application

Medscape from WebMD is mostly used by physicians, medical students, nurses and other health care professionals for clinical information. It has the following features: (i) Drug Information, (ii) Disease and

\(^7\) https://itunes.apple.com/pt/app/epocrates/id281935788?mt=8
Condition Information, (iii) Medical Calculators and (iv) Drug Formulary Information.

The Medscape system is available on Android’s Google Play\(^8\) and iOS’ App Store\(^9\). The mobile application supports offline access but, like Epocrates, it does not have any Natural Language processing. The only queries supported are keyword-based queries.

The Android application alone has over 5 million downloads, and is available for free on Google Play. It supports Android 2.3 version or above and it is not developed in Java. It is developed in HTML. Similarly to Epocrates, upon entering the application the database is downloaded, and stored offline for further offline uses.

### 2.4.3 Drugs.com’s Mobile Application

Drugs.com Medication Guide\(^10\) is the Android and iOS\(^11\) version of the Drugs.com website. The Android application on Google Play has over 500,00 downloads and is available for free. It supports lower versions of Android than the previous mentioned applications. Users with Android 2.2 version or above can use the application.

The main features of the Drugs.com website are also present in this available application. The application features: (i) a pill identifier, (ii) drug listing and (iii) interactions checker. Unlike the Epocrates and Medscape applications, it is only available if the smartphone is connected to the Internet. Therefore, the database is not downloaded to the device. Additionally, it only supports keyword-based queries.

### 2.4.4 eMed.pt

eMed.pt is the official Infarmed Android\(^12\) and iOS application\(^13\). It is the only application available in Portuguese at the moment that returns information about medicines.

The application allows the user to search for medicines and returns information active substances, price, dosage and composition. It has some features, besides being in Portuguese, that distinguishes it from the mobile application of Epocrates, Medscape and Drugs.com. On eMed.pt, a user can create alarms for taking medicines. Furthermore, a map is available with all the Portuguese pharmacies’ location, together with telephone information.

It is also the only application of the above mentioned that features barcode scanning (i.e., you can scan a barcode of a medicine to get data about it, instead of inputting its name).

The Android application has over 5000 downloads on Google Play. It is also available for free. Users must have an Android with version 2.3.3 or above in order to download and install the application on their smartphone. Upon installation, it is available entirely offline.
2.5 Voice Recognition Systems

In this section, we present the state of the art in terms of voice recognition systems. In particular, we discuss Google's voice recognition system\textsuperscript{14}, CMU Sphinx, developed at Carnegie Mellon University\textsuperscript{15}, and Nuance Dragon\textsuperscript{16}.

The Google's voice recognition system is widely used among mobile applications. It already supports the Portuguese language, without any further training. This system is not intended to be used for continuous recognition, since it streams audio to remote servers to perform speech recognition, which means it cannot be used completely offline.

CMU Sphinx has a version that can be used in embedded systems (e.g., based on an ARM processor) named PocketSphinx. PocketSphinx is under active development and incorporates features such as fixed-point arithmetic and efficient algorithms for GMM \cite{11} computation.

PocketSphinx can be used completely offline and packaged with the client. Voice recognition systems require a model to train the system and tweak the parameters. However, currently it does not have any Portuguese model to train on, which means it does not support Portuguese voice recognition. The system supports training by a user built acoustic model. However, according to the creators of CMU Sphinx, you would need 1 hour of recording for command and control for single speaker, 5 hour of recordings of 200 speakers for command and control for many speakers, 10 hours of recordings for single speaker dictation, 50 hours of recordings of 200 speakers for many speakers dictation, knowledge of phonetic structure of the language and approximately 1 month to train the model and optimize parameters. CMU Sphinx is released under BSD \textsuperscript{17} style license, which imposes minimal restrictions on the redistribution of the software.

Nuance Dragon is developed by Nuance and has support for speech-to-text recognition in all the major mobile platforms: Android, iOS and Windows Phone. It also supports Portuguese Language without any model training and, like Google's voice recognition, is not intended for continuous recognition, since the voice recognition is performed at Nuance's online servers. It is free up to 20,000 transactions (voice processing) per month, charging $.008 per transaction after that. It also offers Cloud Text-to-Speech (TTS) for 40+ languages and 60+ voices.

2.6 Comparison of question-answering systems

In Sections 2.1 and 2.2 we detailed MedicineAsk and MEANS, which are two question-answering systems for medicines. Both systems are able to provide an answer to Natural Language questions about medicines and active substances. MEANS supports questions in English, while MedicineAsk supports questions in Portuguese. A summarized comparison of the two systems is presented on Table 2.1.

There are some differences between MedicineAsk and MEANS. MedicineAsk uses Infarmed's website as its information source, while MEANS uses publicly available resources. The method used by

\textsuperscript{14}http://developer.android.com/reference/android/speech/SpeechRecognizer.html
\textsuperscript{15}http://cmusphinx.sourceforge.net/
\textsuperscript{16}http://developer.nuance.com/
\textsuperscript{17}http://en.wikipedia.org/wiki/BSD_licenses
Table 2.1: Comparison between MedicineAsk and MEANS.

MedicineAsk for answering the questions is by querying the extracted information using SQL. On the other hand, MEANS annotates the processed documents in RDF to build a graph traversed to answer queries using SPARQL. MedicineAsk can answer questions about medicines, active substances and what they treat. MEANS has a broader scope, answering questions about medicines, active substances and what they treat, who should take medicines and differences between diseases identification. Finally, MedicineAsk is available in Portuguese while MEANS is available in English.

2.7 Comparison of voice recognition systems

In Section 2.5 we detailed three voice recognition systems that can be used on embedded systems: Google’s voice recognition system, CMU Sphinx and Nuance Dragon. A summarized comparison of the three systems is presented in Table 2.2.

Table 2.2: Comparison between Google’s voice recognition system, CMU Sphinx and Nuance Dragon.

Google’s and Nuance’s voice recognition are not open-source, therefore they cannot be modified in any way. CMU Sphinx is open-source with a BSD License, which imposes minimal restrictions on the redistribution of the software. Both Google’s voice recognition system and CMU Sphinx’s are available for free, while Nuance’s is free only up to 20,000 transactions per month, charging $.008 per transaction after that. Google’s voice recognition system and Nuance’s is available for the Portuguese language while CMU Sphinx does not yet support Portuguese language (however the system can be trained like explained in Section 3.5). Both Google’s voice recognition and Nuance’s is performed on online servers (cloud speech recognition), therefore they are only available if the client is connected to the network, while CMU Sphinx’s voice recognition is performed locally.
2.8 Comparison of existing medical mobile applications

In Section 2.4 we detailed several mobile applications: Epocrates, Medscape, Drugs.com and eMed.pt. A summarized comparison of the four applications is presented in Table 2.3.

All of the described applications are available for Android and iOS. eMed.pt is the exception in terms of language, since it is the only mentioned application with information in Portuguese. All applications are available to download and use for free, however Epocrates and Medscape have a paid subscription with more functionalities. Offline use is possible in all applications except for Drugs.com which is only available if the device is connected to the network. Keyboard-based search is also supported by all applications, however none of them supports natural language search or speech recognition. Epocrates, Medscape and eMed.pt have a local database on the device which is downloaded from an online server, while Drugs.com does not. Drugs.com and eMed.pt are the only applications developed natively for the operating systems, being developed in Java (native programming language of Android) and Objective-C (native programming language of iOS). Both Epocrates and Medscape are developed in HTML.

<table>
<thead>
<tr>
<th></th>
<th>Epocrates</th>
<th>Medscape</th>
<th>Drugs.com</th>
<th>eMed.pt</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>English</td>
<td>English</td>
<td>English</td>
<td>Portuguese</td>
</tr>
<tr>
<td><strong>Operating Systems (OS)</strong></td>
<td>Android/iOS</td>
<td>Android/iOS</td>
<td>Android/iOS</td>
<td>Android/iOS</td>
</tr>
<tr>
<td><strong>Minimum Version</strong></td>
<td>Android: 2.3 iOS: 7.0</td>
<td>Android: 2.2 iOS: 7.0</td>
<td>Android: 2.3 iOS: 7.0</td>
<td>Android: 2.3.3 iOS: 8.0</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>Free download with possibility of paid subscription</td>
<td>Free download with possibility of paid subscription</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td><strong>Offline Use</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Local Database</strong></td>
<td>Downloaded to the phone on first use</td>
<td>Downloaded to the phone on first use</td>
<td>Does not have</td>
<td>Downloaded to the phone on first use</td>
</tr>
<tr>
<td><strong>Keyword-Based Search</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Natural Language Search</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Speech Recognition</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Main Programming Language</strong></td>
<td>HTML</td>
<td>HTML</td>
<td>Java/Objective-C</td>
<td>Java/Objective-C</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison between Epocrates, Medscape, Drugs.com and eMed.pt.

2.9 Discussion

Since neither of these applications fulfill the set of requirements listed in Section 1, we propose a mobile application, named ProntAPP, which fulfills the needed requirements. This application will use MedicineAsk for the question-answering system since it is the most advanced available in Portuguese, while MEANS is only available in English. In terms of voice recognition, we will use Google’s voice recognition for the Android mobile application and Nuance Dragon voice recognition for the iOS mobile application, since they are the most advanced in their respective operating systems. Although Epocrates,
Medscape, Drugs.com and eMed.pt present detailed information about medicines, they fail to meet our requirements of being able to answer natural language questions in Portuguese, having easy access to frequently used medicines, an alarms functionality (except for eMed.pt) and neither of these has a voice recognition system, hence the need for the development of ProntAPP.
Chapter 3

ProntAPP: Architecture

In this chapter, we start by presenting the features of ProntAPP that are needed to satisfy the requirements listed in Chapter 1 (Section 3.1). Second, we present the proposed architecture for ProntAPP (Section 3.2). Third, we discuss the main components of the proposed architecture, namely the question-answering module, the voice recognition module, the database and the online server API (Section 3.3). Fourth, we present the main differences between native and hybrid mobile applications that were taken into account in our decision of developing native mobile applications (Section 3.4). Fifth, we present a mockup of the proposed user interface and explain the user interaction with the application (Section 3.5). Finally we present a summary of this chapter (Section 3.6).

3.1 Features

In order to satisfy the set of requirements listed in Section 1.2, the following set of features (and corresponding requirements) must be supported by ProntApp:

- **Question-Answering for medicines in Portuguese**: ProntAPP must recognize Portuguese natural language questions. It must also enable users to perform keyword-based search (R1).

- **Data about medicines**: ProntApp must provide a vast set of information about medicines, namely the contra-indications, pricing and dosage (R2).

- **Online server communication**: ProntApp must leverage on the computational power of online servers to perform highly intensive tasks, like voice recognition and natural language processing, so that it can be also available for low-end devices (R1,R3,R6).

- **Voice recognition**: ProntApp must provide a voice recognition interface, so that users are able to search for what they need with a simple interface and in a small amount of steps (R3).

- **Favorites**: ProntApp must support a favorites functionality, so that the user can add the most often accessed medicines to be accessed quickly (R4).
• **Alarm**: ProntApp must support the ability to set recurring alarms for medicines. This alarm system must remind the user to take its medicines in due time (R5).

### 3.2 Architecture

ProntApp features an **online architecture**, which means that its main functionalities are only available if the user is connected to the Internet. We have chosen this architecture because it is quite popular in mobile applications nowadays, and facilitates further ports of the mobile application to other operating systems, since all the main business logic is stored server-side. Furthermore, we can leverage on the computational power of online servers to perform intensive tasks, like voice recognition.

There are two databases in ProntAPP: *(1)* one available online in the question-answering online server, which we refer to as **online database** and *(2)* one local database available offline in the mobile client application, which we refer to as **offline database**. The **online database** stores all the medicines data, active substances, interactions, medical conditions, etc. This information is only available in the **online database**. The **offline database** is only present in the mobile client application, and only stores the information about medicines added to the favorites list, and about the alarms configured by the user.

![Figure 3.1: The architecture of ProntApp.](image-url)

The architecture of ProntApp is represented in Figure 3.1. ProntApp works on a mobile phone which runs the mobile client application. The mobile client application is divided into three layers: the Presentation Layer, the Business Layer and the Data Layer. The **Presentation Layer** provides a user interface where the input of the application is specified (e.g. form inputs, buttons, etc.) and its output is shown (e.g. tables, text, etc.). It also contains the presentation logic which shows the data in the native user interface of the operating system (i.e. in the case of Android, this logic means displaying...
the data in Android’s views). The Presentation Layer communicates with the Business Layer when a user interacts with the application. The Business Layer contains all the libraries that ProntAPP uses. In particular, it makes the necessary requests to perform the voice recognition (which is performed in online servers), question-answering processing (performed in an online server) and business logic (e.g. alarm functionality, favorites logic, server communication).

When the user accesses its Favorite medicines or the Alarms, the Business Layer requests data from the Data Layer. The Data Layer stores the local database, handles all the database communication using data access components (i.e., SQL Connectors), and converts the data returned by the database into structures that can be handled by the other layers, with database helpers which perform the mapping from SQL cursors into structures.

As mentioned above, an offline database is stored in the mobile client application. Since the search functionality is only available if the user is online, we found that it was not necessary to store a copy of the online database on the client application. Therefore, the local database only stores information about the favorite medicines, not all medicines. When a user adds a medicine as a favorite, all the information is fetched and stored for offline use. The local database also stores information about the alarms scheduled by the user.

The online server provides an API that the mobile application uses for the question-answering functionality, via HTTP requests, further detailed in Section 3.3.3.

3.3 Modules

This section details the modules of the architecture presented in Figure 3.1, presented in Section 3.2. In Section 3.3.1 we discuss the Voice Recognition module and explain how it is used. In Section 3.3.2 we present some considerations about the online and offline database and lastly in Section 3.3.3 we describe the Question-Answering module and how we approached the integration with the mobile client application.

3.3.1 Voice Recognition Module

Voice recognition is a very CPU intensive task to be handled on smartphones. Furthermore, the best voice recognition services available in Portuguese (Google and Nuance Dragon) are only available when the smartphone is connected to the Internet, since the speech processing and recognition is performed on online servers. This was taken into consideration when integrating the voice recognition functionality. When the user asks a question to ProntAPP, the voice recognition module sends the data to online servers (Google in the case of Android, Nuance Dragon in the case of iOS). These voice recognition systems output a string (which is the result of the voice mapping process) with a certain degree of precision. Since the input of the question-answering module is a string, the retrieved string from the voice recognition system is then analysed by the question-answering module, in the same manner as a regular text question is analysed.


3.3.2 Database

As discussed earlier, there are two databases in the scope of ProntAPP: one available online on the server, which we refer to as online database and one available offline on the mobile client application, which we refer to as offline database.

The online database stores all the medicines data, active substances, interactions, medical conditions, etc. This information is only available in the online database, and not on the local database. This online database is stored in the online server.

The offline database is only present in the mobile applications, and is typically in an SQLite format since it is the format natively supported by all mobile operating systems. Since the search functionality is only available if the user is online, we found that it was not necessary to store a copy of the database on the client application. Therefore, the offline database simply stores the favorite medicines (which can be fully accessed offline) and the scheduled alarms.

3.3.3 Question-Answering Module and Question-Answering Online Server

The question-answering module is responsible for interpreting the natural language question and retrieving the results. This question-answering module of the mobile applications connects to a question-answering online server via HTTP requests in order to do this.

The online server performs the natural language interpretation (and stores the online database, as explained above). This means that the server needs to provide external access to the information about medicines, so that they are available to the mobile client application. This is done by offering an API to any clients that require access to this information. An API is a set of functions and procedures that allow the creation of applications which access the features or data of another service. This API needs to expose two functions that allow the mobile client applications to perform two actions: (1) process the natural language and retrieve information about medicines, and (2) retrieve the available alarm quantity types and dosages for scheduling alarms in the mobile client applications.

The server's API retrieves the requested information in a generic format (i.e. JSON, XML) so that each client that wants to access it can process and display it arbitrarily, not being forced to display it in pre-defined format (i.e. HTML). When a user makes a question, the question-answering module sends the corresponding string to the question-answering online server, which performs the analysis and retrieves the answer. The response is then parsed and displayed in the mobile client application.

3.4 Native vs. Hybrid Applications

Building native applications means using the native language of the platform (i.e., Objective-C on iOS, and Java on Android). The main advantage of native applications against hybrid applications is their performance. Native apps are compiled into machine code (Dalvik byte code under Android), which gives the best performance you can get from the mobile phone.
Best performance includes fast and high framerate animations as well as full access to phone hardware, multi-touch support and the latest APIs available directly from the SDK. As code must be written specifically for each platform, code cannot be re-used in different applications. The business logic may be the same, but the language, APIs and the development process is different.

Hybrid applications are web applications (or web pages) in the native browser, such as UIWebView in iOS and WebView in Android (not Safari or Chrome). Hybrid apps are developed using HTML, CSS and Javascript, and then wrapped in a native application using platforms like Cordova\(^1\). This allows you to use any web-native framework you want.

The application development is faster, simpler and the application is easier to maintain. The main problem with hybrid apps is that they still depend on the native browser, which means they are not as fast as native apps.

Both native and hybrid are ways to fulfill the different needs and preferences of users and developers, however taking into account that performance is a critical point in our application, and direct access to the latest APIs is needed in order to access the microphone and perform voice recognition, we have decided to develop two native applications, one for iPhone (Objective-C) and one for Android (Java).

### 3.5 User Interface

The user interface mockup was designed on NinjaMock\(^2\), which is an online tool that facilitates the user interface design phase. NinjaMock delivers all of the typical controls used in mobile apps and web designs: iOS, Android and Windows Phone. Therefore it is a very useful tool when prototyping apps for mobile device projects. A mockup of the graphical user interface is presented in Figure 3.2.

On the top of the screen, on the action bar, there is the application icon, the name of the application and a help icon. The help icon displays a dialog with a brief explanation of the accepted questions, and a list of questions that the user can click and test immediately. Below the action bar, there are three tabs. By default, the search tab is selected and the search screen displayed. The user can either search with the phone’s keyboard and click "Pesquisar", or he can click on the microphone icon and make a voice-based search. Then, a list of results is presented, the name, the retail price, the laboratory and the package size. The arrow on the list items represent that the user can click on an item and receive more information about that medicine. When the user clicks the item on the list, he is presented with the medicine details screen, with more detailed information about the medicine.

In addition to the information presented on the previous screen, the user is also presented with more specific information about the composition of the medicine, if the medicine is a generic or not, pharmaceutical form, grouping, compaticipation and PMU. On the top of the screen, on the action bar, there are three icons present. The left icon enables the user to add an alarm to take that specific medicine at a given time. The alarm will be added to the list of alarms accessible through the last separator of the first screen. The center icon (the heart) allows the user to add this medicine to his

\(^1\)https://cordova.apache.org/
\(^2\)https://ninjamock.com/
Figure 3.2: ProntAPP’s User Interface Mockup

The favorites list for easier access (or remove it, if already a favorite). The favorites list is accessible through the second separator on the first screen. Finally, the last icon allows the user to share this medicine’s information (e.g. the name, the retail price, the laboratory, the package size, etc.) via e-mail, SMS, or any other share media depending on the applications installed on the smartphone (Twitter, Facebook, etc.).

The favorites screen is very similar to the search screen, however without the search box and buttons, it simply presents a list of the medicines without any further additions. The alarms screen was not mocked and was simply introduced directly in the application by taking into account designs of other alarm applications, like the Android application Timely³.

3.6 Summary

In this chapter, we started by presenting the features of ProntAPP, which provides users with vast information about medicines, a question-answering system for medicines in Portuguese, voice recognition, favorite medicines and alarms. Second, we presented the proposed architecture (regardless of the oper-

ating system it is implemented on), which is module-based. Third, we proceeded by discussing the main modules present in our proposed architecture, namely the question-answering module, voice recognition module, database and the online server API. Fourth, we presented the main differences between native and hybrid applications that were taken into account in our decision of making purely native applications. Finally, we presented a mockup of the proposed user interface made in MockNinja and explained the user interaction with the application, detailing the purpose of each button and how each feature will be present in the user interface.
Chapter 4

ProntAPP: Implementation

In this section we describe the implementation of ProntAPP. We start by presenting the chosen target mobile operating systems, Android and iPhone, and our reasons. Next, we present the question-answering module. In particular, the changes applied to the MedicineAsk system and the implementation of the API. We proceed by discussing some details about the databases of ProntAPP (online and offline). Next, we describe the implementation of the voice recognition module, and the implementation of the alarms and favorites functionalities, both for Android and iPhone. We present the main libraries used in our development process and lastly we present some screenshots of the final user interface.

4.1 Target Mobile Operating Systems

There are three main markets in terms of mobile applications: Android, iOS and Windows Phone. In Portugal, Android and iPhones take the majority of the market share.\(^1\) Android alone represents 82% of the smartphones sold in Portugal. Therefore, we believe that these are the mobile platforms that must be prioritized. Accordingly, we will develop ProntApp as an Android and an iPhone native mobile application.

4.1.1 Android

If we analyze Android versions’ market distribution,\(^2\) we see that approximately 99.8% of Android smartphones are on version 2.3.3 (Android SDK 10) or above. Therefore, the application was developed to work on devices with the Android SDK 10 or above.

The application is developed in Java language, using Android’s SDK. We use a typical Model-View-Controller [8] software architecture since it is the standard when developing Android mobile applications.

The Android Application was developed taking into account the most recent material design guidelines proposed by Google. Expanding upon the “card” motifs first seen in Google Now, it is a design with

---

2. [https://developer.android.com/about/dashboards/index.html](https://developer.android.com/about/dashboards/index.html)
increased use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows.

Regarding the voice recognition functionality, ProntApp uses Google’s voice recognition, since it is the most advanced voice recognition system available in Portuguese, and has great support and tools for integrating the system with mobile applications. Google’s voice recognition is processed on Google’s online servers, which means that for using this specific functionality the smartphone must be connected to the network, as explained previously.

Figure 4.1 shows the Android application’s architecture, as an implementation of the architecture presented in Section 3.2.

![Android Application Architecture](image)

In this figure, we notice that the user interface is implemented using native Android views, provided by the Android SDK. Furthermore, the voice recognition module is implemented using Google’s Speech Recognizer. The question-answering module is implemented by accessing the new MedicineAsk’s API using an Android software library called Retrofit. In addition, the local database is an SQLite database[10] and is accessed through an Android software library called ActiveAndroid. These software libraries are better described in Section 4.7.1.

### 4.1.2 iPhone

Approximately 86% of Apple devices are on iOS 8.0 or higher (which means iPhone 4S or better). Therefore, this is the operating system version that most developers target, since they can make use of

---

3 [http://square.github.io/retrofit/](http://square.github.io/retrofit/)
new features of XCode like Auto-Layout and adapt their design to multiple screen densities (iPhone 5, 6 and 6 Plus). Consequently, ProntApp was developed to work on devices with iOS version 8.0 or higher (including iPads, although the design is not adapted to tablet devices).

The application is developed in Objective-C, using the iOS SDK. Like on Android, we use a typical Model-View-Controller software architecture since it is the standard when developing iPhone mobile applications.

The iPhone application was developed by the standard iOS design guidelines, meaning that the design was not a direct port from the Android version (for instance, on Android it is usual to have the tabs on top of the screen, on iPhone tabs usually appear at the bottom).

Regarding the voice recognition functionality, ProntApp uses Nuance Dragon Mobile SDK\(^6\). It is the most advanced voice recognition system available in Portuguese for iPhone, as discussed in Section 2.7. This voice recognition, like Google’s, is processed on online servers.

Figure 4.2 shows the iPhone application’s architecture, as an implementation of the architecture presented in Section 3.2.

![iPhone Application Architecture](image)

In this figure, we notice that the user interface is implemented using native iOS views, provided by the UIKit Framework\(^7\) in the iPhone SDK. Furthermore, the voice recognition module is implemented using Nuance Mobile Dragon SDK. The question-answering module is implemented by accessing the new MedicineAsk's API using an iOS software library called RestKit\(^8\). In addition, the local database is an SQLite database and is accessed through an iOS software library called FMDB\(^9\). These software

\(^6\)http://developer.nuance.com/
\(^8\)https://github.com/RestKit/RestKit
\(^9\)https://github.com/ccgus/fmdb
4.2 Question-Answering Module

The question-answering module used by ProntAPP is the same used in MedicineAsk. A major part of the development process was the adaptation of MedicineAsk to retrieve responses in a format other than HTML. Most mobile operating systems support the display of HTML, however we wanted to have more control over how the data is presented, and not simply display the retrieved HTML. This resulted in the development of an API, so that the mobile client applications can retrieve the response, process it, and display it in its own native user interface, like they were communicating directly with the database.

The MedicineAsk system is very coupled with the web-interface (meaning that the code related to user interface, business logic and database accesses in intertwined). Therefore, one of the challenges to modifying this Question-Answering module was to separate the presentation logic from the processing logic, so that we could change the data format retrieved by the Question-Answering module.

Also, many configuration files were hard-coded in MedicineAsk and its sub-libraries. The database accesses were also hard-coded and only available to MySQL databases (now also being available to SQLite databases). In this section, we present our implementation choices to deal with these issues, the API developed to publish MedicineAsk to the mobile applications and also some previously existing bugs that were found and fixed along the development process.

These modifications allowed us to have the same source code of MedicineAsk running on the web (essentially keeping MedicineAsk’s website intact) to also be used by the mobile applications. This approach will also facilitate further updates to the Question-Answering module, since any updates to the MedicineAsk’s NLI and NLP modules are transparently updated and used by the mobile applications, without any need of changes to the mobile applications’ source code.

4.2.1 Delegation Pattern

To solve the issues described earlier, we implemented a delegation pattern in MedicineAsk, mainly in the parts that are configuration-dependant (paths to configuration files, database hosts, usernames and passwords, etc...). This is intended so that any program that wants to use MedicineAsk can provide its own configurations, without having to recompile it with the appropriate paths. In software engineering, the delegation pattern is a design pattern in object-oriented programming where an object, instead of performing one of its stated tasks, delegates that task to an associated helper object. There is an Inversion of Responsibility in which a helper object, known as a delegate, is given the responsibility to execute a task for the delegator. This is further exemplified in Figure 4.3.

4.2.2 Domain Classes and Response Handlers

MedicineAsk did not have any domain classes. This means that the logic related to querying the database and displaying the results was intertwined. It was necessary to change this to be able to
expose MedicineAsk’s results through an API or through a simple method call. We created the relevant domain classes, and a generic Response Handler. A Response Handler is an abstract class that represents an entity that will deal with the response. As we stated earlier, MedicineAsk only returned the results in HTML. This needed to be modified, however without breaking the current web-interface of MedicineAsk. To achieve this, we created an abstract Response Handler which is called by the MedicineAsk system when building an answer. We then made the current HTML printer class a sub-class of the Response Handler, so that it can still print the HTML, and created another sub-class to handle the domain classes per-se and be able to return them via an API or a simple method call.

4.2.3 Bug Fixes

After having the mobile applications working with MedicineAsk’s server through the API we started noticing some bugs on the server, specifically on the type of responses retrieved.

For instance, MedicineAsk was incapable of answering questions that consisted of simple keywords (i.e. “Paracetamol”). After analyzing the system, we discovered that MedicineAsk was sometimes neglecting the SVM answer (simple questions did not go through the SVM, only keywords). We had to modify it in order to fix the bug, and MedicineAsk started answering keyword questions correctly.

After this, even the online version of MedicineAsk started displaying keyword questions correctly, however there seemed to be multiple results missing. We double-checked and crossed information with the MySQL database used by MedicineAsk and confirmed the problem. This was a bug with the result printer (both HTML and JSON) which was looping the SQL cursor to fill the response. The case was that this SQL cursor was advancing every two rows instead of row by row, due to a misplaced call to the function `cursor.next()`. This was also identified and fixed, and all the results were now being presented correctly.
4.3 Online Server API

To be able to expose MedicineAsk to other interfaces, an API was developed. This API works side-by-side with the current MedicineAsk deployment. This API was developed with Jersey\(^\text{10}\), an open source Web Services framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs.

An endpoint was exposed through this API called `analyze`. This endpoint accepts a GET request with a query parameter with the user’s question. After MedicineAsk’s NLI and NLP analysis and database retrieval this endpoints retrieves the data in a JSON format.

In listing 4.1 is an example of a JSON response from the Medicine Ask server when answering to the question “What are the indications of Paracetamol?” (“Quais as indicações do Paracetamol?”). The server identifies the medicines retrieving all the information about them.

```json
{
    "answers": [
        {
            "medicines": [
                {
                    "idMedicine": 364,
                    "name": "Takipirina Flashtab 250 mg",
                    "laboratory": "Angelini",
                    "packing": "Blister - 12 unidade(s)",
                    "composition": "250 mg",
                    "farmaceuticForm": "Comp. dispersivel",
                    "dispense": "MSRM",
                    "group": "Orais liquidas e semi-solidas",
                    "comparticipation": 0,
                    "PMU": 3.380000114440918,
                    "PVP": 0.2816999852657318,
                    "generic": 0,
                    "cheaperMedicine": false
                },
                ...
                {
                    "idMedicine": 404,
                    "name": "Panasorbe",
                    "laboratory": "Sanofi Aventis",
                    "packing": "Blister - 10 unidade(s)",
                    "composition": "500 mg",
                    "farmaceuticForm": "Supositrio",
                }
            ]
        }
    ]
}

\(^{10}\text{https://jersey.java.net/}\)
In the JSON response, the detailed information of the medicine corresponds to the medicine id on MedicineAsk's database, the name of the medicine, the laboratory, the type of packing and composition, farmaceutic form, dispense, group, if it is comparticipated by the government, the RRP (price) and PMU, and if the medicine is a generic or not. In addition to the medicines information, it also displayed information about chapters, the substance, indexes and the mapped questions, if present.

This JSON response is then parsed on Android and iOS by their respective libraries and displayed on screen using native views.

In addition to the analyze endpoint another endpoint was exposed called alarm_quantity_types which retrieves a JSON response with the alarm quantity types and dosages available for scheduling alarms in ProntAPP's alarm functionality. We go into further detail about these quantity types and dosages in Section 4.4.1.
4.4 Database

In this section, we discuss the implementation of both the online database and the offline databases (iPhone and Android), and we present their database schema.

4.4.1 Online Database

The online database available on the server is in MySQL format. We wanted to enhance the alarms functionality with some features, like choosing the type of medicine dosage and type (e.g. Take 1/2 Pill), therefore we had to extend this database with two new tables called `alarm_quantity_type` and `alarm_quantity_option`. The `alarm_quantity_type` table stores the type of alarm quantities (i.e. pill, applications) while the `alarm_quantity_option` stores the options for each type (i.e. 1 pill, 1/2 pill, 3/4 pill). The relation between these two new tables can be found on figure 4.4.

![Figure 4.4: Extensions to the Database](image)

The `alarm_quantity_option` table stores a foreign key to the `alarm_quantity_type` so that we can infer the options of each type. Figures 4.5 and 4.6 show the content of the tables so that their relation and meaning can be better presented.

<table>
<thead>
<tr>
<th>id_alarm_quantity_type</th>
<th>name</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Comprimido</td>
<td>Comprimido</td>
</tr>
<tr>
<td>2</td>
<td>Sol. Oral</td>
<td>ml</td>
</tr>
<tr>
<td>3</td>
<td>Gotas</td>
<td>Gotas</td>
</tr>
<tr>
<td>4</td>
<td>Aplicação Pomada</td>
<td>Aplicações</td>
</tr>
</tbody>
</table>

![Figure 4.5: Content of the “alarm_quantity_type” table](image)

From the content of these tables we can infer that we have four options to present in the alarm dosage type: Comprimido (pill), Sol. Oral (oral solution), Gotas (drops) and Aplicação Pomada (ointment application). We can also infer that the Comprimido type has five options for the quantity: 1, 3/4, 1/2, 1/4 and 2. Since the other quantity types do not have any quantity options, the mobile client applications will consider them an open field, while showing the desired unit.

These changes were made by creating a database model using MySQL Workbench

the necessary changes to the entity association model and then syncing the changes using MySQL Workbench’s built-in feature.

### 4.4.2 Android

The local database present in the Android mobile application is in the SQLite format. Although Android provides classes in their SDK to directly access the database, we chose to use an ORM (Object-relational mapping) library to further facilitate this data access and storage. The software library we chose to use is ActiveAndroid and is further described in Section 4.7.1.

Android’s local database stores all the information relative to the favorite medicines, and also stores information about the alarms.

### 4.4.3 iPhone

The iPhone SDK provides developers with a standard, near-transparent interface to a database file called Core Data. With Core Data, an app can define a database schema, create a database file, and create and manage record data. Core Data is hardware-agnostic. Runtime support is available on the same processors that MacOS X and iOS support. Plus, Core Data works directly with SQLite, the public-domain database engine bundled with MacOS X and iOS. Although the iPhone SDK provides this interface, we chose to interact directly with the database with a software library called FMDB. The reasoning for this choice was the ease of use of FMDB in relation to Core Data (since Core Data is intended to larger projects), since the developer can easily provide their own SQL queries and parse the data from SQL cursors directly.

iPhone’s local database stores all the information relative to the favorite medicines, and also stores information about the alarms.

### 4.5 Voice Recognition Module

#### 4.5.1 Android

Google provides an API to Android developers to facilitate the integration of mobile applications with its voice recognition system. The Android SDK contains a class called *SpeechRecognizer*.\(^\text{12}\)

\[^{12}\text{http://developer.android.com/reference/android/speech/SpeechRecognizer.html}\]
class provides access to the speech recognition service, which is an Android service which enables
access to the speech recognizer. All the voice recognition is performed on Google’s servers, since it is
a heavy task to perform on smartphones, and there is no publicly available SDK announced by Google
for using this service offline. Therefore, the Speech Recognizer API is not intended to be used for
continuous recognition, since it streams audio to remote servers to perform speech recognition, which
would consume a significant amount of battery and bandwidth. Since our application has no need for
continuous voice recognition (i.e., the voice recognition is only enabled when the user chooses to make
a question, and it is not active all the time), we can use it without these drawbacks.

4.5.2 iPhone

For the Voice Recognition Module on iPhone, we chose to use the Nuance Dragon Mobile SDK which
features the SpeechKit Framework. The SpeechKit framework is a high-level framework with two major
components for developers: the speech recognizer and the text-to-speech synthesizer. For the Voice
Recognition Module, we only used the speech recognizer. The framework carries out the following
processes:

- The audio component manages the audio system for recording and playback to give user feedback.
- The networking component manages the connection to the server and automatically re-establishes
timed-out connections.
- The end-of-speech detector determines when the user stops speaking and automatically stops
recording.
- The encoding component manages the streaming audio’s compression to reduce bandwidth re-
quirements and decrease latency.

SpeechKit follows a server-based architecture and relies on the Nuance speech server for voice
recognition and text-to-speech synthesis. For voice recognition, the SKRecognizer class sends audio
streams to the server which then returns a list of text results.

Like Google’s, this SDK is not intended to be used for continuous recognition, since it streams audio
to remote servers to perform speech recognition, which would consume a significant amount of battery
and bandwidth. Since our application has no need for continuous voice recognition, we can use it without
these drawbacks.

4.6 Alarms and Favorites

In this section we explain how we implemented both the Alarms and Favorites functionalities in the
Android and iPhone applications.
4.6.1 Android

Favorites

When a user is viewing the medicine detail and taps the favorite icon, a query is made to the local database using ActiveAndroid. This query verifies if the medicine is already present in the favorites table or not. If it is not present, a dialog is presented asking the user to confirm if he wants to add the medicine to the favorites. If the user confirms, a new row is inserted in the favorites table in the SQLite database storing all the information about this medicine.

If the medicine is already present in the favorites table, a dialog is presented asking the user to confirm if he wants to remove the medicine from the favorites. If the user confirms the action, the row is deleted from the SQLite database.

When the user enters the favorites screen, a query is made to the local database and all the rows of this table are then presented in an Android ListView using a custom Adapter.

Alarms

When checking a medicine’s information, the user can schedule an alarm by pressing the clock icon. This icon opens an Android custom AlertDialog. Before the dialog opens, a web-service call is made to the alarm_quantity_types endpoint to retrieve the possible dosage types and amounts. These are then presented in Android Spinners (dropdown list). The alarms are scheduled using Android’s built-in class called AlarmManager. Alarms based on the AlarmManager class give a way to perform time-based operations outside the lifetime of the application. An alarm can be scheduled with this class using setRepeating() method, which schedules an alarm that repeats with an specified interval, or simply scheduled using the set() method which schedules a one-time alarm. In our approach to this, when a user schedules, for example, a repeating Monday and Wednesday, two repeating alarms are scheduled with a repeat interval of one week. These alarms must be scheduled with different IDs so they do not overlap each other. Then, these IDs are stored in the local database in the SystemAlarms table so that they can be cancelled if turned off in the alarms section.

The AlarmManager itself class does not show the notifications that ProntAPP presents. It simply notifies a BroadcastReceiver that the alarm has fired. A broadcast receiver (short receiver) is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens. This means that we had to implement our own BroadcastReceiver which fires a notification when it is triggered.

4.6.2 iPhone

Favorites

When a user is viewing the medicine detail and taps the favorite icon, a query is made to the local database using FMDB. This query verifies if the medicine is already present in the favorites table or not. If it is not present, a dialog is presented asking the user to confirm if he wants to add the medicine to
the database. If the user confirms, a new row is inserted in the favorites table in the SQLite database storing all the information about this medicine.

If the medicine is already present in the favorites table, a dialog is presented asking the user to confirm if he wants to remove the medicine from the favorites. If the user confirms the action, the row is deleted from the SQLite database.

When the user enters the favorites screen, a query is made to the local database and all the rows of this table are then presented in an iPhone **TableView** using custom **UITableViewCell**s.

### Alarms

When checking a medicine’s information, the user can schedule an alarm by pressing the clock icon. This icon opens an iOS custom **View**. Before the dialog opens, a web-service call is made to the *alarm_quantity_types* endpoint to retrieve the possible dosage types and amounts. These are then presented using iOS’s library **ActionSheetPicker**, further described in the next sections. The alarms are scheduled using iOS’s built-in class called **UILocalNotification**. An alarm can be scheduled to repeat with a specified interval, or simply schedule a one-time alarm. In our approach to this, when a user schedules, for example, a repeating Monday and Wednesday, two repeating alarms are scheduled with a repeat interval of one week. These alarms must be scheduled with different IDs so they do not overlap each other. Then, these IDs are stored in the local database in the **SystemAlarms** table so that they can be cancelled if turned off in the alarms section.

### 4.7 Software Libraries

In this section we present the main libraries used in the development of the mobile applications and a brief description of all of them, providing insights on their value to mobile software developers.

#### 4.7.1 Android

The software libraries dependencies in Android were managed by using Gradle’s \(^{13}\) built-in dependency management system. Gradle is an advanced build toolkit that manages dependencies and is integrated into Android’s development IDE, Android Studio.

**Retrofit**

Retrofit is a type-safe REST client for Android built by Square. The library provides a powerful framework for authenticating and interacting with APIs and sending network requests with OkHttp.

This library makes downloading JSON or XML data from a web API fairly straightforward. Once the data is downloaded then it is parsed into a Plain Old Java Object (POJO) which must be defined for each “resource” in the response.

\(^{13}\)http://gradle.org/
This library was used to communicate with MedicineAsk’s API and retrieve the necessary JSON responses from the server.

**Active Android**

ActiveAndroid is an active record style ORM (object relational mapper, which allows the developer to save and retrieve SQLite database records without ever writing a single SQL statement. Each database record is wrapped into a class with methods like `save()` and `delete()`.

This library is used to access the local SQLite database containing information about favorite medicines and alarms.

**Butterknife**

In every Android application, you have to use the `findViewById()` method for each view in the layout that you want to use in your application’s code. However, as applications’ designs get more complex layouts, the call to this method becomes repetitive and this is where the Butter Knife library comes in.

The Butterknife, developed and maintained by Jake Wharton (Square Inc.), has annotations that help developers to instantiate the views from our activity or fragment. It also has annotations to handle events like `onClick()`, `onLongClick()`, etc.

This library was used to eliminate all the boilerplate code and make the code simpler and easier to read and maintain.

**4.7.2 iPhone**

The software libraries dependencies in iPhone were managed by using CocoaPods dependency management system, a dependency manager for Swift and Objective-C Cocoa projects. Although it is not directly integrated into iPhone’s development IDE (XCode) like Gradle is integrated into Android’s development IDE (Android Studio), this is a must have when a developer wants to integrate several different libraries, developed by other users, into their own iOS application.

**RestKit**

RestKit is a modern Objective-C framework for implementing RESTful web services clients on iOS and Mac OS X. It provides a powerful object mapping engine and a simple set of networking primitives for mapping HTTP requests and responses built on top of AFNetworking. It has an elegant, carefully designed set of APIs that make accessing and modeling RESTful resources very easy.

This library is used to communicate with MedicineAsk’s API and retrieve the necessary JSON responses from the server.

**FMDB**

FMDB is an Objective-C wrapper around SQLite, an alternative to using iOS’ Core Data.
This library is used to access the local SQLite database containing information about favorite medicines and alarms.

**MLPAutoCompleteTextField**

MLPAutoCompleteTextField is a subclass of UITextField that behaves like a typical UITextField with one notable exception: it manages a drop down table of autocomplete suggestions that update as the user types.

This library is used to implement the auto-complete functionality when typing a question in the search box.

**Nuance Dragon Mobile SDK**

The Dragon Mobile SDK provides speech services to developers, with features like speech recognition and text-to-speech functionality. We use this library to implement the voice recognition functionality in iOS.

**ActionSheetPicker**

ActionSheetPicker is an iOS library which shows a **PickerView**, allowing user to select from a number of immutable options. This is used to present the dosage types and quantities in the alarms section. There are 4 distinct picker view options: **ActionSheetStringPicker**, **ActionSheetDistancePicker**, **ActionSheetDatePicker**, and **ActionSheetCustomPicker**. For our project, we used the **ActionSheetStringPicker**.

**4.8 User Interface**

In this section, we present and discuss the final user interface of ProntAPP in both the Android and the iPhone versions.

**4.8.1 Main Screen**

The main screen of the application is presented in Figure 4.7 and Figure 4.8. This screen of the application provides a tab-based navigation throughout the main sections of the application: search, favorites and alarms. On the Android version the tabs appear on the top of the screen and on the iPhone version the tabs appear on the bottom of the screen with icons, following both platforms user design principles and guidelines. At the top of the screen both applications present a help icon. Other main difference is the placement of the title of the application, since in Android it is presented on the top left, and in iPhone it is presented in the top center of the screen.

In both platforms a help message is presented with instructions to the user, as seen in Figure 4.7. This message is also shown if there are no results for a given search.
Figure 4.7: Main Screen before performing a question

Figure 4.8 presents the user interface after a search. In the blue rectangle with white letters the application notifies the user of the question that was understood. Below that, a list is presented with sections for each active substance and a list of medicines.

The results list is clickable and, when a user selects an item from the list, he is presented with the medicine detail screen.

4.8.2 Medicine Detail Screen

The medicine details screen is presented in Figure 4.9. This screen is presented when a medicine is clicked in the search list. The screen presents the user with information about the medicine, in particular it shows the name of the medicine, the laboratory, the price, if the medicine is a generic or not, pharmaceutical form, grouping, coparticipation and PMU. On the top of the screen, a user can add the medicine to its favorites (or remove it) by pressing the heart icon, or he can add an alarm by pressing the clock icon. When a user clicks the clock icon, he is presented with the add alarm dialog.

There are no particular differences between these screens in Android and in iPhone, other than some recommended spacings. Also, in the iPhone version a back button is present on the screen since it is considered a good practice and iPhones do not have a physical back button. On Android, a user can return to the previous screen by pressing the physical back button of his device.
4.8.3 Add Alarm Dialog

The add alarm dialog is shown in Figure 4.10. This dialog is presented when a users clicks the clock icon in the medicine detail screen in order to add a new alarm for a given medicine. In the top of the dialog a user can select the type of dosage (i.e., pill, oral solution, etc.) and the quantity. Next, he can select if the alarm should repeat weekly and days. In the bottom of the dialog, the user can add multiple hours for the alarm.

There are some differences in the implementation and user experience of this screen in Android and in iPhone. In Android, the types of dosage and quantities are presented in dropdown lists called Spinners. In iPhone, there is no equivalent for this user control. Therefore, we had to use the software library ActionSheetPicker (detailed in Section 4.7.2) in order to present a list of selectable options. In addition, we also used this library to select the hours, while in Android the hours are selected through a native DateTimePicker dialog.

4.8.4 Alarms List Screen

The alarms list screen in shown in Figure 4.11. This screen is accessible by clicking the Alarms tab in the main screen. In this screen, a list of scheduled alarms is presented. The user can toggle them on or off, and when the alarm is disabled a dialog appears asking the user if he also wants to remove the alarm.
There are no particular differences between these screens in Android and in iPhone, since we were able to use similar user controls in both platforms.
Figure 4.10: Add Alarm Screen
Figure 4.11: Alarms List Screen
Chapter 5

Experimental Validation

This section describes the experiments we conducted to evaluate the ProntApp mobile application. The evaluation was divided into two parts: usability evaluation and server load evaluation. The objective of the usability evaluation is to help us understand our targeted audience and ensure that the application is fulfilling the user’s needs. On the other hand, the objective of the server load evaluation is to ensure that the server is handling the users requests properly, and also to detect bugs that may emerged during the development phase.

5.1 Usability Evaluation

In this section, we describe the usability evaluation process based on a survey. We begin by introducing the organization and setup of the questionnaire and then we describe an analysis of the answers to the questions we obtained regarding some of the questions posed to the users.

5.1.1 Questionnaire

In order to evaluate the usability of the developed application, we have collected the feedback from 26 users, ranging from common people to medical staff, which have answered a survey. The survey is presented in the Appendix A and it is divided into four sections.

The first section collects general information regarding the user, such as the user’s gender and age range. The second section contains a set of questions related to the user’s experience with mobile applications, such as the type of smartphone they own and the most used mobile applications. The third section of the survey contains questions regarding specific screens of the application, and the user can rate the usefulness, responsiveness and attractiveness of the designed application. The fourth and last section asks the user if he or she would install the application in the future, if there is any other operating system that he or she thinks the application should support (i.e. Blackberry, Windows Phone, etc.) and leave some observations that may improve the application.

The questionnaire is analyzed in such a way that the users fill in the first two sections of the questionnaire, and then they were asked to interact with the application (iOS or Android) in order to perform
three scenarios. No help was given during that time. Finally, the users were asked to fill the third and fourth sections of the questionnaire describing their opinion and impressions regarding the application.

5.1.2 Results: Characterization of users and devices

Among our 26 users, we had a very broad distribution of genders, ages and medical expertise. Figure 5.1 shows the distribution of the obtained answers regarding the gender. 57.7% were males while 42.3% were females.

Gender

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>15</td>
<td>57.7%</td>
</tr>
<tr>
<td>Female</td>
<td>11</td>
<td>42.3%</td>
</tr>
</tbody>
</table>

Figure 5.1: Gender Distribution of Users.

Figure 5.2 illustrated the age distribution of users: 19.2% of the users are between the 15-24 age range, 23.1% between the 25-34 age range, 26.9% between the 35-44 age range, 7.7% in the 45-54 age range, 15.4% in the 55-64 age range and 7.7% being more than 64 years old.

Age

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15-24</td>
<td>5</td>
<td>19.2%</td>
</tr>
<tr>
<td>25-34</td>
<td>6</td>
<td>23.1%</td>
</tr>
<tr>
<td>35-44</td>
<td>7</td>
<td>26.9%</td>
</tr>
<tr>
<td>45-54</td>
<td>2</td>
<td>7.7%</td>
</tr>
<tr>
<td>55-64</td>
<td>4</td>
<td>15.4%</td>
</tr>
<tr>
<td>64+</td>
<td>2</td>
<td>7.7%</td>
</tr>
</tbody>
</table>

Figure 5.2: Age Distribution of Users.

Figure 5.3 shows the distribution of the obtained answers regarding the profession. 30.8% of users are currently working in the medical field, while the remaining 69.2% have other professions.

Taking into account the results shown in Figures 5.1, 5.2 and 5.3, we conclude that we achieved a very good distribution in terms of gender, ages and area of expertise, which we think were very valuable in gathering data from the rest of the questionnaire.

In terms of devices used, he had 42.86% of users testing ProntAPP on an Android device, and
57.14% testing ProntAPP on an iPhone device. Figure 5.4 shows that, from the users testing the application on an Android device, 16.7% were using a device with an operating system version between 4.0 and 4.4 (Android Ice Cream Sandwich to Android KitKat) and the remaining 83.3% were using a device with an operating system version 5.0 or higher (Android Lollipop).

Among the users testing ProntAPP on an iPhone, 31.3% of them were using an iOS 8 version and 68.8% were using an iOS 9 version, however it is expected that much more people will be using iOS 9 since all devices running iOS 8 are upgradable to iOS 9. This distribution can be found in Figure 5.5.
When considering the amount of experience that our users had with mobile applications, only 11.5% have been using mobile applications for less than 1 year, while 30.8% have been using them between 1-2 years, 23.1% between 3-4 years and 34.6% have been using them for more than 4 years (see Figure 5.6).

![Pie chart showing years of expertise of users](image)

**Figure 5.6: Years of Expertise of Users.**

Figure 5.7 shows that, from our test audience, one user (3.8%) rarely uses mobile applications, while five users (19.2%) use mobile applications from time to time, eleven users (42.3%) use different applications every day and nine users (34.6%) use every day the same applications.

![Pie chart showing mobile app use frequency](image)

**Figure 5.7: Mobile App Use Frequency of Users.**

This means that our test audience was actually quite experienced using mobile applications, as our statistical data supports this statement.

### 5.1.3 Results: ProntAPP’s Main Screen

In terms of appearance and content organization, most of our users were pleased, only 3.8% of users rated a 3 out of 5, as seen on Figure 5.8. 46.2% of users rated a 4 out of 5 and the remaining 50% rated a 5 out of 5, bringing the average answer to a 4.46.

![Pie chart showing ease of performing voice question](image)

**Figure 5.9: Distribution of answers regarding ease of performing voice-based question.**

15.4% of users gave a 3 out of 5. 34.6% of users rated a 4 out of 5 and the remaining 50% rated a 5 out of 5, bringing the average answer rating to a 4.03. The majority of the users that rated a 5 were using the Android version of ProntAPP, while all the users that rated a 5 were using the iOS version of ProntAPP.
Regarding the ease of making a written question, Figure 5.10, shows that 26.9% of users rated a 4 out of 5, while the remaining 73.1% rated a 5 out of 5, turning the final average into 4.39.

In terms of the favorites functionality, 3.8% of the users rated the usefulness of this feature a 3 out of 5, 42.3% rated a 4 out of 5 and the remaining 53.8% rated a 5 out of 5, as seen on Figure 5.11. In relation to the alarms functionality, 23.1% of users rated a 4 out of 5 and 76.9% rated a 5 out of 5, as seen on Figure 5.12.

The ratings for the favorites functionality and for the alarms functionality showed that while users preferred the alarms functionality in relation to the favorites functionality, they saw value in both function-
alities and thought they were very useful.

Users also gave some valuable free text feedback in terms of user experience. One user stated that the word “mapped” (“mapeada”, in Portuguese) is not very correct for a regular user, being a somewhat technical word. Another user gave us feedback regarding the icon used to show that an alarm in the alarms list is a repeating alarm, since the intention of the icon is not very clear. Finally, another user stated that the search box should have a clear button so that when a user wants to perform another question he can just click the clear sign (i.e. a cross) instead of deleting all the text.

### 5.1.4 Results: ProntAPP’s Medicine Detail Screen

In terms of the medicine detail screen, we had mixed feedback from our test audience. Figure 5.13 shows that, in terms of the content and appearance of this screen, 3.8% of users rated a 2 out of 5, 3.8% rated a 3 out of 5, 50% of users rated a 4 out of 5 and the remaining 42.3% rated a 5 out of 5. This means that, while the majority of users thought the screen was organized and had a good appearance, some users gave a negative feedback in relation to this screen.

Regarding the ease of use in accessing information about medicines, 7.7% of users rated a 3 out of 5, 38.5% of users rated a 4 out of 5 and 53.8% of users rated a 5 out of 5 (as seen on Figure 5.14).

Considering the favorites and alarms features, the majority of users thought that they were easy to
use. Regarding the favorites functionality, seen on Figure 5.15 only 3.8% of users rated a 2 out of 5, while 30.8% rated a 4 out of 5 and 65.4% rated a 5 out of 5. In terms of the alarms functionality, 3.8% of users rated 3 out of 5, 26.9% rated a 4 out of 5 and the remaining 69.2% rated a 5 out of 5 (as shown on Figure 5.16). These results support the statement that this screen’s functionalities are very easy to use, however the data could be presented in a better way.

Rate how easy it is to consult information about medicines

<table>
<thead>
<tr>
<th>Ease of Use</th>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Difficult</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>7.7%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>38.5%</td>
</tr>
<tr>
<td>Very Easy</td>
<td>5</td>
<td>14</td>
<td>53.8%</td>
</tr>
</tbody>
</table>

Figure 5.14: Medicine Detail Screen Ease of Use.

Users also gave some feedback using a free text field on the questionnaire. Users had some concerns regarding the colors used in the separators, stating that the color contrast is not very good. A bug was found on iPhones 6 that prevented the users from adding an alarm. Furthermore, a user stated that
he did not know the meaning of some acronyms, so this should be explained in the application in the help section.

5.1.5 Results: ProntAPP

From our 26 users, 96.2% of them stated that they found the application useful and that they would download it to their phone. Only 3.8% of users said that the application was not useful at all and would not download it. This data can be seen on Figure 5.17 and Figure 5.18.

Did you find the application useful?

<table>
<thead>
<tr>
<th></th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>%</td>
<td>96.2%</td>
<td>3.8%</td>
</tr>
</tbody>
</table>

Figure 5.17: Usefulness of the Application Rating.

Thirteen users found that the application would be useful in other operating systems. In particular, 72.7% think that Windows Phone would be a suitable platform for ProntAPP, while 18.2% think that ProntAPP should be developed for BlackBerry devices in the future (as seen on Figure 5.19).

Our tests proved that the features that we added to the application were interesting to our users and proven to be useful. The majority of our users thought that the favorites functionality and the alarms functionality were very useful.

Some users also gave us some insights in terms of features that they want ProntAPP to have in the future. In particular, some users thought that having a map with the location of nearby pharmacies would be very useful.
5.2 Server Evaluation

5.2.1 Experimental Setup

In this section, we detail the experimental tests to evaluate the performance of the ProntAPP server. The tests were performed against the server’s RESTful API. We used the *Gatling Stress tool*¹ to perform the load tests. Gatling is a project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. It is a Scala-based, high performance load and stress test tool. In the context of Gatling, the tests are named *simulations* and are written in Scala². Each simulation is composed by a *scenario*, which represent users’ behaviours, composed by one or multiple requests.

To achieve reliable feedback, we created a scenario for each of the two different requests: a user performing a natural language search (which we refer to from now on as "natural language scenario") and a user performing a keyword-based search (which we refer to from now on as "keyword scenario").

In the natural language scenario, the question performed was: *What are the contra-indications of Paracetamol? (Quais são as contra-indicações do Paracetamol)*, while in the keyword scenario the search was simply *Paracetamol*. We executed these scenarios varying the number of users within 1-10 range, where each user performed the two requests. These tests were made with 1 user, 5 simultaneous users

---

¹[http://gatling.io/](http://gatling.io/)
and 10 simultaneous users.

The server is hosted on Amazon Web Services' Elastic Cloud Service (EC2) running on an EC2 small instance with 2GB RAM and 1 vCPU, while the database is hosted on Amazon Web Services' Redundant Data Storage (RDS) running on an RDS small instance as well.

### 5.2.2 Results

With 1 user our server handled the requests as we expected. In the natural language scenario, the server took 877ms to answer the request, while it took the server 1969ms to answer the request on the keyword scenario.

With 5 requests, we started to notice that the server was not handling concurrent requests properly. This was even more noticeable with ten requests, since the server was simply receiving all the requests and answering all of them at once.

<table>
<thead>
<tr>
<th>Requests</th>
<th>Executions</th>
<th>Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>OK</td>
</tr>
<tr>
<td>Global Information</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.1: Results of Natural Language Server Load Tests (5 simultaneous requests)

<table>
<thead>
<tr>
<th>Requests</th>
<th>Executions</th>
<th>Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>OK</td>
</tr>
<tr>
<td>Global Information</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.2: Results of Keyword Server Load Tests (5 simultaneous requests)

As Table 5.1 demonstrates, with the natural language scenario it took the server an average of 3640ms to answer each request, with a minimum of 3435ms and a maximum of 3947ms. The standard deviation was approximately 167ms. On the other hand, on the keyword scenario it took the server an average of 9370ms to answer each requests, having a minimum of 9356ms and a maximum of 9377ms, with a standard deviation of only 7ms, as seen on Table 5.2. As we can see, the averages were almost five times higher than on the single request scenario, which identifies a scalability issue on the server.

<table>
<thead>
<tr>
<th>Requests</th>
<th>Executions</th>
<th>Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>OK</td>
</tr>
<tr>
<td>Global Information</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.3: Results of Natural Language Server Load Tests (10 simultaneous requests)

<table>
<thead>
<tr>
<th>Requests</th>
<th>Executions</th>
<th>Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>OK</td>
</tr>
<tr>
<td>Global Information</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.4: Results of Keyword Server Load Tests (10 simultaneous requests)

With ten simultaneous requests the server performed proportionally to the five simultaneous requests, in terms of increasing the average response time. On the natural language scenario, demonstrated on Table 5.3, it took the server an average of 9234ms to answer each requests, the fastest request being answered in 9068ms and the slowest in 9492ms. The standard deviation for this scenario
was 138ms. The keyword scenario, seen on Table 5.4, had an average of 25918ms for each answer, with a minimum of 20783ms and a maximum of 27752ms, having a standard deviation of 2120ms. This means that the averages were almost ten times higher than on the single request scenario.

![Simultaneous Requests](image)

**Figure 5.20: Summarized Results of Server Load Tests**

A summarized comparison of the average response time can be found on Figure 5.20, which clearly demonstrates a scalability issue on the server, and an exponential trend more noticeable on the keyword scenario.

### 5.2.3 Conclusions

Our server load tests detected without a doubt that the server is not handling concurrent requests properly. The causes for this may be many: an unintended database lock while answering a request, a bad implementation of database queries (i.e. making multiple queries instead of JOIN queries or preloading the queries) or others. These load tests performed using the Gatling Stress Tool were very valuable in detecting this problem which will need to be addressed in the future to properly scale this platform to multiple users.
Chapter 6

Conclusions

This chapter describes the main conclusions obtained from the research and development of the ProntAPP mobile applications. Section 6.1 shows a summary of this thesis, as well as the contributions brought. Section 6.2 presents some limitations of our implementation as well as possible solutions to those limitations. We also mention some ideas on how ProntAPP could be further improved.

6.1 Summary and Contributions

In this document, we have proposed the development of ProntAPP, a mobile question-answering system for medicines. We started by presenting the objective, requirements, a brief summary of our proposed solution, contributions and the thesis outline. In Section 2, we reviewed the most relevant question-answering systems for medicines, existing Android and iOS medical mobile applications and state-of-the-art of voice recognition systems. Section 3 described ProntAPP’s architecture and its main components. In Section 4 we presented the implementation of ProntAPP: the question-answering module, online server API, database, voice recognition module, alarms and favorites functionality, the software libraries used and the user interface. And finally, in Section 5 we described the experiments conducted to validate ProntAPP and the results obtained.

As a result of this thesis, two native mobile applications were developed: one for Android and one for iPhone. The applications will be available for download on Google Play and AppStore. The main features of this application are the ability to answer natural languages in Portuguese about medicines, voice recognition using Google's voice recognition and Nuance Dragon, the ability to add a medicine as a favorite and the ability to add a repeating alarm for several hours of the day for a given medicine. This application is the only one currently available that has this functionalities, being innovative in its field.

Furthermore, the question-answering module of the application was implemented using MedicineAsk. In order to do this, we had to do some major code refactoring in the online version of MedicineAsk in order to develop an API so that the question-answering functionality could be accessed from outside clients (Android and iPhone). This API was developed with Jersey¹ which supports the development of

¹https://jersey.java.net/
6.2 Future Work

We have identified some limitations with the mobile applications of ProntAPP and mainly with the online server. In this section we propose some ideas for future solutions to these issues, and we also propose some new features based on the feedback we gathered from users. Sections 6.2.1 to 6.2.5 detail these issues and new features.

6.2.1 Server Scalability and Optimization

Our stress tests to the server detected that the server is not scaling correctly, which means that it is not handling concurrent requests properly and may be a problem if the applications are used by multiple users. We believe that the causes for this issue may be an unintended database lock while answering a request or a bad implementation of database queries which could be optimized (i.e. the server may be making multiple queries instead of JOIN queries or preloading them). Since the server used was an AWS EC2 small instance, it would also be interesting to evaluate the performance on an AWS EC2 medium instance.

6.2.2 Pharmacies Location

A feature that would be very valuable to add to the current version of the mobile applications of ProntAPP would be the location of nearby pharmacies. This information could be retrieved using FourSquare's API, which is a service that presents information about several services and locations. These pharmacies could be displayed in the application using Google Maps for Android and Apple Maps for iPhone.

6.2.3 Windows Phone Support

From our questionnaire we gathered that several people want the application to be available for Windows Phone. This means that a native application in C could be developed for Windows Phone, which would also automatically support all Windows devices (tablets and computers) since Microsoft embraced the Universal Apps methodology, in which all their devices (computers, tablets and smartphones) are running the same operating system.

6.2.4 Tablet Support

ProntAPP was developed for Android and iPhone smartphones, not taking into account the user interface of tablets. The applications could benefit from having a user interface specifically designed for tablets, taking into account their main pros in regards to smartphones, namely the screen size.

2https://pt.foursquare.com/
6.2.5 Natural Language Search Improvements

ProntAPP’s natural language server has some issues with some natural language questions. For example, the question "What are the medicines indicated for headaches?" ("Quais os medicamentos indicados para a dor de cabeça?") is badly interpreted as "What are the medicines for pain that do not have as precaution head" ("Quais os medicamentos para a dor que não têm como precaução cabeça"). These issues must be addressed to further improve the natural language functionality.

6.2.6 User Interface Improvements

From our questionnaire we gathered some small user improvements that could be made in both versions of ProntAPP, namely:

- The word "mapped" ("mapeada", in Portuguese) is not very correct for a regular user, being a somewhat technical word, and should be replaced for a more familiar one.

- The intention of the icon used to show that an alarm in the alarms list repeats every week is not very clear, and therefore it should be replaced.

- The search box should have a clear button so that when a user wants to perform another questions he can click the clear sign instead of deleting all the text.

- The color contrast in the separators is not very clear, and should be addressed.

- A list of acronyms and their meaning could be added to the help section.

6.2.7 Further Evaluation Tests

It would be interesting to measure the time taken by the users to perform the scenarios in the experimental tests to further evaluate the usability of the application. In addition, it would be valuable to evaluate the success rate of the natural language analysis to further improve the natural language recognition on the online server.
References


Appendix A

Evaluation Questionnaire
ProntAPP - Avaliação de Usabilidade

Este inquérito refere-se à aplicação ProntAPP, disponível para smartphones Android e iPhone. A aplicação foi desenvolvida como projecto final do Mestrado em Engenharia Informática e de Computadores no Instituto Superior Técnico (Campus da Alameda). Este inquérito demora aproximadamente 10 minutos a ser respondido. Deverá ter a aplicação instalada no seu telefone para que possa interagir com a mesma quando requisitado. Obrigado pela sua ajuda.

* Required

[1/4] Informação Geral

1. **Sexo**
   *Mark only one oval.*
   - [ ] Masculino
   - [ ] Feminino

2. **Idade**
   *Mark only one oval.*
   - [ ] 15-24
   - [ ] 25-34
   - [ ] 35-44
   - [ ] 45-54
   - [ ] 55-64
   - [ ] 64+

3. **É um profissional da área da saúde?**
   *Mark only one oval.*
   - [ ] Sim
   - [ ] Não


Por favor preencha estas questões antes de interagir com a aplicação.

4. **Caso tenha um telefone Android, qual é a versão do sistema operativo que está a usar?**
   Pode ver a versão do sistema operativo indo às Definições > Acerca.
   *Mark only one oval.*
   - [ ] 1.5 - 1.6
   - [ ] 2.0 - 2.3.3
   - [ ] 4.0 - 4.4
   - [ ] 5.0+
5. **Caso tenha um telefone iPhone, qual é a versão do sistema operativo que está a usar?**
   Pode ver a versão do sistema operativo indo às Definições > Acerca  
   *Mark only one oval.*
   - [ ] 6.x
   - [ ] 7.x
   - [ ] 8.x
   - [ ] 9.x

6. **Há quantos anos interage com aplicações para telemóveis?** *
   *Mark only one oval.*
   - [ ] Menos de 1 Ano
   - [ ] 1 - 2 Anos
   - [ ] 3 - 4 Anos
   - [ ] Mais de 4 Anos

7. **Com que frequência usa aplicações para telemóveis?** *
   *Mark only one oval.*
   - [ ] Raramente uso
   - [ ] Uso de vez em quando
   - [ ] Uso todos os dias diferentes aplicações
   - [ ] Uso todos os dias as mesmas aplicações

---

Para a próxima parte do questionário, por favor interaja com a aplicação conforme descrito nos próximos passos.
Deverá ter a aplicação ProntAPP instalada no seu smartphone Android ou iPhone.

**Cenário 1**
Faça uma pesquisa pelas indicações do ‘Paracetamol’ fazendo uma pesquisa por voz. De seguida, abra um medicamento e verifique se este é ou não um medicamento genérico.

**Cenário 2**
Faça uma pesquisa pelo dosagem do ‘Maxilase’ e adicione-o à sua lista de favoritos. De seguida, aceda à sua lista de favoritos.

**Cenário 3**
Aceda ao medicamento ‘Maxilase’ da sua lista de favoritos e marque um alarme para tomar 1/2 Comprimido todas as 2ªs Feiras às 8 da noite. De seguida, confirme que ficou agendado no ecrã de alarmes.

[3/4] **Avaliação de Usabilidade**
Por favor respondá a estas questões após usar a aplicação.

---

**Ecrã Principal**

Esta secção do questionário está apenas relacionada com o ecrã principal.

https://docs.google.com/forms/d/1jLM5r-pNT0BtxTCa09g91BMKeoulijh023qNflfXMbm0/printform
8. Avalie a aparência e a organização de conteúdo deste ecrã. *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Não atractivo e difícil de usar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. Avalie o grau de dificuldade de colocar uma questão por voz. *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muito Difícil</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. Avalie o grau de dificuldade de colocar uma questão por escrito. *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muito Difícil</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11. Avalie o grau de utilidade da lista de favoritos *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nada Útil</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12. Avalie o grau de utilidade da lista de alarmes *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nada Útil</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

13. Por favor adicione algum comentário extra que tenha a fazer em relação a este ecrã

----------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------

Ecrã de Detalhe de Medicamento

Esta secção do questionário está apenas relacionada com o ecrã de detalhes de um medicamento.
14. **Avalie a aparência e a organização de conteúdo deste ecrã.** *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Não atraente e difícil de usar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atractivo e fácil de usar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15. **Avalie a facilidade em consultar informação sobre medicamentos.** *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muito difícil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muito fácil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16. **Avalie a facilidade em adicionar um medicamento aos seus favoritos** *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muito difícil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muito fácil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17. **Avalie a facilidade em adicionar um alarme para um medicamento** *Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muito difícil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muito fácil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18. **Por favoradicione algum comentário extra que tenha a fazer em relação a este ecrã**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[4/4] **Feedback**

19. **Achou a aplicação útil?** *Mark only one oval.*

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://docs.google.com/forms/d/1jLM5r-pNToBtxTCa03g91BMKeouljT023qNfiXMbn0/printform
20. **Descarregaria esta aplicação para o seu smartphone?** *Mark only one oval.*
   - Sim
   - Não

21. **Acharia útil ter esta aplicação disponível em outros sistemas operativos? Se sim, quais?**
   *Deixe em branco caso ache que não.* *Mark only one oval.*
   - Windows Phone
   - Blackberry
   - Other: ..............................................................................................................

22. **Acha que a aplicação deveria ter mais funcionalidades? Se sim, quais?**
    ..............................................................................................................
    ..............................................................................................................
    ..............................................................................................................
    ..............................................................................................................
    ..............................................................................................................

**Obrigado**