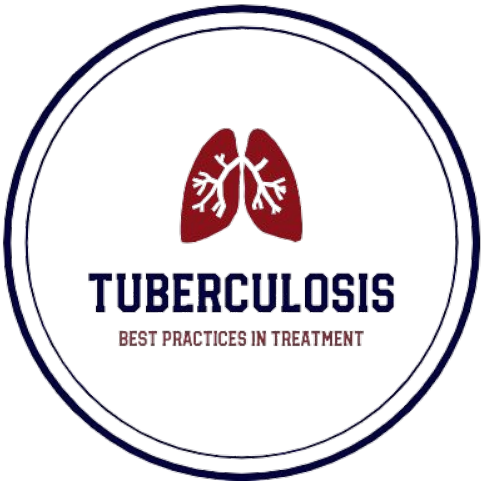
Implementation of an APP for supporting best practices in tuberculosis treatment

Shirley Fermino Rodrigues Fortes
Master in Network and Information Systems Engineering
Department of Computer Science
Faculty of Sciences of Porto University
2024



Implementation of an APP for supporting best practices in tuberculosis treatment

Shirley Fermino Rodrigues Fortes

Dissertation conducted as part of the Master's degree in Network and Information Systems Engineering.

Department of Computer Science

2024

Advisor
João Pedro Pedroso, Associate Professor, Faculty of Sciences

Co-advisor
Raquel Duarte, Full Professor, ICBAS & Integrated Member (PhD), ISPUP

Abstract

Tuberculosis (TB) is an issue of major concern worldwide, as it poses a serious health concern to the world's population, particularly in areas with poor health facilities. TB requires long, complicated antibiotic treatments that, if not completely followed, can increase the chance for the evolution of drug resistance. As new TB treatment guidelines emerges quickly, updating healthcare providers about the most current treatment guidelines is difficult to achieve, as offline methods (which rely heavily on direct interactions with specialized services) are inefficient.

That is the motivation for this thesis: to address this gap by developing an m-Health application together with the appropriate specification of the main characteristics and functionalities to support the dissemination of clinical guidelines in the management of TB patients.

The application will assist healthcare workers in obtaining the latest treatment guidelines to consult whenever required. Currently, when doctors have doubts, they typically consult specialists at referral centers, such as ISPUP[1], to seek answers to their questions. The aim of this app is to provide an alternative: through the app, healthcare workers can consult the latest questions asked, search the history, and if their question hasn't been answered, submit it for analysis by a moderator. Once the moderator provides a recommendation, it will be available on the app for all users.

This app will be of assistance in ensuring the implementation and extension of treatment regimens for TB among a host of health practitioners, supporting existing treatment plans. It empowers global health management by implementing Mobile Health (mHealth) technologies to fill out gaps in medical information delivery as recommended by the World Health Organization (WHO) "End TB Strategy" and improve TB treatment outcome globally.

Keywords: tuberculosis, global health, m-Health application, clinical guidelines, health management, healthcare workers, WHO End TB Strategy, medical information delivery, treatment regimens.

[1] Instituto de Saúde Pública da Universidade do Porto

Resumo

A Tuberculose (TB) é um problema de saúde global significativo, causando doenças e mortes generalizadas, particularmente em áreas com instalações de saúde inadequadas. O tratamento da TB envolve regimes antibióticos longos e complexos, e a falta de adesão completa a esses tratamentos pode levar à resistência aos medicamentos. Manter os profissionais de saúde atualizados sobre as novas diretrizes de tratamento da TB é desafiador, especialmente por meio de métodos offline tradicionais que dependem de interações diretas com serviços especializados.

Para enfrentar esse desafio, esta tese propõe o desenvolvimento de um aplicativo de m-Health projetado para disseminar as diretrizes clínicas mais recentes para o tratamento de pacientes com TB.

Este aplicativo ajudará os profissionais de saúde a acessar diretrizes de tratamento atualizadas sempre que necessário. Atualmente, os médicos frequentemente consultam especialistas em centros de referência, como o ISPUP, para obter orientações quando têm dúvidas. Este aplicativo visa oferecer uma alternativa: os profissionais de saúde podem consultar perguntas previamente respondidas, pesquisar no histórico e enviar novas perguntas para análise de um moderador. Uma vez que o moderador forneça uma recomendação, esta ficará disponível no aplicativo para todos os usuários.

Este aplicativo será útil na implementação e extensão dos regimes de tratamento da TB entre vários profissionais de saúde, complementando os planos de tratamento existentes. Ao integrar tecnologias de mHealth, este estudo visa preencher lacunas na entrega de informações médicas segundo a "Estratégia End TB" da Organização Mundial da Saúde (OMS), melhorando, assim, os resultados do tratamento da TB globalmente.

Palavras-chave: tuberculose, saúde global, aplicação m-Health, diretrizes clínicas, gestão da saúde, profissionais de saúde, Estratégia End TB da OMS, entrega de informações médicas, regimes de tratamento.

Acknowledgements

First of all, I would like to thank my family, especially my mother, Ariana. I would like to express my sincere appreciation to my loving fiancé, Renato, who always believed in me and stood by my side. I also thank my best friend Adélcia, for her support since day one.

Special appreciation goes to my grandparents, Fernanda and João Capisto (forever in my heart), who have always been a tremendous source of inspiration for me.

Last but not the least, for the final words of appreciation, I would like to thank my advisor, João Pedro Pedroso.

Dedico à minha mãe . . .

Contents

List of Figures

Listings

Acronyms

TB Tuberculosis

UI User Interface

IT Information Technology

MVC Model–View–Controller

SDK Software Development Kit

DR Drug Resistance

AI Artificial Intelligence

ML Machine Learning

APP Application

WHO World Health Organization

mHealth Mobile Health

CME Continuing Medical Education

IDE Integrated Development Environment

Chapter 1

Introduction

In this chapter, the problem is overviewed, the study's importance is explained along with goals for the proposed solution. It supports TB treatment with timely access to information, introducing an app as an innovative way of enhancing TB information dissemination.

1.1 Background

Despite medical advances, tuberculosis (TB) remains a lethal infectious disease, claiming millions of lives even in recent years. The World Health Organization (WHO) has recently approved several initiatives to tackle the global TB crisis, and the new "End TB Strategy" of the WHO is one of them, designed with a view to the new sustainable development agenda of the next 15 years. It provides a road map for nations to decrease the incidence rate of tuberculosis by 80 percent, deaths due to tuberculosis by 90 percent, and eliminate the high cost on the affected households through the year 2030. The Strategy is not a 'one size fits all' approach, and it must be noted that the success of the Strategy lies in its adoption in a way that is suitable for each country's context [1].

This is according to recent findings made by the WHO, which reveal that tuberculosis is among the top ten diseases and ailments that cause deaths in the world, with millions of people being affected by new cases of tuberculosis every year [12]. Tuberculosis is an ailment that results from the pathogenic bacteria Mycobacterium tuberculosis, with a proclivity to affect the respiratory tracts, mainly the lungs, and a potential to transverse to any other region of the body. TB patients undergo long-term management that requires compliance with multiple antibiotics, as the development of drug resistance is potential if the patient fails to follow strict medication protocols [2].

This is an important issue because, even with better medical knowledge and well-formulated and uniform treatment approaches, an important problem concerns bringing new knowledge to caregivers. Currently, in many healthcare practices, especially those from developing countries or

those with restricted access to healthcare information, physicians and other healthcare workers do not have easy and convenient access to the latest guidelines and recommendations regarding TB treatment. This is compounded by the dynamic nature of knowledge delivery, with new information flooding the system and calls for frequent change due to developed strains and resistance patterns.

The present method to fill up this information deficit usually entails healthcare workers reaching out to these specific services, for example, the Institute of Public Health of the University of Porto (ISPUP), to get information on their queries. Although this method is informative, it is very slow in generating information, which may lead to more time being taken to make pertinent decisions, hence compromising the care of the patients. Therefore, there is an evident need for a more integrated and less time-consuming way to offer professionals-at-large authoritative data and suggestions.

1.2 Problem Statement

This thesis focuses on the lack of an efficient mechanism for delivering new practices concerning TB treatment. Currently, when doctors have doubts, they typically consult specialists at referral centers, such as ISPUP, to seek answers to their questions. The fact that it heavily relies on direct communication with specialized services to obtain the information necessary for implementing new practices in the clinic is hardly efficient and poses limitations to theory-to-practice translation in a timely manner. Up to now there is no available, easily accessible and centralized platform for healthcare workers in the region that enables them to easily retrieve the latest guidelines, documented past searches, as well as expert advice on the treatment of TB.

1.3 Objectives

Hence, the research focus of this thesis is on developing a Mobile Application (APP) that will assist in the provision of information on TB best practices. The specific objectives include:

1. Creating an efficient and user-oriented application through which the healthcare workers could easily find the updated TB treatment guidelines.
2. Providing users with the option to submit new questions through the app, which will be moderated and answered by experts.
3. Guaranteeing that all recommendations given by the moderators are made available to all potential users.
4. Assessing whether the app contributes to the availability and convenience of TB treatment information for stakeholders.

1.4 Significance of the Study

This study will thus contribute to the public health by ensuring a smooth flow of relevant information concerning the treatment of TB. It has the potential of helping the healthcare professions to get information quickly and easily update the guidelines in the treatment of the patients, hence enhancing the fight against TB and support the overall efforts of TB control and elimination.

It is for this reason that the creation and usage of this app support international health goals as is seen in the strategies developed by the WHO's "End TB Strategy", highlighting integration, patient-centered care and prevention. In this regard, this particular application can be helpful in filling gaps in information sharing, since it helps disseminate best practices as fast as possible to various health care agents that are involved in the fight against the TB disease.

Chapter 2

State of the Art

This chapter sets the foundation for understanding the current landscape on information dissemination tools for TB treatment, highlighting the gaps and challenges that the proposed app seeks to address.

2.1 Overview of Tuberculosis

Tuberculosis is an infectious disease that is largely transmitted from person to person through the air, namely by droplets created and expelled by the infected person when coughing, talking, or sneezing. It is caused by Mycobacterium tuberculosis, also known as Koch's bacillus. It is a serious, but potentially curable, disease [14] .

The World Health Organization (WHO) has several studies, one of which is an estimation that the disease kills about 1.1 million people annually. For example, in 2021, the global death rate was 1.4 million, and the total number of new infection cases was about 10 million [11].

The source of treatment for TB is predominantly through the use of antibiotics for an average of 6–9 months, under a method known as direct observation therapy. Isoniazid, rifampicin, ethambutol and pyrazinamide are some of the drugs used in the treatment of tuberculosis [10]. This treatment regimen is very important in ensuring that patients do not develop Drug Resistance (DR) TB (DR-TB) which may occur if they do not take their medications strictly. DR-TB presents the need for constant additions to the traditional treatment procedures and the communication of these additions among the medical practitioners in the shortest time possible.

2.2 Existing Solutions for Disseminating Medical Information

There are numerous techniques and media that can be employed in the process of distributing medical knowledge to professionals at the moment. These are printed guidelines, text-based

databases, activities under Continuing Medical Education (CME), and digital health tools. Each method has its advantages and limitations as illustrated in the mind map in Figure 2.1.

Printed Guidelines: Traditionally, the use of printed media was the most common way of providing guidance to medical professionals. Although they are general, and many people consider them to be rather authoritative, they do not fit well with rapidly evolving fields like TB treatment, on account of their static nature. A limitation that can be noted in printed materials is the infrequency of updating the information presented, which can result in a lack of the latest information for practitioners.

Online Databases: Sources such as PubMed and the Cochrane Library also present extensive information on medical research and guidelines. However, these resources could be tedious and, at times, time-consuming to search for and access, especially for healthcare providers working under a lot of stress.

Continuing Medical Education (CME) Programs: CMEs provide formatted learning sessions to healthcare professionals so that they can access the latest developments. However, they demand much time alignment and may not be available to practice or practitioners in some rural or constrained circumstances.

Digital Health Tools: Current technological enhancements have called for the creation of other effective health technologies, including mobile applications and online portals that provide real-time medical information. These tools have the benefit of being available to the users all the time and can updated frequently, thus overcoming some of the challenges faced by traditional methods [4].

Figure 2.1: Medical Knowledge Distribution Methods

2.3 Technological Frameworks in Healthcare

Technological systems in the provision of healthcare have radically transformed the way health details are shared in the community. Key technological advancements include:

- **Mobile Health (mHealth):** mHealth is the utilization of mobile devices in the delivery of healthcare practices or services. Mobile applications have especially been featured as versatile technologies in the provision of care, where features such as monitoring, data capture, education, and information are possible. The nature of mHealth solutions, where they fit well on smartphones, which are easily portable, makes them very effective in enhancing the delivery of healthcare [5, 6, 17].
- **Cloud Computing:** Cloud computing offers cost-effective and flexible approaches for the hosting of storage needed by healthcare applications [7].
- **Artificial Intelligence (AI) and Machine Learning (ML):** Uses of Artificial Intelligence and machine learning are being integrated into various healthcare solutions, mainly in analyzing, recommending, predicting or even making decisions. These technologies can be adopted for the enhancement of medical information systems by referencing the content offered to users and their usage of the systems [8, 9].
- **Blockchain:** Due to the decentralized and shared ledger nature of blockchain technology, its capability to provide protection and accountability for medical information is highly beneficial. A major application of this technology can be observed in the reliability of information associated with medical guidelines and suggestions [13, 18].

Figure 2.2 illustrates the available technological systems in the provision of healthcare.

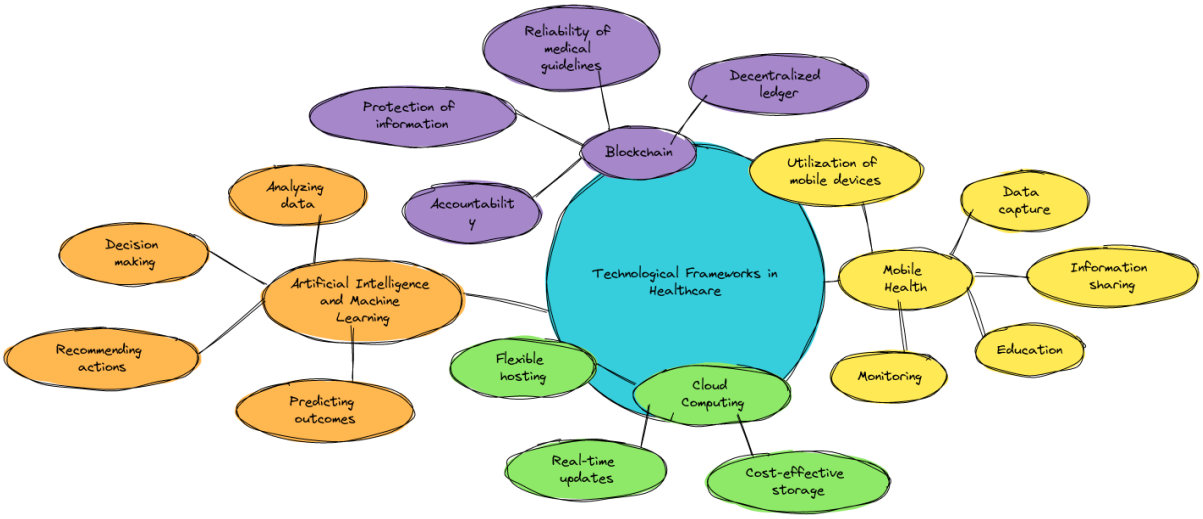


Figure 2.2: Technological Frameworks in Healthcare

2.4 Gap Analysis

Despite the availability of various tools and methods for disseminating medical information, significant gaps remain in ensuring that healthcare providers have timely access to updated TB treatment guidelines. Key gaps identified in the current solutions include:

- **Real-Time Updates:** Many existing methods, such as printed guidelines and traditional CME programs, do not provide real-time updates, leading to delays in the dissemination of the latest information.
- **Accessibility:** Online databases and CME programs, while valuable, may not be easily accessible to all healthcare providers, particularly those in remote or resource-limited settings.
- **Interactivity:** There is a lack of platforms that allow for interactive engagement, where healthcare providers can ask questions and receive expert recommendations promptly.
- **Centralization:** Current solutions often lack a centralized platform that consolidates the latest guidelines, historical queries, and expert recommendations in one accessible location.
- **Integration with Clinical Workflows:** Existing solutions may not be well integrated into the daily clinical workflows of healthcare providers, making the access and use of guidelines burdensome.

The use of smartphones and tablets by health workers is noted to be on the rise, and the article by Codyre (2014) [3] presents a view on how mHealth apps could offer point-of-care access to current practice resources which may enhance accessibility and may support real-time updates. But it also points to issues concerning mHealth, including at the same time the adequate evaluation of apps and data protection issues. Hence, despite the potential that mHealth holds in mitigating some of the gaps above in relaying TB treatment guidelines, there is a need for additional innovation in effectiveness, accessibility, and proper means of applying such systems.

Hence, one must note that the work of Codyre (2014) is helpful but at the same time, in doing so, it is necessary to point to the breakthrough in the development of mHealth in the last ten years. Some of these obstacles still exist, but problems and opportunities provided by the mHealth initiative will continue to present solutions for the gaps in disseminating TB treatment recommendations. It is highly suggested, however, that more research and innovations are required to enhance the mHealth possibilities in terms of access, use, and appropriate application.

2.5 Conclusion

The current literature and the technological system also assist on the mission to end tuberculosis and demonstrate that it is possible to enhance the method utilized to disseminate its treatment.

The idea of using an app means developing an environment where actual updates on associated information are available and easy to access. The engrossment of available information regarding the treatment of TB can be a great advantage to healthcare providers in furthering the access and utilization of newer and probably better methods of managing TB. This study seeks to conceptualize such an app to address TB while focusing on the prospects of mHealth for enhancing TB treatment rates.

Chapter 3

Methodology

The methodology chapter outlines the systematic approach taken to develop a mobile application for “normal users” and a website for moderators. This separation is strategic, as it leverages the advantages of different platforms to meet the specific needs of each user group: for mobile users, convenience and navigability were valued; while for moderators, the main concern is the effectiveness and functionality when they are using computers. The techniques employed to realize the set goals are explained as well as the research and development path taken. Once this stage is completed, chapter 4 on Implementation shall give insight on how these functionalities will be put into practice.

3.1 Requirements Analysis

The purpose of this requirements analysis is to describe in detail the functional and non-functional requirements of the intended application. Healthcare professionals will be able to obtain updated information, access a searchable archive of previous inquiries and advice, and make fresh inquiries through the app. We will consider two types of users: Normal Users and Moderators.

Normal Users: These are healthcare workers who will use the mobile app to submit questions and search the history of previous recommendations. The main requisites of the mobile application is an easy way to submit questions, receive notices, as well as recommendations and answers provided by moderators.

Moderators: These are healthcare workers with special permissions and the expertise to answer users’ questions and provide recommendations. The website for moderators required some essential features that include the ability to review and reply to user questions and managing user roles.

3.1.1 Functional Requirements

User Management

- Users must be able to register and create an account.
- Users must be able to log in and log out securely.
- Users must view their profile information.
- Moderators must be able to assign moderator rights to other users.

Question Management

- Users must be able to submit new questions.
- Moderators must be able to review, answer, or reject questions.
- Users must be able to search for previously asked questions and recommendations.

Notification System

- The system must send notifications to users about updates or responses.
- Users must be able to view and manage notifications.

Article Viewing

- Users must be able to view all answered questions (articles).
- The system must display a list of all answered questions.
- Users must be able to select and view the details of an answered question.
- Users must be able to bookmark articles.

Search Functionality

- Users must be able to search for questions and recommendations using keywords.
- The system must display search results relevant to the entered keywords.

Bookmark Management

- Users must be able to bookmark articles for future reference.
- Users must be able to view a list of their bookmarked articles.
- Users must be able to remove bookmarks.

3.1.2 Non-Functional Requirements

Performance

- The app should load and respond to user actions within 2 seconds.
- The search functionality should return results within 1 second.

Security

- User data must be encrypted.
- The system must implement robust authentication and authorization mechanisms.

Usability

- The app should have a user-friendly interface that is intuitive and easy to navigate.

Maintainability

- The codebase should be modular and well-documented to facilitate maintenance and future development.
- The system should support easy deployment of updates and new features.

3.2 Use Cases

The "Use Cases" section is intended to define and describe all the interactions of the users with the application. In this section, explaining how the app's features address user needs and requirements are shown through specific scenarios that include the interaction of the different users (healthcare professionals - normal users, moderators). The Figure 3.1 represents the Use Case diagram.

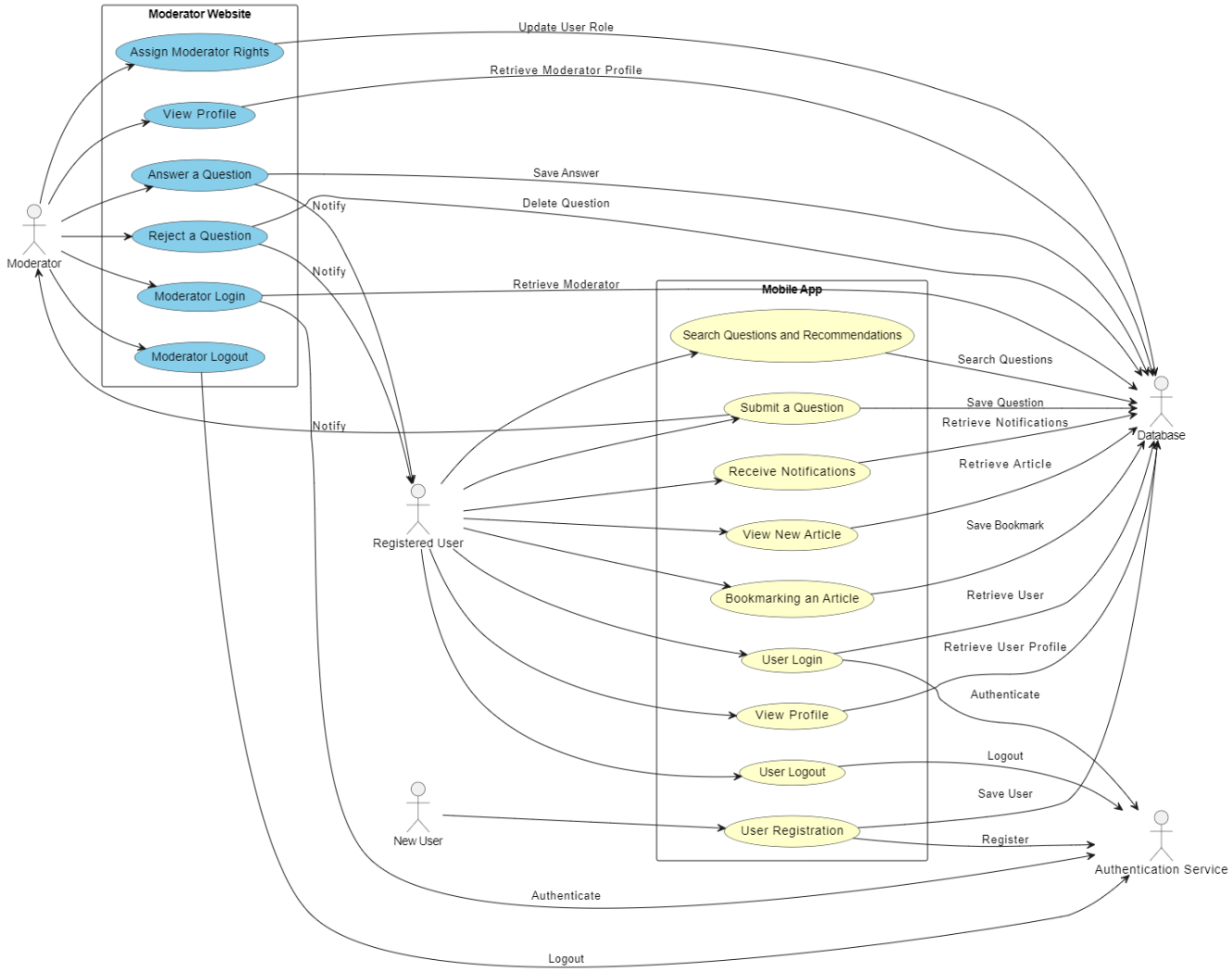


Figure 3.1: Use Case Diagram

3.2.1 Mobile App Use Cases

3.2.1.1 Use Case: User Registration

- **Actors:** New User
- **Description:** A new user registers for an account on the app.
- **Preconditions:** The user has installed the app and opened it.
- **Post-conditions:** The user is registered and can log in.
- **Main Flow:**
  1. The user selects the "Sign Up" option.
  2. The user enters required details (username, email, password).

3. The system validates the entered details.
4. The system creates a new user account.
5. The user receives a confirmation email.

3.2.1.2 Use Case: User Login

- **Actors:** Registered User
- **Description:** A registered user logs into their account.
- **Preconditions:** The user has a registered account.
- **Post-conditions:** The user is logged in and has access to the app's features.
- **Main Flow:**
  1. The user selects the "Login" option.
  2. The user enters their email and password.
  3. The system validates the credentials.
  4. The system grants access to the user's account.

3.2.1.3 Use Case: Submit a Question

- **Actors:** Registered User
- **Description:** The user submits a question about TB treatment.
- **Preconditions:** The user is logged in.
- **Post-conditions:** The question is submitted for review by moderators.
- **Main Flow:**
  1. The user navigates to the "New" section.
  2. The user enters the question details (title, description).
  3. The user submits the question.
  4. The system validates the question details.
  5. The system records the question.

3.2.1.4 Use Case: Search Questions and Recommendations

- **Actors:** Registered User
- **Description:** The user searches for previously asked questions and recommendations.
- **Preconditions:** The user is logged in.
- **Post-conditions:** The user views search results.
- **Main Flow:**
  1. The user selects the "Search" tab.
  2. The user enters search keywords.
  3. The system displays search results matching the keywords.
  4. The user selects a result to view detailed information.

3.2.1.5 Use Case: Receive Notifications

- **Actors:** Registered User
- **Description:** Users receive notifications about updates or responses.
- **Preconditions:** The user is logged in and has enabled notifications.
- **Post-conditions:** The user views the notification.
- **Main Flow:**
  1. The system generates a notification for an update or response.
  2. The system sends the notification to the user.
  3. The user views the notification with relevant information.

3.2.1.6 Use Case: View Article

- **Actors:** Registered User
- **Description:** The user views an answered question (article) that was posted on the app.
- **Preconditions:**
  1. The user is logged in.
  2. There are answered questions (articles) available.
- **Post-conditions:** The user views the details of the answered question.
- **Main Flow:**

1. The user navigates to the "Home" section.
2. The system displays a list of answered questions (articles).
3. The user selects a question from the list.
4. The system displays the details of the answered question, including the question, answer, and timestamp.
5. The user can bookmark the article for future reference.

3.2.1.7 Use Case: Bookmarking an Article

- **Actors:** Registered User
- **Description:** The user bookmarks an article (answered question) for future reference.
- **Preconditions:**
  1. The user is logged in.
  2. There are articles available to bookmark.
- **Post-conditions:** The selected article is bookmarked and can be accessed from the user's "My Saved" list.
- **Main Flow:**
  1. The user views an article (answered question) in the app.
  2. The user selects the "Bookmark" icon associated with the article.
  3. The system adds the article to the user's bookmarks ("My Saved").

3.2.1.8 Use Case: View Profile

- **Actors:** Registered User
- **Description:** User views profile information.
- **Preconditions:** The user is logged in.
- **Post-conditions:** User is able to see its profile information.
- **Main Flow:**
  1. The user navigates to the "Profile" section.
  2. The system returns the user's profile information.

3.2.1.9 Use Case: View Bookmarked Articles

- **Actors:** Registered User
- **Description:** The user sees all articles it has bookmarked.
- **Preconditions:**
  1. The user is logged in.
  2. There are articles bookmarked by the user.
- **Post-conditions:** The user has access to all its bookmarked articles on the “My Saved” list.
- **Main Flow:**
  1. The user navigates to the "Profile" section.
  2. The user selects the “My Saved” option.
  3. The user sees a list of all its bookmarked articles.

3.2.1.10 Use Case: View User’s Questions

- **Actors:** Registered User
- **Description:** The user sees all questions it has submitted and the corresponding details.
- **Preconditions:**
  1. The user is logged in.
  2. The user has questions that were submitted.
- **Post-conditions:** The user has access to all its questions on the “My Questions” list.
- **Main Flow:**
  1. The user navigates to the "Profile" section.
  2. The user selects the “My Questions” option.
  3. The user sees a list of all questions it has submitted in the system and the corresponding details (title, description, status, timestamps).

3.2.1.11 Use Case: User Logout

- **Actors:** Registered User
- **Description:** A registered user logs out of their account.
- **Preconditions:** The user is logged in.

- **Post-conditions:** The user is logged out and returned to the login screen.
- **Main Flow:**
  1. The user navigates to the "Profile" section.
  2. The user selects the "Logout" option.
  3. The system ends the user session.
  4. The system returns the user to the login screen.

3.2.2 Moderator Website Use Cases

3.2.2.1 Use Case: Moderator Login

- **Actors:** Moderator
- **Description:** A moderator logs into their account.
- **Preconditions:** The moderator has a registered account and moderator privileges.
- **Post-conditions:** The moderator is logged in and has access to the moderator's website.
- **Main Flow:**
  1. The moderator selects the "Login" option.
  2. The moderator enters their email and password.
  3. The system validates the credentials.
  4. The system grants access to the moderator's account.

3.2.2.2 Use Case: Answer a Question

- **Actors:** Moderator
- **Description:** A moderator reviews and answers a submitted question.
- **Preconditions:** The moderator is logged in and has moderator privileges.
- **Post-conditions:** The question is answered and the response is stored in the system.
- **Main Flow:**
  1. The moderator navigates to the "New Questions" section.
  2. The moderator selects a question to review.
  3. The moderator enters the answer and submits it.
  4. The system records the answer and notifies the user who submitted the question.

3.2.2.3 Use Case: Reject a Question

- **Actors:** Moderator
- **Description:** A moderator reviews and rejects a submitted question.
- **Preconditions:** The moderator is logged in and has moderator privileges.
- **Post-conditions:** The question is rejected and deleted from the system.
- **Main Flow:**
  1. The moderator navigates to the "New Questions" section.
  2. The moderator selects a question to review.
  3. The moderator rejects the question.
  4. The system deletes the question and notifies the user who submitted the question.

3.2.2.4 Use Case: Assign Moderator Rights

- **Actors:** Existing Moderator, New Moderator (User)
- **Description:** An existing moderator assigns moderator rights to another user.
- **Preconditions:**
  1. The existing moderator is logged in and has moderator privileges.
  2. The user to be assigned as a moderator has a registered account.
- **Post-conditions:** The selected user is granted moderator rights and can perform moderator functions.
- **Main Flow:**
  1. The existing moderator navigates to the "List of Moderators" section.
  2. The moderator adds the email of the user to be assigned moderator rights.
  3. The system updates the user's role to moderator by adding the user's info to the "Moderators" collection in the database.

3.2.2.5 Use Case: View Profile

- **Actors:** Moderator
- **Description:** Moderator views profile information.
- **Preconditions:** The moderator is logged in.
- **Post-conditions:** Moderator is able to see its profile information.

- **Main Flow:**
  1. The moderator navigates to the "Profile" section.
  2. The system returns the moderator's profile information.

3.2.2.6 Use Case: Moderator Logout

- **Actors:** Moderator
- **Description:** Moderator logs out of their account.
- **Preconditions:** The moderator is logged in.
- **Post-conditions:** The moderator is logged out and returned to the login screen.
- **Main Flow:**
  1. The moderator navigates to the "Profile" section.
  2. The moderator selects the "Logout" option.
  3. The system ends the moderator session.
  4. The system returns the moderator to the login screen.

3.3 Development Tools and Technologies

This section provides information on the tools and technologies that were employed in the development of the mobile and web applications.

3.3.1 Programming Languages

The development of both the mobile and web applications was carried out using the following programming language:

- **Dart:** Dart is the primary programming language used for the development of the applications. Dart is optimized for building high-performance, cross-platform mobile and web applications. The choice of Dart was influenced by its compatibility with Flutter, which provides a single codebase for both mobile and web applications. Having previously used Dart in coursework, I had a good grasp of its syntax and features. This familiarity reduced the learning curve and allowed for a more efficient development process.

3.3.2 Frameworks

The applications leverage the following framework:

- **Flutter:** Flutter was chosen as the framework for this application due to its numerous advantages and my prior experience with it in another coursework project. Flutter is an open source UI kit by Google. It is used to build apps for Android, iOS, Linux, Mac, Windows, Google Fuchsia and web, from a single codebase. It has a large amount of pre-designed widgets and tools to make it easier to build beautiful and responsive User Interface (UI). Moreover, Flutter has a massive community that presents many resources and tutorials along with constant improvements and the best practices in development. The rendering engine of the framework used to support the collection is highly optimized to give good impact animations and UI rendering. This together with my previous course and successful attempt of using Flutter makes the choice to be perfect for this application.

3.3.3 Backend Services

The backend services used in the development of the applications include:

- **Firebase:** Firebase is a platform developed by Google for creating mobile and web applications. It provides a variety of tools and services to help with the development, including authentication, cloud storage, and real-time databases. Working on the Flutter, Firebase facilitates and optimizing process of developing since it is deeply integrated into the framework, and Flutter plugins for Firebase improve the process of integration of backend capabilities into the application.
  - **Firebase Authentication:** Firebase Authentication provides backend services, easy-to-use Software Development Kit (SDK), and ready-made UI libraries to authenticate users to the app. It also allows users to enter their credentials in the form of email and password, phone number, as well as federated login services from Google and Facebook. For the tuberculosis application, email and password is the only option used.
  - **Firestore:** Firestore database is a NoSQL document that is built into Firebase. It is employed in the storage and synchronization of real-time data across user applications. The choice of Firestore was in terms of its scalability, real-time data synchronization, and seamless integration with Firebase Authentication. Firestore is used for storing and syncing data in real-time. Data such as user, moderator, questions, notifications, and bookmarks are stored in Firestore collections.

3.3.4 Development Environment

The development environment setup is as follows:

- **Integrated Development Environment (IDE):**
  - **Visual Studio Code (VS Code):** VS Code is Microsoft's free, lightweight yet powerful code editor, which is rather popular and recognized in the Information Technology (IT) field. VS Code was the main instrument for writing, debugging and testing Dart code for the Flutter applications. My choice of Visual Studio Code (VS Code) was primarily driven by my familiarity with its versatile and powerful features.
- **Version Control System:**
  - **Git:** Git is a distributed version control system that is important in tracking the alterations made to a source code as the development of a particular software goes on.
  - **GitHub:** GitHub is an online repository which utilizes the Git in its functioning. It hosts software development and version control with the help of Git. GitHub was applied as the code repository hosting service and for version control.

3.3.5 Design Tools

The applications leverage the following design too:

- **Figma:** Figma is a web-based design tool used for creating mockups and prototypes of the user interface (UI) for both the mobile and web applications. Having prior experience with Figma, I was able to leverage my existing skills to quickly start the design process, reducing the learning curve and enhancing productivity.

These tools and technologies were effectively used to improve the development cycle to help in the coding, testing, as well as deployment of the applications. It allowed a highly reliable, expandable, and sustainable design, that takes into account the needs of the health care givers and the moderators using the app.

3.4 User Interface Design

3.4.1 Design Principles

The design of the user interface (UI) for both the mobile and web applications was guided by several key principles to ensure usability, accessibility, and an overall positive user experience.

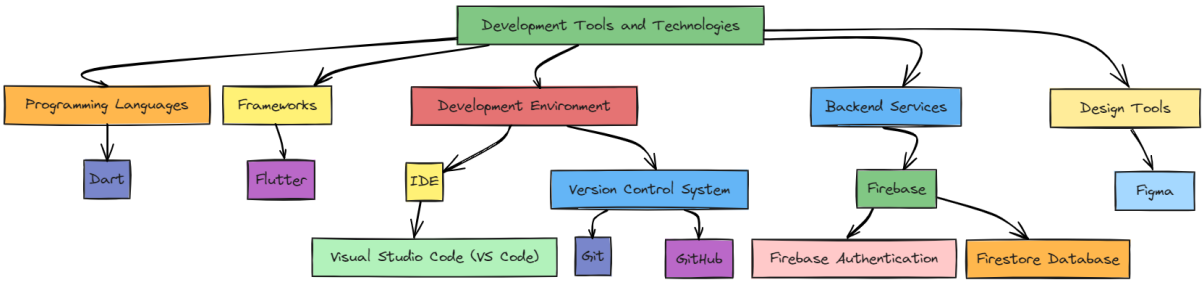


Figure 3.2: Development Tools and Technologies

Simplicity: This has helped shape the user interface of the application to be very simple, hence not requiring its users to undertake a steep learning curve. To eliminate any superfluous components and attain a clutter-free approach to the application's key components, some features are eliminated.

Consistency: Repetition of colors, fonts and layout are kept constant at certain sections of the application so as to give a consistent feel to other sections of the application. The navigation components should be placed in the same location with the action buttons as this makes it easy for a user to know or predict on how to go about using the app.

Responsive Design: This has pulled through the responsiveness of the application UI thus supporting the application across devices and screens. This is especially important to the mobile app, which should run on different smartphone screen sizes.

Feedback and Confirmation: The application offers a response to the users and their actions like a form submitting and a button click instantly. Confirmation messages are displayed for critical actions to prevent errors.

User-Centered Design: User feedback was considered during the design process to ensure that the UI meets the needs and preferences of the target audience.

3.4.2 User Interface Components - Mockups

Prior the technical development of the application, it is advisable to establish a solid foundation through mockups. Mockups aims at visualizing the main interface components. They offer a blueprint that guides the developer to turn ideas in the design into working code while contributing to achieve coherence in the UI.

Mockups play a role in the UI design process by:

- **Visual Representation:** They illustrate the real use of the kind of application the developer is creating, and how users will engage it, showcasing key UI components such as navigation bars, input fields, buttons, and content layouts.
- **Feedback and Iteration:** They allow stakeholders to give early feedback on designs and let

designers improve UI designs that are subjected to cycles of usability testing and feedback from users.

- **Visual Representation:** They help to coordinate developers and stakeholders visions of the application, making them consistent before actual coding is conducted.

In this way, we check that all the elements of the UI design provided are compliant with usability standards from the very beginning of the development phase. In this part of the work, the focus will be made on the specific elements of such UI that are intended for the application, with the description of their functions and the ways they support user-oriented interaction. These components fit into an overall strategy of supporting the design with the consideration of goals such as simplicity and consistency in the application. Figures 3.3, 3.4, 3.5, 3.6, 5.8, 3.8 and 3.9 illustrates the mockups that were created for the application.

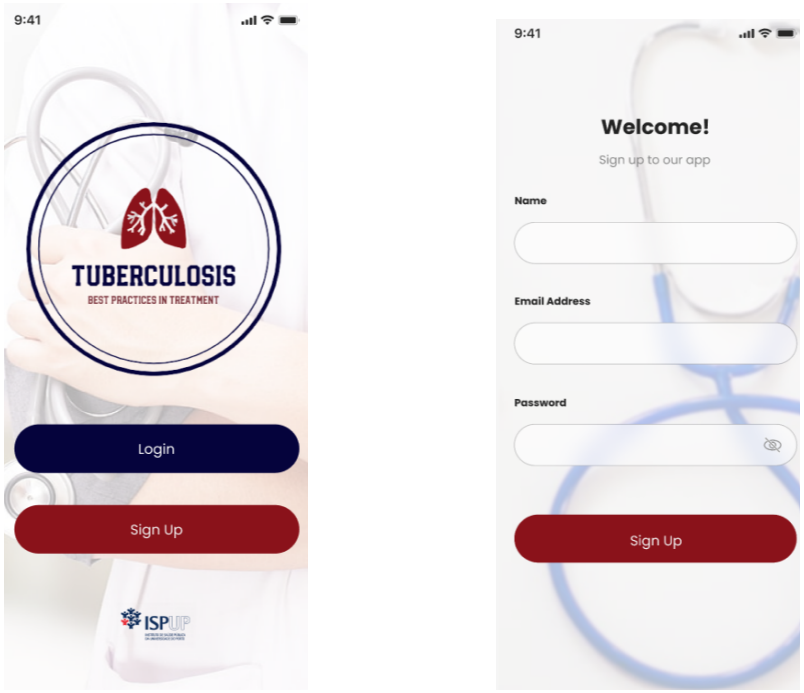
3.4.2.1 Mobile Application

Presentation Screen:

- **App Logo:** The app logo is the visible representation of the application's purpose.
- **Login Button:** By clicking this button the user will be redirected to the login screen, allowing existing users to access their accounts quickly and securely.
- **Sign Up Button:** By clicking this button the user will be redirected to the sign up screen, enabling new users to create an account and start using the app.
- **ISPUP Logo:** The logo of the ISPUP stands for the affiliation of the application to the Institute of Public Health of the University of Porto (ISPUP), which gives it credibility and authority.

Sign Up Screen:

- **Input Fields:**
  - **Name:** This field requires the user to enter their full name.
  - **Email Address:** Users are required to input their email address, which will be used for account verification, communication, and login credentials.
  - **Password:** This field requires the user to create a password for their account. An associated eye icon allows users to toggle the visibility of their password, helping to prevent typos and ensuring secure password creation.
- **Sign Up Button:** This is the call-to-action button that, when clicked, submits the user's information to create their account, completing the registration process and granting access to the application.



(a) Presentation Screen (b) Sign Up Screen

Figure 3.3: Mockups: Presentation And Sign Up - Mobile App

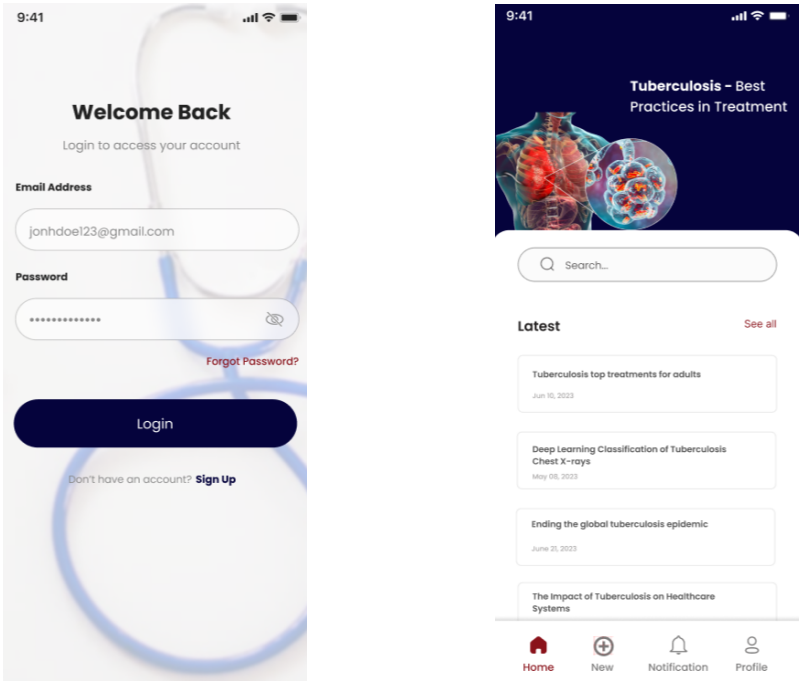
Login Screen:

- **Input Fields:**
  - **Email Address:** Users are required to input their email address.
  - **Password:** This field requires the user to input their password. An associated eye icon allows users to toggle the visibility of their password.
- **Login Button:** This is the call-to-action button that, when clicked, submits the user's information, verifies their account and grants access to the application.

Home Screen:

- **Search Bar:** Allows users to quickly search for specific questions or answers.
- **Latest Articles List:** Displays a list of the latest articles (answered questions) in a scrollable format.
- **See All:** By clicking the "See all" option the user will be redirected to a view with all the articles available in the application.
- **Button Navigation Bar:** It allows users to seamlessly switch between different screens within the application: Home, New (for submitting new questions), Notification and Profile.

New Question Screen:



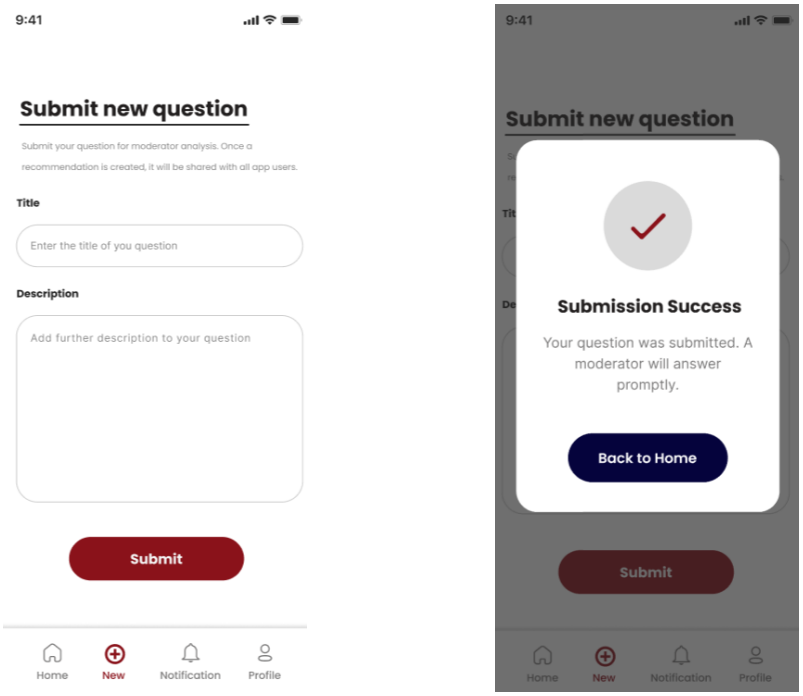
(a) Login Screen (b) Home Screen

Figure 3.4: Mockups: Login and Home - Mobile App

- **Input Fields:**
  - **Title:** Users should enter the main question here. This field captures the core inquiry succinctly and clearly.
  - **Description:** This field allows users to provide detailed information and context about their question. This additional information helps moderators to give a more informed and accurate response.
- **Submit Button:** Clicking this button saves the question to the database and makes it available for moderators to review and respond to on the moderator website.
- **Button Navigation Bar:** It allows users to seamlessly switch between different screens within the application: Home, New (for submitting new questions), Notification and Profile.

Article Detail Screen:

- **Question:** Displays the title of the question for easy identification.
- **User:** Indicates the date the question was answered, which is also when the article was posted in the app.
- **Bookmark Icon:** Allows users to bookmark the article for quick and convenient access in the future.



(a) Submit New Question (b) Submission Success

Figure 3.5: Mockups: New Question - Mobile App

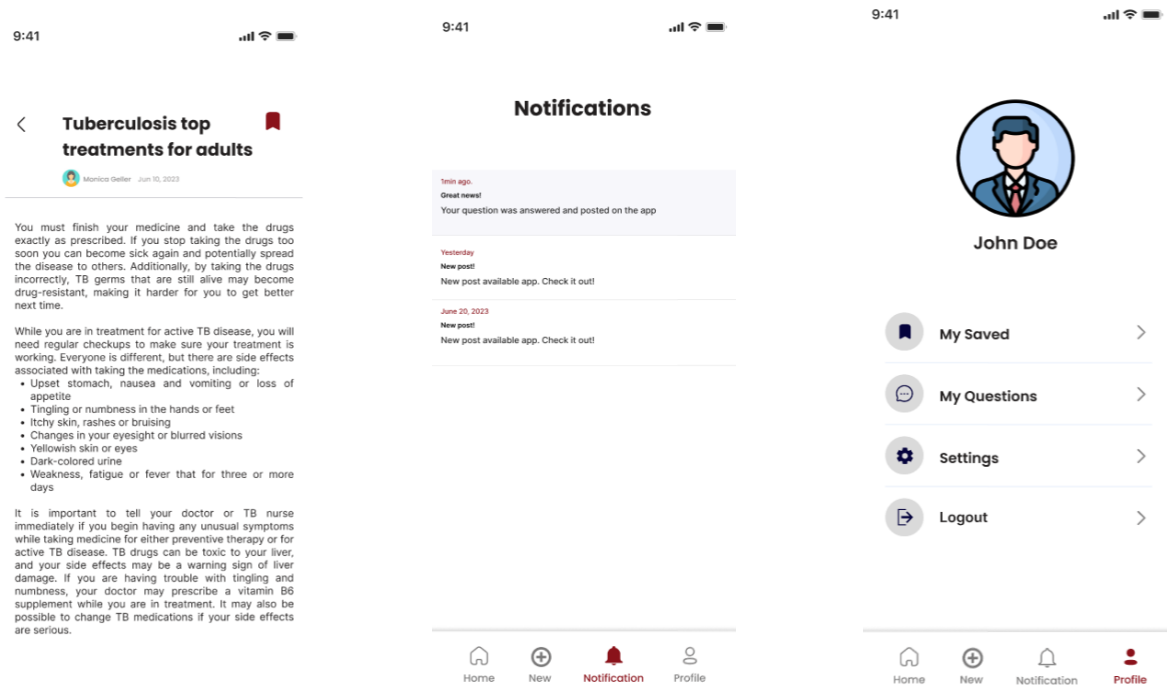
- **Answer:** Presents the detailed answer or recommendation provided by the moderator, offering a comprehensive response to the question.

Notifications Screen:

- **Notification List:** Displays a list of notifications in chronological order. Each notification includes a message, timestamp, and read/unread status.
- **Notification Details:** Tapping on a notification shows more details and marks it as read.
- **Button Navigation Bar:** It allows users to seamlessly switch between different screens within the application: Home, New (for submitting new questions), Notification and Profile.

User Profile Screen:

- **User Information:** Displays the user's profile information, including username and email.
- **My Saved:** By clicking this option the user will see a list of all articles it has bookmarked.
- **My Questions:** By clicking this option the user will see a list of all questions asked by the user and the corresponding details and answers.
- **Settings:** Allows users to manage their preferences, including activating notifications and changing their password.



(a) Article Details (b) Notifications (c) User Profile

Figure 3.6: Mockups: Article, Notifications, and User Profile - Mobile App

- **Logout Button:** Allows users to log out of the application, returning to the login screen.
- **Button Navigation Bar:** It allows users to seamlessly switch between different screens within the application: Home, New (for submitting new questions), Notification and Profile.

3.4.2.2 Moderators Website

Login Page:

- **Input Fields:**
  - **Email Address:**Moderators are required to input their email address.
  - **Password:** This field requires the moderator to input their password.
- **Login Button:** This is the call-to-action button that, when clicked, submits the moderator's information, verifies their account, and grants access to the website in case they have the necessary permissions.
- **App Logo:** The app logo is the visible representation of the application's purpose.
- **ISPUP Logo:** The logo of the ISPUP stands for the affiliation of the application to the Institute of Public Health of the University of Porto (ISPUP), which gives it credibility and authority.

Navigation Menu:

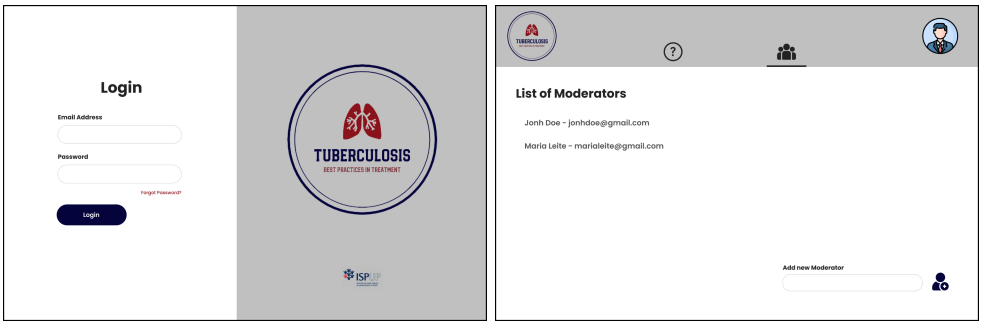
- Provides easy access to different sections of the web app, such as "New Questions" and "List of Moderators".

New Questions Page:

- Lists new questions submitted by users. Moderators can review each question, provide an answer, or reject the question.

Question Detail Page:

- **Moderator List:** Displays a list of all moderators, including their usernames and emails.
- **Add Moderator Button:** Provides a form for adding a new moderator by entering their email address.



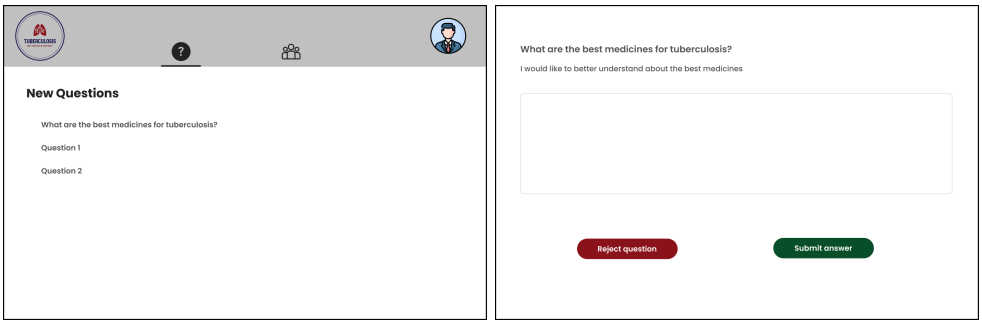
(a) Login Page (b) Moderators Page

Figure 3.7: Mockups: Login and Moderators Page - Website

Moderators List Page:

- **Question Title and Description:** Displays the title and detailed description of the user's question.
- **Answer Form:** Includes a form for moderators to input their answer or recommendation.
- **Submit Answer Button:** Submits the moderator's answer, automatically making the article available in the app for all users to access.
- **Reject Button:** Allows moderators to reject the question if they believe it has already been answered in the app. When a question is rejected, it is deleted from the database.

Moderator Profile Page:



(a) New Questions (b) Question Details

Figure 3.8: Mockups: Questions Page - Website

- **User Information:** Displays the moderator's profile information, including username and email.
- **Logout Button:** Allows moderators to log out of their account, returning to the login page.

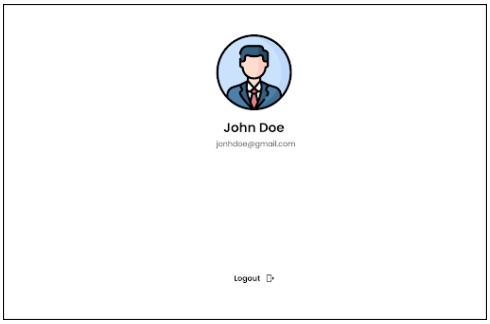


Figure 3.9: Mockups: Moderator Profile - Website

Chapter 4

Implementation

The implementation chapter gives insights into the process that went into transforming the proposed application design into a usable mobile and web app. This chapter focuses on architectural design and presents the features that were incorporated, and the flow used in the user experience. It starts from the discussion of the system architectural framework in which it is described, the architectural pattern and design principles have been used.

Following that, it provides insights into the actual features of the application to consider how each feature was designed and incorporated into the system. Furthermore, it contains the sequence diagrams to define how the user will interact with the application for a smooth experience. This chapter will discuss the practical processes performed during the development phase in more detail to ensure that the challenges as well as their solutions are well understood.

4.1 Client-Server Architecture

This section describes the client-server architecture, which is important in the development of the application. It focuses on coordination of the mobile/web clients developed using Flutter and the Firebase server to support instant data flow, secure sign-in, and retrieval/storing of the data.

4.1.1 Client-Server Interactions

Client Side

- **Technical Details:** Utilizes Flutter SDK for building native interfaces on iOS, Android and Web platforms. Widgets and plugins in Flutter facilitate UI components and app logic.
- **Integration:** Communicates with Firebase services (Firestore, Authentication) via Dart SDK.

Server Side

- **Firebase Authentication:** Manages user authentication and access control for both the mobile and web apps.
- **Firestore:** Serves as the database to store users, questions, answers, notifications, and bookmarks.
- **Integration:** Implements Firestore rules for secure data access and Firestore triggers for real-time updates across clients.

Data Flow:

- Users interact with the mobile app to perform various actions, such as submitting new questions, viewing articles, and bookmarking.
- Moderators use the web app to review new questions and provide answers, as well as manage user roles.
- Data is synchronized between the client apps and Firestore in real-time, ensuring that changes are reflected immediately across all clients.
- Firebase services handle authentication and real-time database interactions

Figure 4.1 is a clear representation of the system architecture.

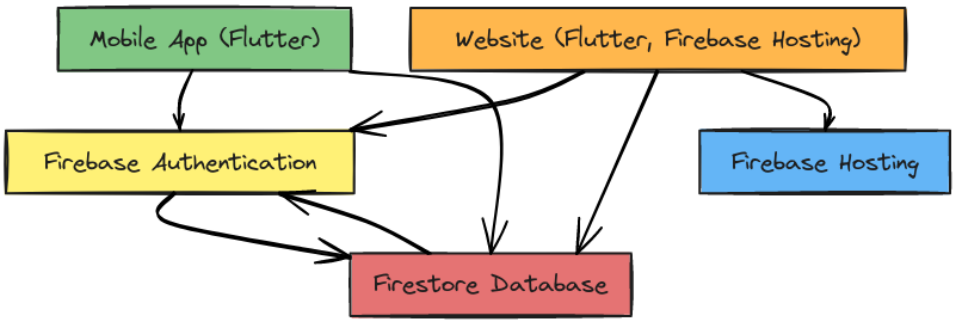


Figure 4.1: System Architecture

4.1.2 Firebase Setup and Configuration

This section describes in detail how Firebase has been set up in connection with Flutter for the development of the application. Configuration means that the client-side, which is mobile and web applications, will interconnect with the Firebase back end services including authentication, databases, and notifications as desired. This setup and configuration was done by following Firebase's official instructions available on their website.

4.1.2.1 Setting Up Firebase Project

The first step in the integration process was to create a Firebase project. Then we access Firebase Console, select the "Add Project" button and follow the instructions given.

Subsequently, Firebase was added to the Flutter application by selecting the Flutter icon in the Firebase Console, registering the app, and downloading the necessary configuration files: `google-services.json` for Android and `GoogleService-Info.plist` for iOS.

4.1.2.2 Flutter Project Configuration

In the Flutter project, dependencies were specified in the `pubspec.yaml` file to include Firebase core services such as `firebase_core`, `firebase_auth` and `cloud_firestore`. These dependencies were installed using the flutter `pub get` command, ensuring the project was equipped with the necessary libraries for Firebase integration.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: latest_version
  firebase_auth: latest_version
  cloud_firestore: latest_version
```

Listing 4.1: Flutter Project Dependencies

Configuring Firebase for Android

For Android, the `google-services.json` file was placed in the android/app directory. The `android/build.gradle` file was updated to include the Google services classpath, and a`android/app/build.gradle` was modified to apply the Google services plugin.

```
dependencies {
  classpath 'com.google.gms:google-services:latest_version'
}
```

Listing 4.2: Android Configuration (1)

```
plugin {
  'com.google.gms.google-services'
}
```

Listing 4.3: Android Configuration (2)

Configuring Firebase for iOS

For iOS, the `GoogleService-Info.plist` file was placed in the ios/Runner directory and

added to the project navigator in Xcode. The dependencies were installed using `pod install` in the ios directory.

Configuring Firebase for Web

For web configuration, the Firebase web app was added in the Firebase Console, and the setup instructions were followed. The `index.html` file was edited to include the Firebase SDK scripts.

```
// Web app's Firebase configuration
const firebaseConfig = {
  apiKey: "**************************",
  authDomain: "******************************",
  projectId: "********************",
  storageBucket: "***************",
  messagingSenderId: "**********",
  appId: "*****************************"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Listing 4.4: Web Configuration

4.1.2.3 Initializing Firebase in Flutter

In the main.dart file, Firebase was initialized to prepare the app for Firebase services. This initialization ensured that Firebase services were ready for use throughout the application.

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  runApp(MyApp());

};
```

Listing 4.5: Initialize Firebase

4.1.2.4 Configuring Firebase Services

Firebase Authentication

The `FirebaseAuthService` class works to incorporate Firebase Authentication with the Flutter application. It uses the `FirebaseAuth` instance for handling the authentication of users and utilizes the `ScaffoldMessenger` for showing error / success messages to the users. Both mobile and web applications use this class.

- **Class Properties**
  - **FirebaseAuth_auth**: An instance of FirebaseAuth used to perform authentication operations.
  - **GlobalKey<ScaffoldMessengerState> scaffoldMessengerKey**: A key to access the ScaffoldMessenger for displaying SnackBars.

- **Methods**
  - **signUpWithEmailAndPassword**:
    - Purpose: Registers a new user with an email and password.
    - Parameters: `String email, String password` – the user's email and password.
    - Returns: `User?` – he authenticated user if successful, otherwise `null`.
    - Error Handling: Catches `FirebaseAuthException` and generic exceptions to provide user feedback via SnackBars.
  - **signInWithEmailAndPassword**:
    - Purpose: Authenticates an existing user with an email and password.
    - Parameters: `String email, String password` – the user's email and password.
    - Returns: `User?` – he authenticated user if successful, otherwise `null`.
    - Error Handling: Catches `FirebaseAuthException` and generic exceptions to provide user feedback via SnackBars.

These figures below represent the views seen within the Firebase console under the Authentication tab. Figure 4.2 displays the list of authenticated users within the project, while Figure 4.3 shows the selected login method for the application.

Cloud Firestore

The Firestore package in Flutter, `cloud_firestore`, facilitated the interaction with the Firestore database, enabling CRUD operations.

Firestore uses a data model where data is captured in collections and documents hence allowing for easy management of the data.

Collection: A collection is a container for documents. Collections do not store data directly but organize documents. Each collection can contain multiple documents.

Figure 4.2: Firebase Authentication - Users

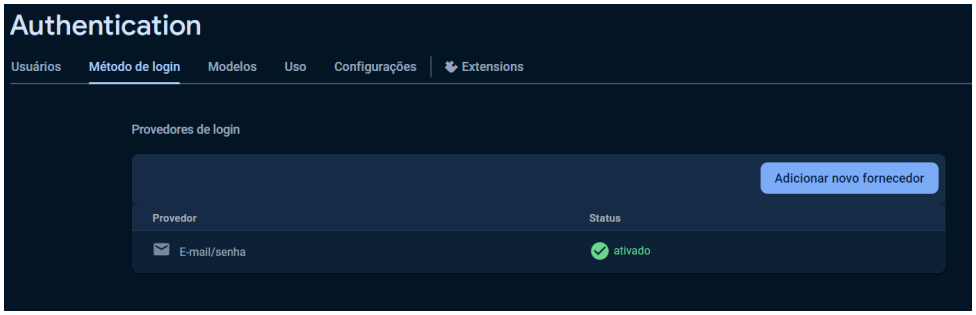


Figure 4.3: Firebase Authentication - Login Mode

Document: A document is a unit of storage within a collection, identified by a unique ID. Documents can contain sub-collections and fields. Each document stores key-value pairs for the attributes of the data model.

The data model for the application includes several key collections, each representing different aspects of the app's data:

Users Collection

- **Purpose:** Stores user details.
- **Fields:**
  - `username`: The username of the user.
  - `email`: The email address of the user.

Moderators Collection

- **Purpose:** Stores details of moderators.
- **Fields:**
  - `username`: The username of the moderator.
  - `email`: The email address of the moderator.

Questions Collection

- **Purpose:** Stores questions submitted by users.
- **Fields:**
  - `Id`: The ID of the question.
  - `title`: The title of the question.
  - `description`: Detailed description of the question.
  - `createdOn`: Timestamp of when the question was created.
  - `isAnswered`: Boolean indicating whether the question has been answered.
  - `answer`: The answer provided by a moderator (if applicable).
  - `answeredOn`: Timestamp of when the question was answered (if applicable).
  - `userEmail`: The email address of the user who submitted the question.

Notifications Collection

- **Purpose:** Represents notifications sent to users about updates and activities.
- **Fields:**
  - `Id`: The ID of the notification.
  - `message`: The notification message.
  - `timestamp`: Timestamp of when the notification was sent.
  - `read`: Boolean indicating whether the notification has been read.
  - `questionId`: (Optional) The ID of the associated question.
  - `userId`: The ID of the user receiving the notification.

Bookmarks Collection

- **Purpose:** Manages user bookmarks for quick access to important questions.
- **Fields:**
  - `userId`::The ID of the user who bookmarked the question.
  - `questionId`: The ID of the bookmarked question.

The Figure 4.4 below showcases the collections within our Firestore database, specifically highlighting the Questions Collection along with its associated fields and information.

In the next section 4.2 about Model–View–Controller (MVC) architecture, we will discuss how these models are implemented and used in the application. The MVC architecture plays a crucial role in the implementation of how data will be stored, processed, and used to create a solid and scalable application.

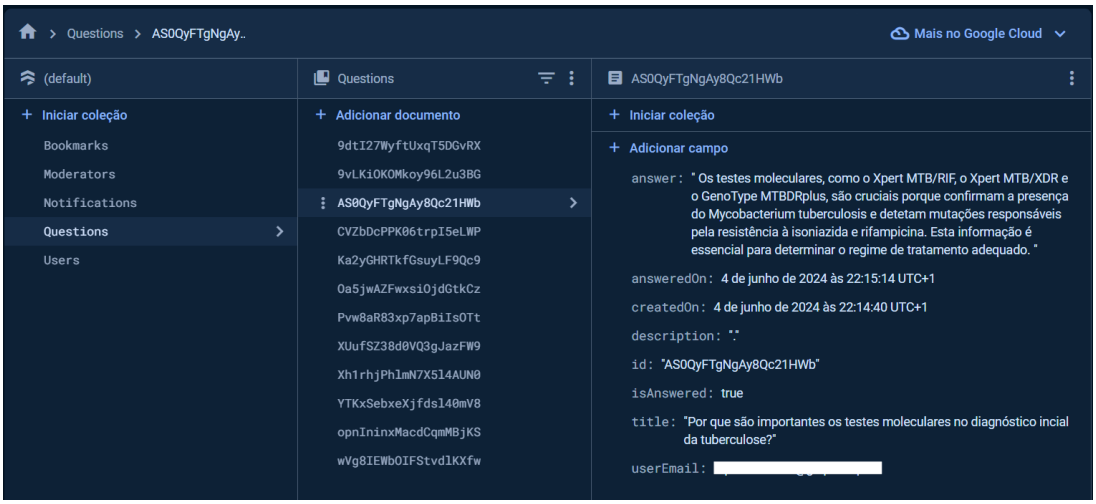


Figure 4.4: Firestore Collections

Firestore Security Rules

The Firestore security rules are designed to ensure that only authenticated users have appropriate access to the data stored in the Firestore database, ensuring data security and privacy.

1. **Users Collection**

```
match /Users/{document=**} {
   allow read, write: if request.auth != null;
}
```

- Explanation: This rule allows only authenticated users to read and write to the "Users" collection.

2. **Questions Collection**

```
match /Questions/{document=**} {
   allow read, write: if request.auth != null;
}
```

- Explanation: This rule permits only authenticated users to write to the "Questions" collection. This prevents unauthorized users from adding new questions, maintaining the integrity of the data.

3. **Moderators Collection**

```
match /Moderators/{document=**} {
  allow read, write: if request.auth != null;
}
```

- Explanation: Authenticated users are allowed to read and write to the "Moderators" collection.

4. **Bookmarks Collection**

```
// General Access Rule
match /Bookmarks/{document=**} {
  allow read, write: if request.auth != null;
}

// User-Specific Rule
match /Bookmarks/{bookmarkId} {
  allow read, write: if request.auth != null && request.auth.uid ==
      resource.data.userId;
}
```

- Explanation: The general rule allows only authenticated users to read and write to the "Bookmarks" collection. The user-specific rule ensures that users can only read and write their own bookmarks, enhancing data privacy and preventing unauthorized access to other users' bookmarks.

5. **Notifications Collection**

```
// General Access Rule
match /Notifications/{document=**} {
  allow read, write: if request.auth != null;
}

// User-Specific Rule
match /Notifications/{notificationId} {
  allow read, write: if request.auth != null && request.auth.uid ==
      resource.data.userId;
}
```

- Explanation: The general rule allows only authenticated users to read and write notifications. The user-specific rule ensures that users can only read and write notifications intended for them, maintaining user-specific privacy.

4.2 System Architecture: Model-View-Controller (MVC)

The Model-View-Controller (MVC) architecture is a crucial component of the application design in this project. It organizes both the mobile app and the web-based moderator interface to improve modularity, adaptability, and maintainability [15, 16].

4.2.1 Components of MVC Architecture

The Model-View-Controller (MVC) architecture separates an application into three main components: Model, View, and Controller. Figure 4.5 is a visual representation of this interaction. Here's how these components typically interact:

1. **Model:**Manages the data and business logic.
2. **View:** Used for all the UI logic of the application. Displays the data and sends user actions to the Controller.
3. **Controller:** Processes user input, interacts with the Model, and updates the View.

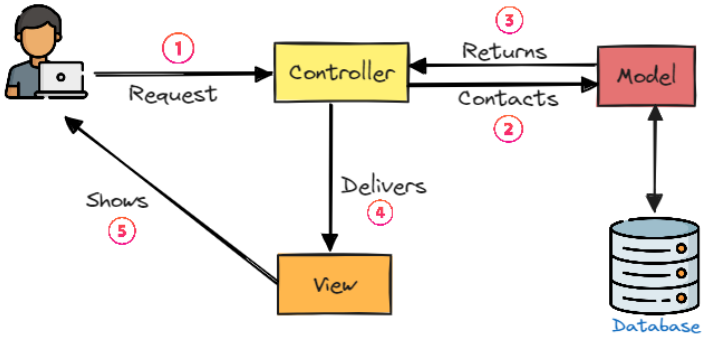


Figure 4.5: MVC Architecture

Model: The Model component deals with organizing the data or the details of business and computation operations. It binds into application all data structures that are used and encodes the logic rules that define the operations of a data. It is connected to the database, so it handles data addition and retrieval. Furthermore, it responds to controller queries, since the controller does not communicate with the database on its own.

- **Repository Pattern:** Each model, such as `UserModel`, `ModeratorModel`, `QuestionModel`, `NotificationModel`, and `BookmarkModel`, is accompanied by a corresponding repository, like `UserRepository` and `QuestionRepository`.
- **Responsibilities:** Repositories encapsulate data access logic, including CRUD (Create, Read, Update, Delete) operations specific to their respective models.

In Listing 4.6, we can see the UserModel class that represents each user in the application. The UserRepository in 4.7 has methods that interact with the Firestore database, and that will be used by the controller as well.

View: The View component is responsible for presenting data to users and capturing user input. It deals with the visual representation of the application's data and the user interface

components. Views are built using data acquired by the model component, but these data are not obtained directly but through the controller.

- **Flutter Widgets:** The view uses Flutter's rich set of widgets to construct UI components for the mobile app.
- **Responsibilities:** The view displays data retrieved from controllers and captures user interactions to initiate actions.

In 4.8 we can see a code snippet for Login View.

```
class UserModel {
  String? id;
  final String username;
  final String email;

  UserModel({
    this.id,
    required this.username,
    required this.email,
  });

  Map<String, dynamic> toJson() {
    return {
      "UserName": username,
      "Email": email,
    };
  }
}
```

Listing 4.6: User Model

```
class UserRepository {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<void> saveUser(UserModel user) async {
    try {
      await _firestore.collection('Users').doc(user.id).set(user.toJson());
    } catch (e) {
      ...
    }
  }
}
```

Listing 4.7: User Repository

Controller: The Controller component implements the application logic and mediates between the Model and View. It manages user input and updates the model state based on user

actions. The controller only needs to instruct the model on what to do; it does not have to worry about managing data logic. Data is received from the model, processed, and then provided to the view.

- **Centralized Logic:** Manages user input and orchestrates data flow between the Model and View, ensuring data integrity and adherence to business rules.

An example of a controller is defined in 4.9 that shows a snipped code for login controller.

```
class LoginPage extends StatefulWidget {
  const LoginPage({super.key});
  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  ...

  @override
  Widget build(BuildContext context) {
    // Instantiate the LoginController
    final LoginController loginController = LoginController(
      authService:
          FirebaseAuthService(scaffoldMessengerKey: _scaffoldMessengerKey),
      userRepository: UserRepository(),
      context: context,
    );

    return Scaffold(
      body: SingleChildScrollView(
        child: SizedBox(
          width: double.infinity,
          child: Container(
            width: double.infinity,
            height: screenHeight,
            decoration: BoxDecoration(
              image: DecorationImage(
                fit: BoxFit.cover,
                alignment: Alignment.topCenter,
                image: const AssetImage(
                  'assets/page-1/images/image-8-bg.png',
                ),
        ...
    );
   }
}
```

Listing 4.8: Login Page View

Benefits of using MVC:

1. **Separation of Concerns:** The separation of concerns allows for cleaner code organization. Changes to the Model, View, or Controller can be made independently with minimal impact on other parts of the application. Allows for easier maintenance and testing of data operations independent of user interface components.

2. **Encapsulation of Logic:** Promotes code reusability and simplifies maintenance. Components within each layer (Model, View, and Controller) can be potentially reused in other applications.

3. **Testability:** Each layer can be tested independently, making unit testing and integration testing more manageable.

4. **Clear Separation of UI and Logic:** This fosters a cleaner separation between the application's data logic and its presentation.

5. **Decoupling from Business Logic:** Enhances UI responsiveness and flexibility in design updates.

```
class LoginController {
  final FirebaseAuthService _authService;
  final UserRepository _userRepository;
  final BuildContext context;

  LoginController({
    required FirebaseAuthService authService,
    required UserRepository userRepository,
    required this.context,
  })  : _authService = authService,
        _userRepository = userRepository;

  Future<void> signIn(String email, String password) async {
    try {
      final user =
          await _authService.signInWithEmailAndPassword(email, password);
      if (user != null) {
        final userModel = await _userRepository.getUserById(user.uid);
        if (userModel != null) {
          ...
        }
        Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => const HomePage()),
        );
      } else {
        ...
    } catch (e) {
       ...
    }
  }
```

Listing 4.9: Login Controller

4.2.2 Class Diagrams

The use of class diagrams offers a clear picture of the different components of the program by illustrating their interrelation. These diagrams reflect the Model-View-Controller (MVC) design, which illustrates how data (Model), the interface (View), and the control mechanism (Controller) are organized and function in synchrony. The class diagram 4.6 refers to the mobile application and 4.7 refers to the website.

4.2.2.1 Class Diagram - Mobile Application

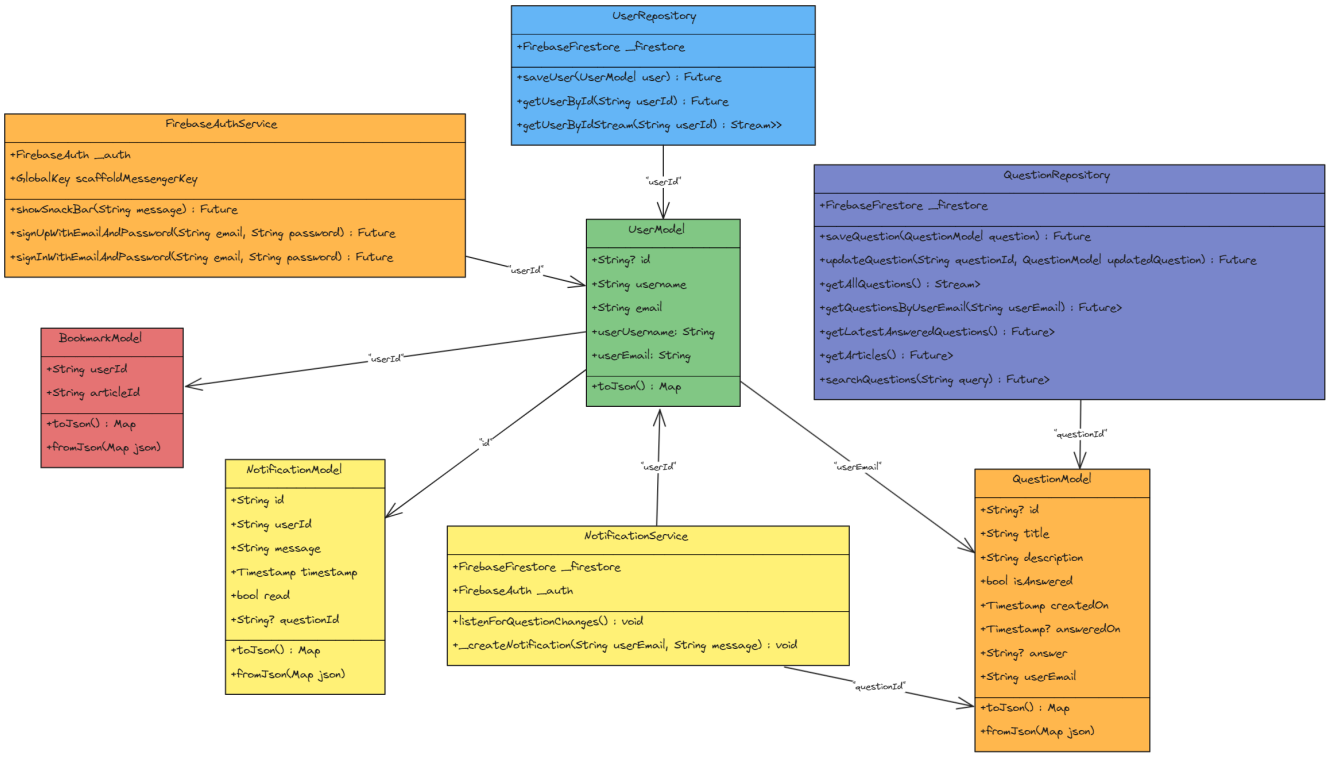


Figure 4.6: Class Diagram - Mobile App

1. **Model:** The Model represents the data structure of the application.
   - **UserModel:** The UserModel class represents a user in the application.
     - Attributes: `id`, `username`, `email`.
     - Methods: *toJson(), getters for username and email.*
   - **QuestionModel:** The QuestionModel class represents a question in the application.
     - Attributes: `id`, `title`, `description`, `isAnswered`, `createdOn`, `answeredOn`, `answer`, `userEmail`.
     - Methods: *toJson(), fromJson(Map json).*
   - **NotificationModel:** The NotificationModel class represents notifications sent to users.

  - Attributes: `id, userId, message, timestamp, read, questionId.`
  - Methods: *toJson(), fromJson(Map json)*.
- **BookmarkModel:** The BookmarkModel class represents user bookmarks.
  - Attributes: `userId, articleId.`
  - Methods: *toJson(), fromJson(Map json)*.

2. **Repository:** Repositories act as intermediaries between the Model and Firestore database, handling data operations. They abstract the complexities of data retrieval and storage, providing a clean API for the Controllers to interact with.
   - **UserRepository:** Handles data operations related to users.
     - Attributes: `_firestore`
     - Methods: *saveUser(UserModel user), getUserById(String userId), getUserByIdStream(String userId)*.
   - **QuestionRepository:** Manages data operations related to questions.
     - Attributes: `_firestore`
     - Methods: *saveQuestion(QuestionModel question), updateQuestion(String questionId, QuestionModel updatedQuestion), getAllQuestions(), getQuestionsByUserEmail(String userEmail), getLatestAnsweredQuestions(), getArticles(), searchQuestions(String query)*.
3. **Services:** We have specific services that handle Notifications and Authentication operations.
   - **NotificationService:** Manages the creation and delivery of notifications.
     - Attributes: `_firestore, _auth.`
     - Methods: *listenForQuestionChanges(), _createNotification(String userEmail, String message)*.
   - **FirebaseAuthService:** Handles user authentication.
     - Attributes: `_firestore, _auth, scaffoldMessengerKey.`
     - Methods: *showSnackBar(String message), signUpWithEmailAndPassword(String email, String password), signInWithEmailAndPassword(String email, String password)*.
4. **Controllers (Not Illustrated in Diagram):** Controller uses the Repository API to perform operations and update the view. Due to limited space, the class diagram does not include controllers.
   - **LoginController:** Handles login logic.
     - Attributes: `_authService, _userRepository, context.`
     - Methods: *signIn(String email, String password)*.
   - **SignUpController:** Handles sign up logic.

- Attributes: `_authService, _userRepository, context.`
- Methods: *signUp(String email, String password, String username).*

- **QuestionController:** Handles all related questions operations.
  - Attributes: `questionRepository, _auth.`
  - Methods: getCurrentUserEmail(), loadQuestionsByUserEmail(String userEmail), showSubmissionDialog(BuildContext context), submitQuestion(BuildContext context, String title, String description),showErrorMessage(BuildContext context, String message),loadArticles(),searchQuestions(String query).

4.2.2.2 Class Diagram - Moderators Website

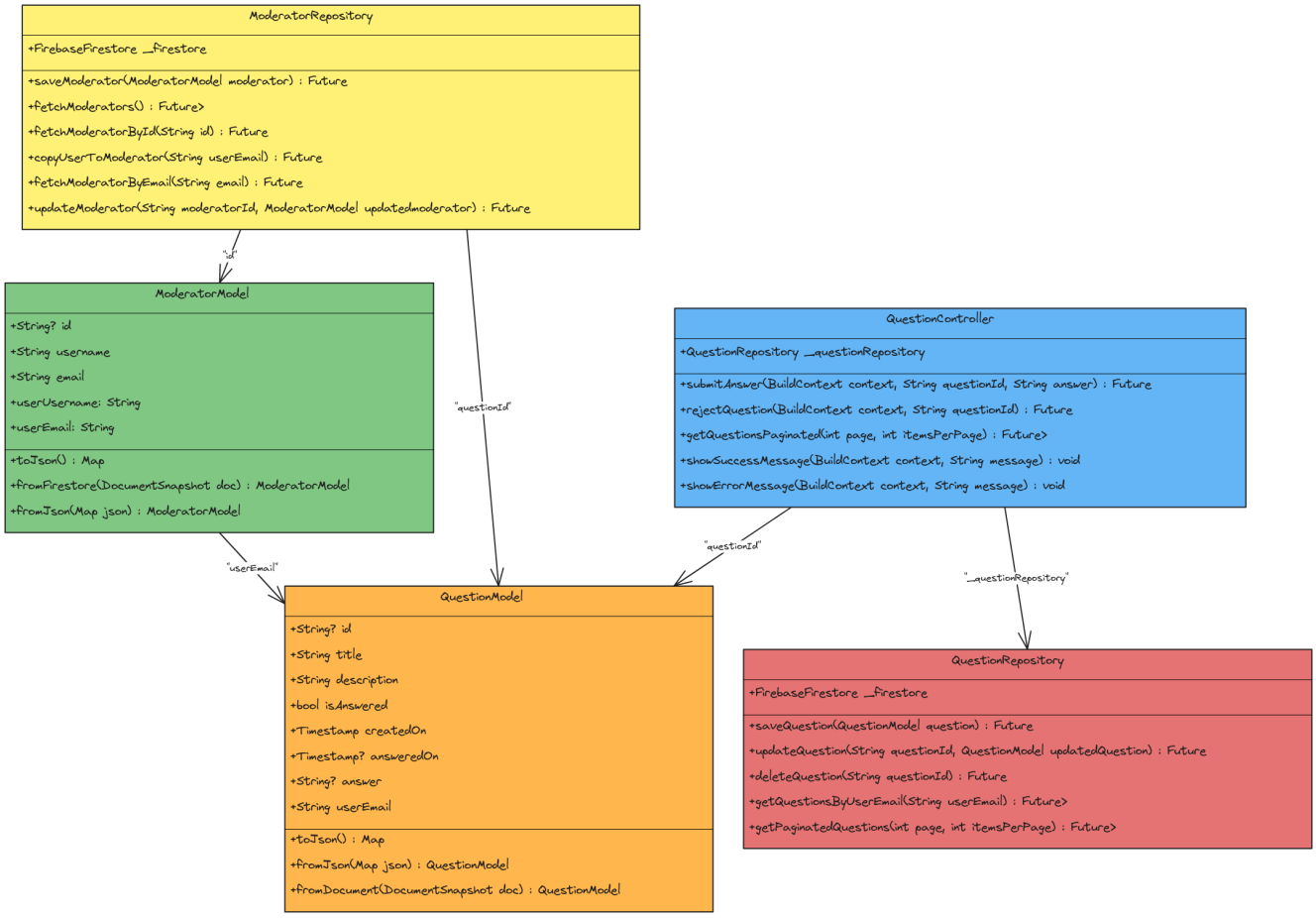


Figure 4.7: Class Diagram - Moderators Website

1. **Model:** The Model component represents the data structure of the moderator website.

   - **ModeratorModel:** The ModeratorModel class represents a moderator in the website.
     - Attributes: `id, username, email.`
     - Methods: *toJson(), fromFirestore(DocumentSnapshot doc), fromJson(Map json)*

- **QuestionModel:** The QuestionModel class represents a question in the website.
  - Attributes: `id, title, description, isAnswered, createdOn, answeredOn, answer, userEmail.`
  - Methods: *toJson(), fromJson(Map json), fromDocument(DocumentSnapshot doc)*.

2. **Repository:** Repositories act as intermediaries between the Model and Firestore database, handling data operations. They abstract the complexities of data retrieval and storage, providing a clean API for the Controllers to interact with.
   - **ModeratorRepository:** Handles data operations related to moderators.
     - Attributes: `_firestore.`
     - Methods: *saveModerator(ModeratorModel moderator), fetchModerators(), fetchModeratorById(String id), copyUserToModerator(String userEmail), fetchModeratorByEmail(String email), updateModerator(String moderatorId, ModeratorModel updatedModerator)*.
   - **QuestionRepository:** Manages data operations related to questions.
     - Attributes: `_firestore.`
     - Methods: *saveQuestion(QuestionModel question), updateQuestion(String questionId, QuestionModel updatedQuestion), deleteQuestion(String questionId), getQuestionsByUserEmail(String userEmail), getPaginatedQuestions(int page, int itemsPerPage)*.
3. **Services:** We have specific services that handle Notifications and Authentication operations.
   - **FirebaseAuthService:** Handles user authentication.
     - Attributes: `_firestore, _auth, scaffoldMessengerKey.`
     - Methods: *showSnackBar(String message), signInWithEmailAndPassword(String email, String password), signOut()*.
4. **Controller (Not Illustrated in Diagram):** The Controller component handles the user input, interacts with the Model, and updates the View accordingly. Due to limited space, the class diagram contains only the QuestionController for illustration.
   - **QuestionController:** Manages operations related to questions, such as submitting answers and rejecting questions.
     - Attributes: `_questionRepository.`
     - Methods: *saveModerator(ModeratorModel moderator), fetchModerators(), fetchModeratorById(String id), copyUserToModerator(String userEmail), fetchModeratorByEmail(String email), updateModerator(String moderatorId, ModeratorModel updatedModerator)*.
   - **LoginController:** Handles login logic.
     - Attributes: `_authService, _moderatorRepository.`
     - Methods: *login(BuildContext context, String email, String password)*.

- **ModeratorController:** Handles login logic.
  - Attributes: `_moderatorRepository`.
  - Methods: *saveModerator(ModeratorModel moderator), fetchModerators(), fetchModeratorById(String id), copyUserToModerator(String userEmail), fetchModeratorByEmail(String email), updateModerator(String moderatorId, ModeratorModel updatedModerator))*.

4.2.2.3 Class Diagram - Complete

The combined class diagram 4.8 merges the mobile app and moderator website, providing a holistic view of the system's architecture.

In conclusion, the class diagram coupled with the MVC architecture offers a sound foundation from which effective and efficient large applications and systems can be developed. The class diagram provides a clear picture about the application modules and how they are interrelated, and the MVC architectural approach promotes a structured way of addressing the issues of organizing the application's logic, its interface and data processing. The result is a neat and clean internal structure of the application which is easily manageable, modifiable and can be expanded to meet ever-increasing demands.

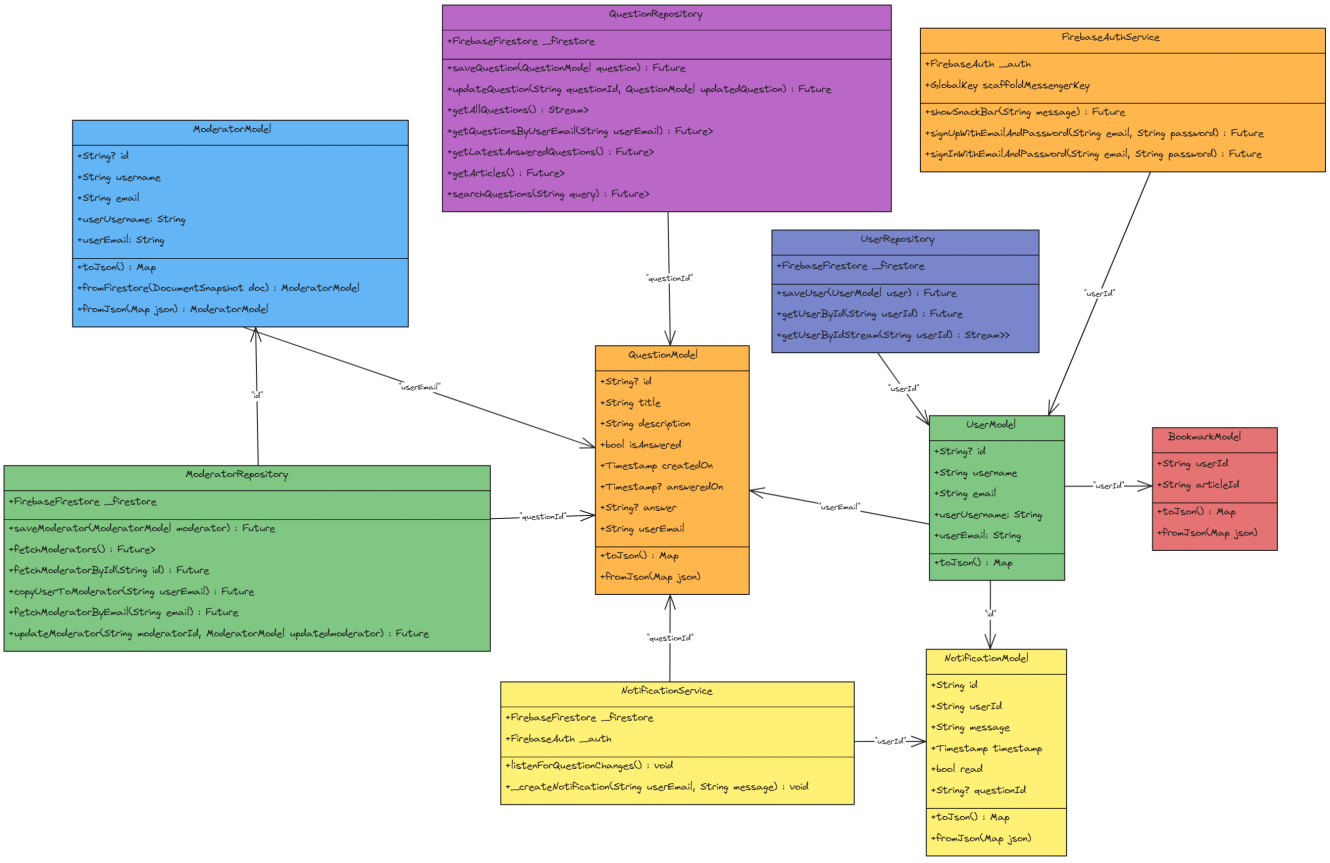


Figure 4.8: Class Diagram - Complete

4.3 User Interaction Flow

The User Interaction Flow section proceeds with focusing on the working of the interaction and, therefore, demonstrates fundamental user operation sequences applying Sequence Diagrams. All the diagrams depict clean user flows beginning with the registration and log-in activities and down to more specific and appealing features such as viewing profiles, sending queries, searching for articles, and receiving notifications. These graphic expressions shed light on the interaction between users and moderators whose duties are portrayed here as being integral to the execution of this app.

4.3.1 User Authentication and Profile Viewing

The flow of actions consisted in user registration, login, and accessing profile information is depicted via the sequence diagram 4.9.

User Registration

1. User selects "Sign Up" option in Mobile App.
2. User enters details (username, email, password).
3. Mobile App sends registration details to Firebase Authentication.
4. Firebase Authentication validates details and creates a new user account.
5. Firebase Authentication sends a confirmation email to the user.

User Login

1. User selects "Login" option in Mobile App.
2. User enters email and password.
3. Mobile App sends credentials to Firebase Authentication.
4. Firebase Authentication validates credentials.
5. Firebase Authentication sends an authentication token to the Mobile App.
6. Mobile App grants access to the user's account.

Viewing Profile

1. User (Mobile App) or Moderator (Website) requests to view profile.
2. Mobile App/Website requests user profile data from Firestore Database.
3. Firestore Database retrieves user profile data and sends it to Mobile App/Website.
4. Mobile App/Website displays user profile.

4.3.2 Question and Notification

The sequence diagram 4.10 depicts the sequential steps involved in submitting, answering/rejecting questions, and receiving notifications.

Submitting a Question

1. User navigates to "New" in Mobile App.
2. User enters question details (title, description).
3. Mobile App sends question details to Firestore Database.
4. Firestore Database validates and records the question.

Answering a Question

1. Moderator navigates to "New Questions" section on Website.
2. Moderator selects a question to review.
3. Moderator enters answer and submits it.
4. Website sends answer details to Firestore Database.
5. Firestore Database records the answer and notifies the user who submitted the question.
6. The answered question is available for all users to view in the mobile application.

Rejecting a Question

1. Moderator navigates to "New Questions" section on Website.
2. Moderator selects a question to review.
3. Moderator rejects the question.
4. The rejected question is deleted from Firestore Database.
5. Firestore Database notifies the user who submitted the question.

Receiving Notifications

1. System generates a notification when a question is answered/rejected.
2. Firebase sends the notification to the user.
3. User views the notification on their device.

4.3.3 Article Viewing, Searching, and Bookmarking

The sequence diagram 4.11 illustrates how users navigate through articles, search for specific information, and save articles for future reference.

Searching Questions and Recommendations

1. User navigates to "Search" tab in Mobile App.
2. User enters search keywords.
3. Mobile App sends search request to Firestore Database.
4. Firestore Database retrieves matching results.
5. Firestore Database sends search results to Mobile App.
6. Mobile App displays search results to the user.

Viewing an Article

1. User navigates to "Home" section in Mobile App.
2. User selects an article to view.
3. Firestore Database fetches article details and sends to Mobile App
4. Mobile App displays article.

Bookmarking an Article

1. User selects Bookmark icon in selected article.
2. Mobile App sends this information to Firestore Database.
3. Firestore database stores article in Bookmarks collection with this associated user.
4. Bookmarked articles are available on "My Saved" section on user's profile.

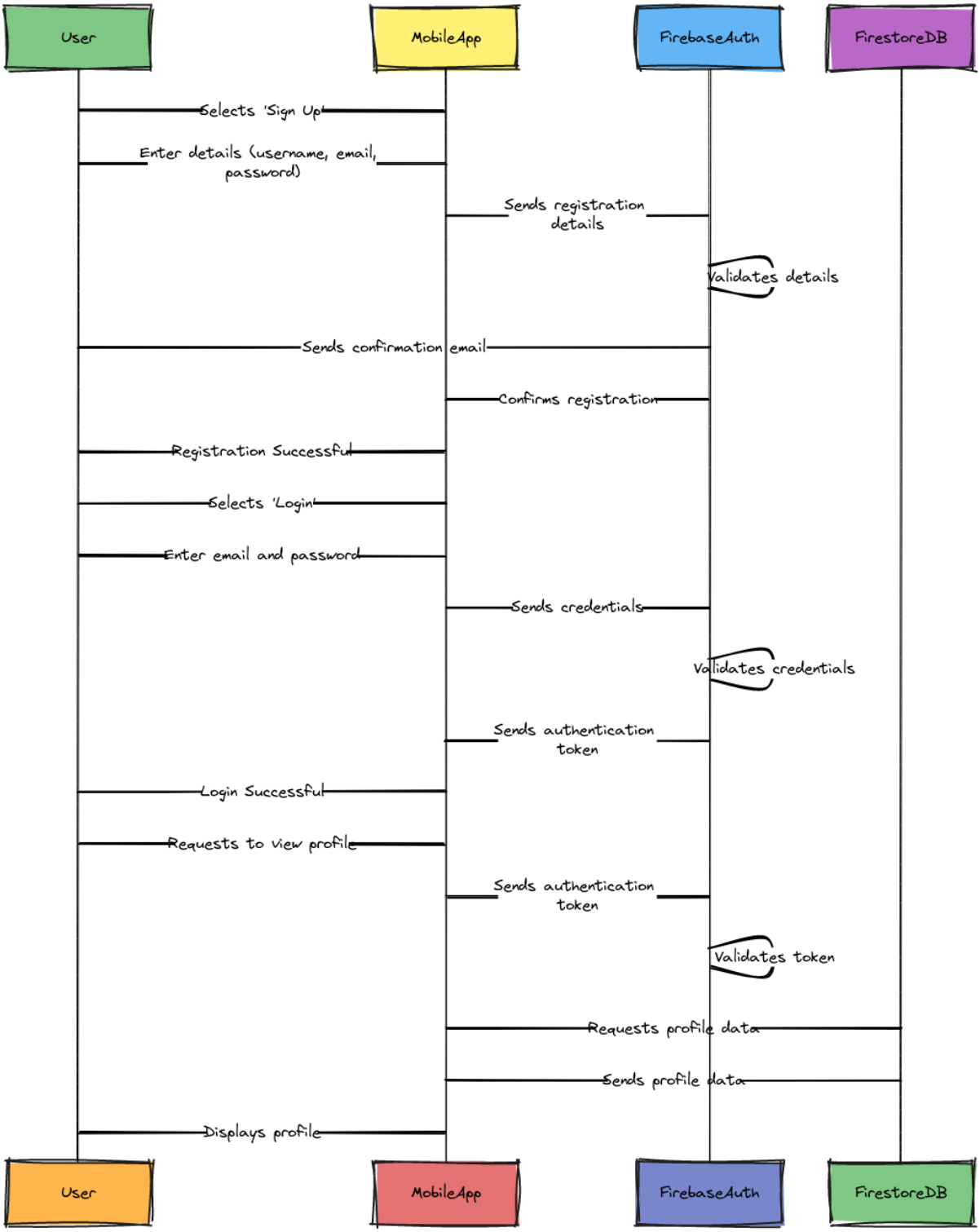


Figure 4.9: Sequence Diagram: User Authentication and Profile Viewing

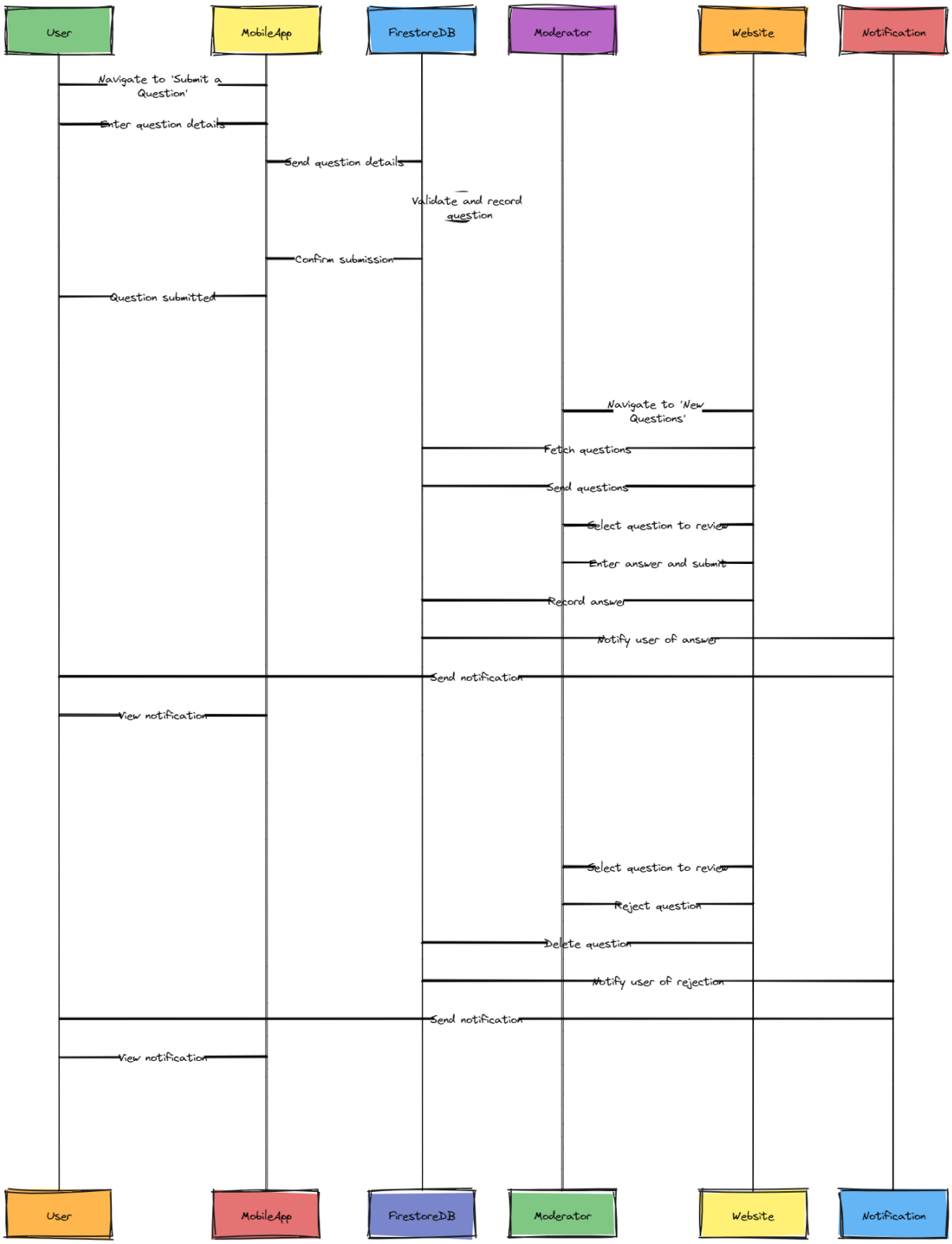


Figure 4.10: Sequence Diagram: Question and Notification

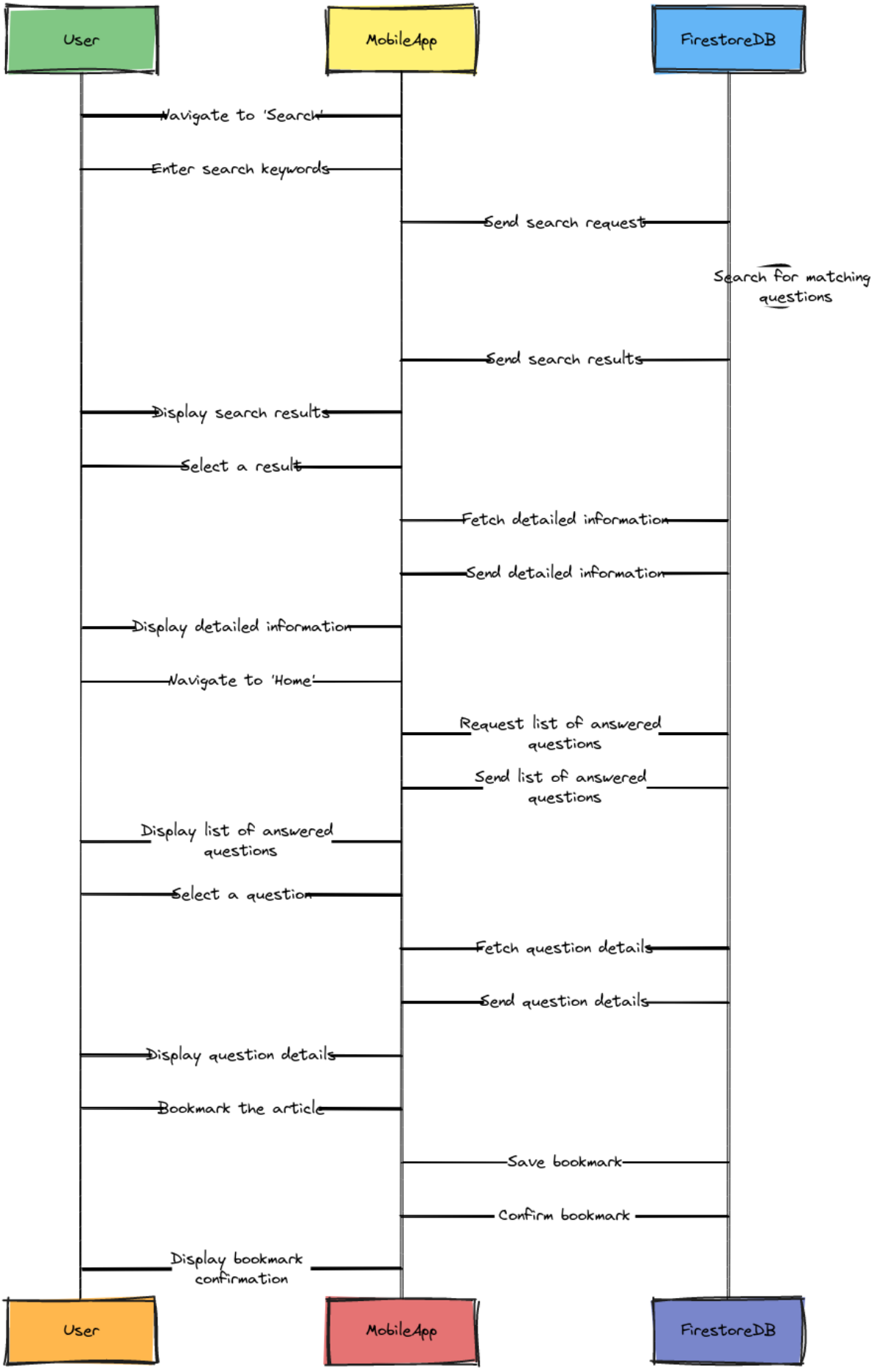


Figure 4.11: Sequence Diagram: Article Viewing, Searching, and Bookmarking

Chapter 5

Evaluation and Discussion

This chapter explains the application evaluation, manual testing carried out, and a demo with the project advisor. The review clarified how the application operates, its usability, and its performance.

5.1 Evaluation

The methods of evaluating the application were manual testing and a demo session with the project advisor. Again, manual testing was used to systematically verify the critical functions and features of the application. This demo aimed to establish the fundamental capabilities of the application and get feedback from the advisor.

5.1.1 Manual Testing Results

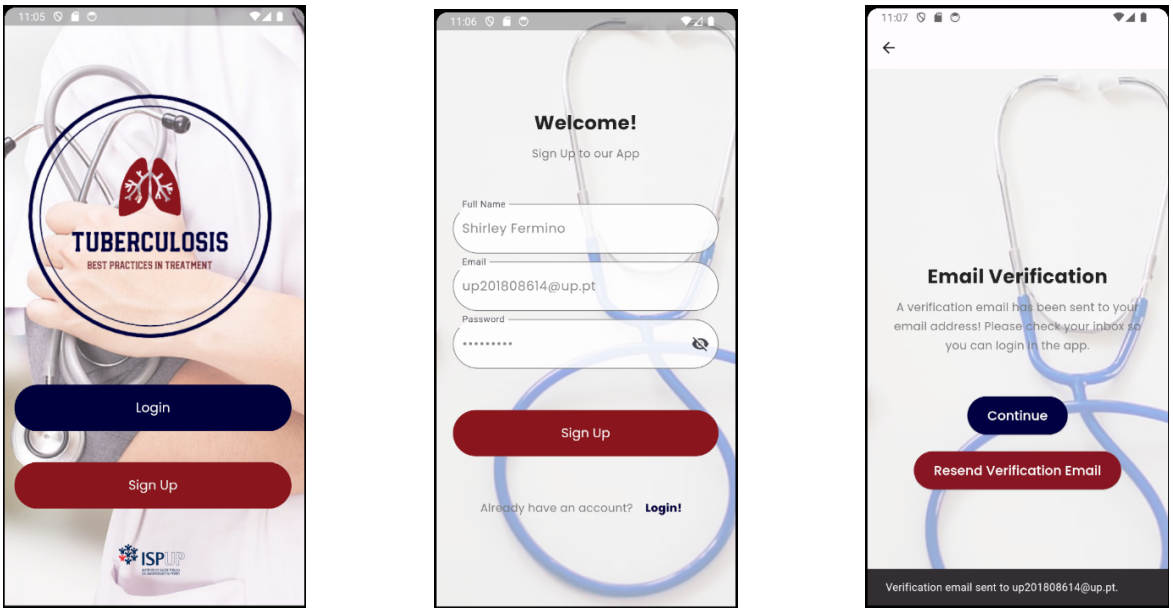
User Authentication: Manual testing has confirmed that functionalities for registration, login, and log out have been successfully implemented. Users can sign up, log in, and log out of their created accounts.

Question Management: Questions related functionalities have been tested successfully. All functionalities are working perfectly, such as submitting new questions, answering/rejecting questions, viewing a list of answered questions(articles), and viewing article details.

Notification System: Testing was done by posting questions, and answering them and checking if the appropriate user received notifications.

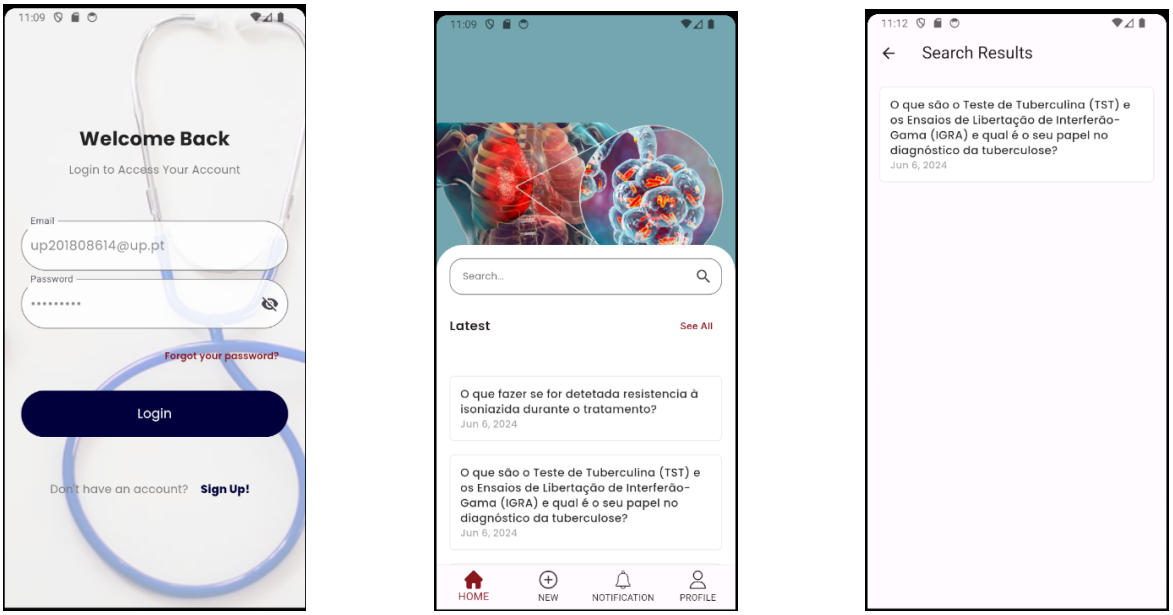
Bookmarking: Through a process of bookmarking and not bookmarking items available to be marked or unmarked, it passed flawlessly. All bookmarks are reflected in the profile under “My Saved” section.

Bellow we can see screenshots of real test performed. First as app user, then as a moderator in the website.



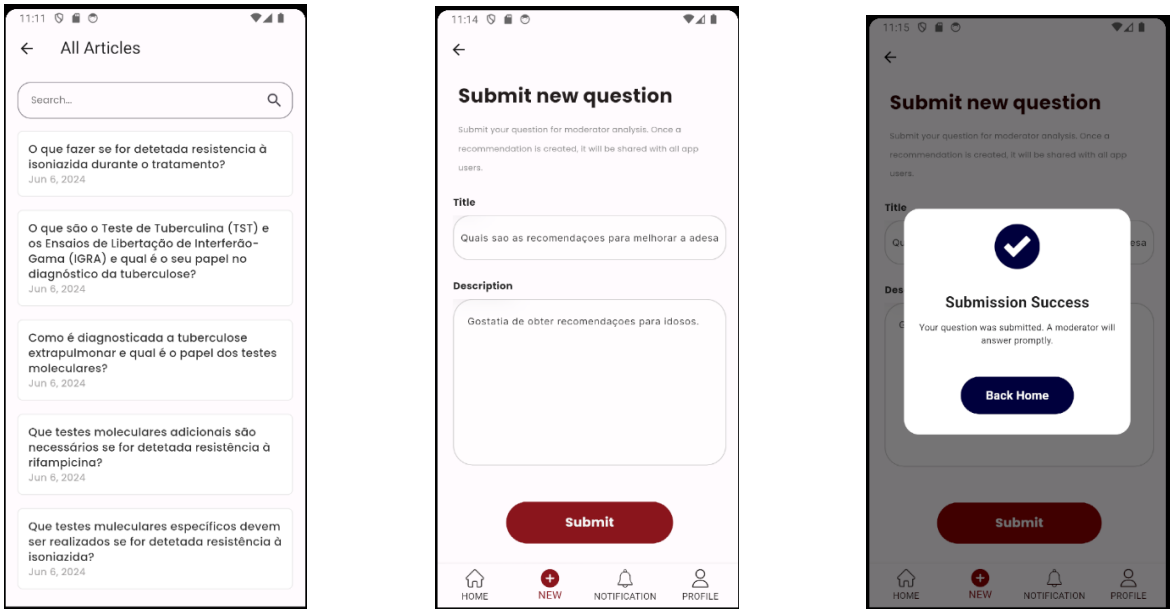
(a) Presentation (b) Sign Up (c) Email Verification

Figure 5.1: Presentation and Sign Up - Mobile App



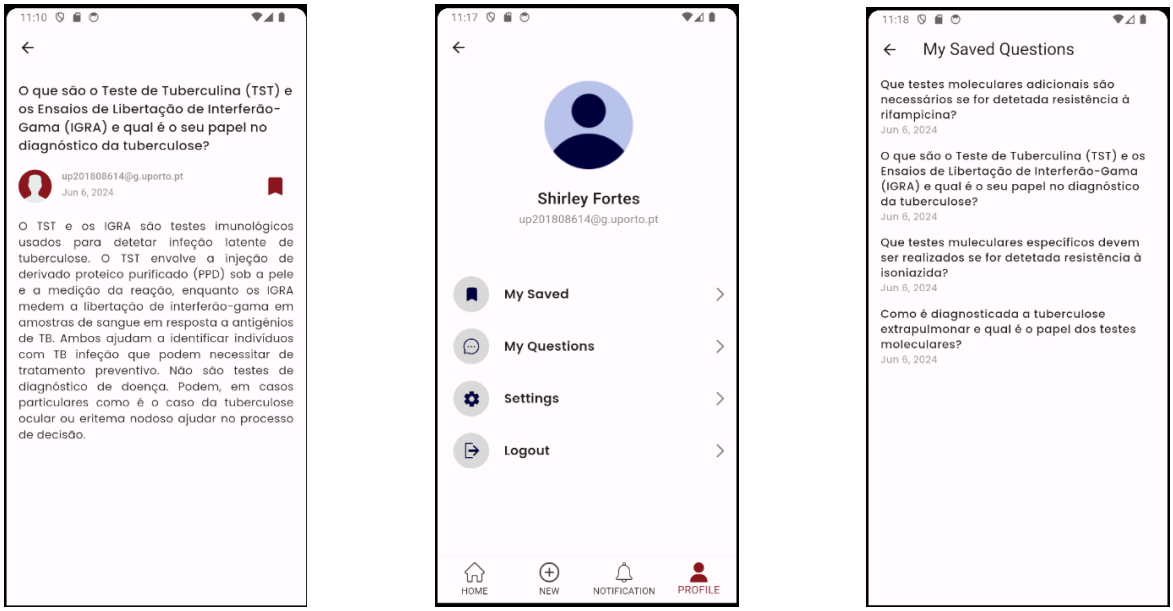
(a) Login (b) Home (c) Search Results

Figure 5.2: Login, Home and Seach - Mobile App



(a) All Articles (b) New Question (c) Submission Success

Figure 5.3: All Articles List and New Question Submission - Mobile App

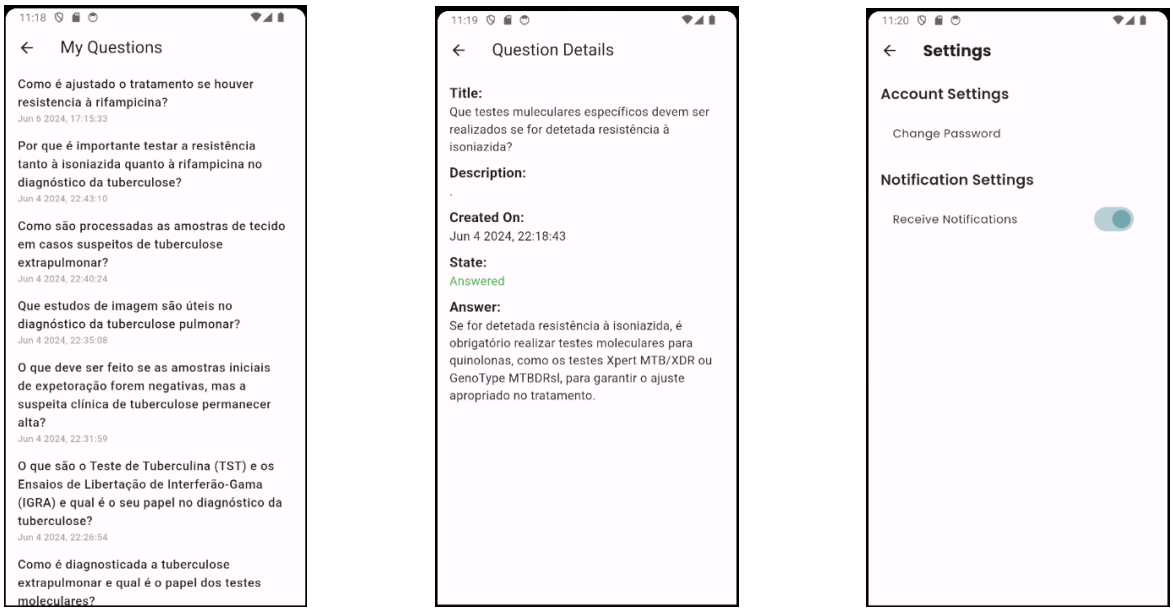


(a) Article Details (b) Profile (c) Bookmarked Articles List

Figure 5.4: Article, Profile and My Saved List - Mobile App

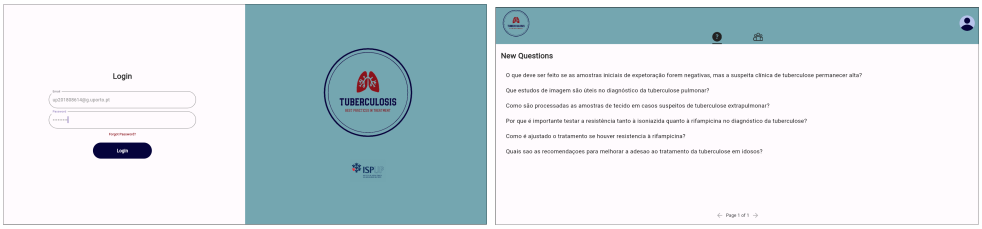
5.1.2 Advisor Demo

During the application demo, the following features were demonstrated: user authentication, question submission, question management, notifications, and bookmarking. The advisor feedback was positive for both overall functionality and user interface.



(a) My Questions List (b) My Question Details (c) Settings

Figure 5.5: My Questions and Settings - Mobile App

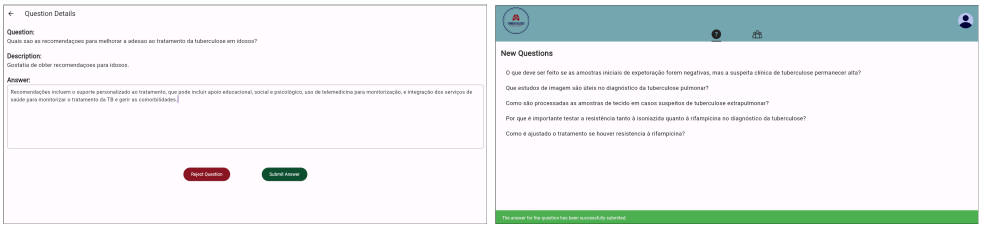


(a) Login (b) New Questions

Figure 5.6: Login and New Questions - Website

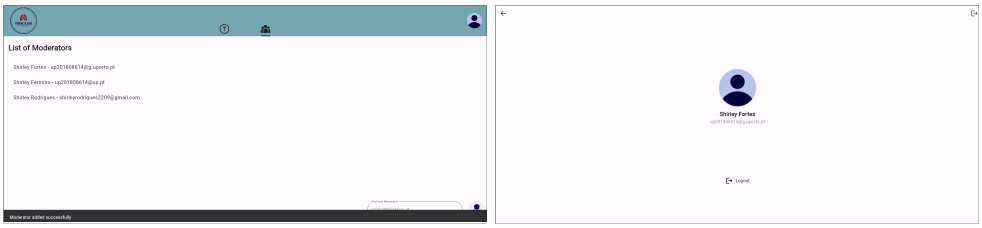
5.2 Discussion

The app has been successfully implemented with all the expected features in place: user authentication, question management, notifications, and bookmarking. The user interface is intuitive, and it has behaved reliably across all tested scenarios.



(a) Question Details and Answer (b) Submit Answer

Figure 5.7: Answer New Question - Website



(a) Add new Moderator (b) Moderator Profile

Figure 5.8: Answer New Question - Website

The m-Health application delivered promising first results on achieving objectives of:

- **Improving Adherence:** Being able to give access on-demand to modern guidelines, this app can improve adherence to standardized TB treatment protocols among health care providers.
- **Speed in the Dissemination of Information:** The application presents a faster and more convenient way of getting expert-reviewed recommendations as opposed to methods followed traditionally, such as specialist service consultations.

Hence we expect application to be motivating enough to use on a routine basis.

5.2.1 Challenges and Limitations

There is no major technical issue that came up during the testing. However, future monitoring might be required regarding possible scalability problems, which are a result of an increased number of users.

Improvements in the directions forward: In subsequent versions, this can be further evolved to reduce the effort of submission and moderation processes with better refinement in the UI elements.

Scalability of the App Deployment: The deployment of the m-Health app can have two meaningfully different implications: scaling up to reach a wider audience or large-scale enhancement of its functionality. This would entail a scaling-up of the m-Health application to get more people via their mobile phones and a scaling-up of its functionality concerning including more infectious diseases or medical specialties.

Potential Impact on Public Health: The scalability of the m-Health app is consistent with global health strategies in use to improve outcomes in TB treatment through information made increasingly accessible through technology.

Long-term Vision: Continued development and adoption of the app could gradually lower TB incidence rates and raise treatment success rates globally.

5.2.2 Conclusion

In conclusion, the preceding review and analysis of the m-Health application to support TB treatment guidelines has the potential to fill the current gaps in disseminating information within healthcare settings. When tested for the first time, the application appeared promising in terms of use and effectiveness. Still, continuous refinement and user feedback would be necessary to ensure sustainability and scaling of impact. Leveraging technology for appropriate use, the application is expected to empower health care workers to provide care in the best way possible and improve the results of TB patients globally.

Chapter 6

Conclusion

6.1 Findings Summary

This research and development project served the objective of implementing a reliable question-and-answer application that is easy to use and features an integrated system for moderator reviews. The system was developed based on Flutter and Firebase in an Model–View–Controller (MVC) pattern to make it easily maintainable and scalable. An overall summative evaluation of the application has confirmed that it meets its primary objectives by allowing user authentication, question management, and an effective notification system. The class diagrams and implementation details presented a well-structured data model with proper integration between different components.

The project also includes the development of a Moderator Web Interface that helps review and manage questions posed by the users. The dual-interface approach ensured user engagement on one end while controlling content quality on the other. The feedback from manual testing and demo provided insights into the application's strengths and areas for improvement.

6.2 Contributions

This project contributed with the following:

- Through the usage of Flutter, it was possible to develop a cross-platform app, allowing the use of mobile and web platforms.
- A moderator interface permits reviewing and taking care of all the user-submitted questions.
- Collections and documents in Firestore database, made possible the ease of management and retrieval of data.

- The intuitive design of the application guarantees smoothness in interaction, which is confirmed via manual testing and feedback.

6.3 Future Work

Although the application can accomplish its primary objectives, there are several potential areas for improvement and enhancement. Incorporating these features will further ensure the application meets the evolving needs of healthcare workers and enhances the user experience.

1. **AI Integration:** Using artificial intelligence and machine learning algorithms to get personalized recommendations for questions and to get a better moderation system. AI could also be used to detect duplicate questions and provide relevant answers to existing questions in the database.
2. **Enhanced Search and Filtering:** Introducing advanced search capabilities, including the use of keywords and filters to help users quickly locate relevant questions and answers. Implementing a system for classification of answers by users through the use of keywords. This will facilitate easier searching and categorization of information.
3. **User Experience Improvements:** Continual refactoring of the user interface for better usability and accessibility based on user feedback. Introducing a voting system that allows users to report errors or outdated information. This will help maintain the accuracy and reliability of the content. Providing an option for users to order articles and questions according to their preferences, such as by relevance, date, or popularity.
4. **Role Management and Permissions:** Introducing a new "Admin" role with special permissions to manage users, oversee content quality, and maintain the overall integrity of the application. Assigning moderators to specific keywords so they receive notifications when questions associated with their designated keywords are submitted. This will ensure prompt and accurate responses.
5. **Multi-Language Support:** Extend the application to be integrated with multiple languages, to reach a wider audience.
6. **Performance Optimization:** Further enhancement of the application's performance, especially for large amounts of data and numerous users.

6.4 Conclusion

This thesis managed to create a fully functional Question and Answer application with Flutter and Firebase as the foundation of an application (and based on the MVC architecture) for supporting online dissemination of best-practices for tuberculosis treatment. The results, as reported in

this dissertation demonstrate that with the use of advanced development frameworks and cloud applications, scalable and maintainable software can be implemented in order to provide an effective, users-friendly application with the potential for a large impact in the health care system.

Bibliography

[1] World Health Organization (WHO). *The End TB Strategy* (cit. on p. 1).

[2] World Health Organization (WHO). *Tuberculosis* (cit. on p. 1).

[3] Patricia Codyre. 'Will an App Fill the Gap? Innovative Technology to Provide Point-of-Care Information'. In: *Frontiers in Public Health* 2 (Feb. 2014), p. 9. DOI: 10.3389/fpubh.2014.00009 (cit. on p. 8).

[4] Shipu Debnath. 'Integrating Information Technology in Healthcare: Recent Developments, Challenges, and Future Prospects for Urban and Regional Health'. In: (July 2023). DOI: 10.30574/wjarr.2023.19.1.1335 (cit. on p. 6).

[5] Michael DiStefano and Harald Schmidt. 'mHealth for Tuberculosis Treatment Adherence: A Framework to Guide Ethical Planning, Implementation, and Evaluation'. In: *Global Health: Science and Practice* (June 2016). DOI: 10.9745/GHSP-D-16-00018 (cit. on p. 7).

[6] Lina Keutzer, Sebastian Wicha and Ulrika Simonsson. 'Mobile Health Apps for Improvement of Tuberculosis Treatment: Descriptive Review'. In: *JMIR mHealth and uHealth* (Apr. 2020). DOI: 10.2196/17246 (cit. on p. 7).

[7] Chen-Lin Lee. 'Exploring the Introduction of Cloud Computing into Medical Information Systems'. In: *Journal of Computers* (2018) (cit. on p. 7).

[8] Charles Lu et al. *An Overview and Case Study of the Clinical AI Model Development Life Cycle for Healthcare Systems*. Mar. 2020 (cit. on p. 7).

[9] Liam McCoy et al. 'Ensuring machine learning for healthcare works for all'. In: *BMJ Health & Care Informatics* 27 (Nov. 2020). DOI: 10.1136/bmjhci-2020-100237 (cit. on p. 7).

[10] Medscape. *Tuberculosis (TB) Treatment & Management*. May 2024. URL: https://emedicine.medscape.com/article/230802-treatment (cit. on p. 5).

[11] World Health Organization. *Global tuberculosis report 2021*. Geneva: World Health Organization, 2021 (cit. on p. 5).

[12] World Health Organization. *Global tuberculosis report 2023*. Geneva: World Health Organization, 2023 (cit. on p. 1).

[13] Hamza Sami Ullah, Samia Aslam and N. Anrjomand. *Blockchain in Healthcare and Medicine: A Contemporary Research of Applications, Challenges, and Future Perspectives*. Mar. 2020 (cit. on p. 7).

[14] Serviço Nacional de Saúde (SNS). *Tuberculosis* (cit. on p. 5).

[15] Zanfina Svirca. *Everything you need to know about MVC architecture* (cit. on p. 41).

[16] tutorialspoint. *MVC Framework - Introduction* (cit. on p. 41).

[17] Raja Vardhan et al. 'Role of mhealth in the management of Tuberculosis: A new approach to improve adherence'. In: *International Journal of Research in Pharmaceutical Sciences* (Sept. 2020) (cit. on p. 7).

[18] Yao Yu et al. 'Blockchain-Based Multi-Role Healthcare Data Sharing System'. In: *2020 IEEE International Conference on E-health Networking, Application & Services (HEALTHCOM)*. 2021. DOI: 10.1109/HEALTHCOM49281.2021.9399028 (cit. on p. 7).