



Final Year Dissertation

Global Health Database System

Author: Naeemullah Khan

Supervisor: Mohammad Hamdan

BSc: Computer Science

22nd April 2021

Table Of CONTENTS

1	Declaration.....	3
	Abstract.....	4
2	Introduction	5
2.1	Context.....	5
2.2	Aims.....	5
2.3	Objectives.....	6
2.4	Relevance	6
2.5	Overview of the Structure.....	7
3	Literature Review	8
3.1	Background	8
3.1.1	Medical History:	8
3.1.2	Medical History Database:	10
3.1.3	Global Health Database System:	10
3.2	Related Work	11
3.2.1	Electronic medical records (EMR) system by Arzon Inc.:	11
3.2.2	Patient Power™:	12
3.2.3	OpenMRS; A Global Medical Records System Collaborative:	13
3.2.4	Personal Medical Database Device:	15
3.2.5	Summary Care Records (SCR):	16
3.2.6	Amazon Comprehend Medical:	17
4	Requirements Analysis.....	18
4.1	Functional Requirements:	19
4.2	Non-Functional Requirements:	19
5	System Design and Implementation	20
5.1	Tools Used:	20
5.1.1	Flutter:	20
5.2	System Architecture:	21
5.2.1	User accounts:	21
5.2.2	Home screen:	23
5.2.3	Profile page:	25
5.2.4	Navigation:	26
5.2.5	Back-end:	26
5.2.6	Libraries Used:	29
5.3	Database Design:	30

5.3.1	Database structure:.....	31
6	Evaluation	33
6.1	Requirements Evaluation:.....	33
6.1.1	Functional Requirements:.....	33
6.1.2	Non-Functional Requirements:.....	34
6.2	System Compatibility Testing:.....	35
6.3	Database Security Testing:.....	36
6.4	UI Testing:	37
7	Conclusion:.....	38
7.1	Project Achievements:	38
7.2	Limitations:	38
7.3	Future Work and Development:	39
8	References	40

1 DECLARATION

I, Naeem Ullah Khan confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Naeem Ullah Khan

Date: 22-04-2021

ABSTRACT

Often there are medical emergencies where the patient is not conscious or responsive. In times like these, the paramedics on scene or the doctor attending the patient usually has no idea of the medical history of the patient. Like what sort of treatment have they been going through? Or if they are allergic to a certain medication etc.

The global health database system is a centralized mobile application that can keep total record of a person's medical history. The idea behind this application is to keep track of user's medical history so it can be accessed by doctors or paramedics on scene in case of emergency, where the patient is not responsive and there is no one else to available to get patient information from. The way it works is that the user will be able to scan their prescriptions using the camera of their smartphone devices, which will automatically update the database with the data collected from the scanned image. The database can be accessed by any certified doctor who is registered on the platform where they can access the patient's information in cases of emergency. Any verified practitioner can apply to have access to an admin account from where they can search for an individual's record using the identification number (i.e. passport number or personal identification number) of an individual to access their complete medical history.

Keywords:

Electronic Medical Record (EMR).

Optical Character Recognition (OCR).

Database Management System (DBMS).

2 INTRODUCTION

2.1 CONTEXT

The global health database system is a centralized medical record aimed to provide an easy access to patient's medical history in case of emergency. It is a database of person's medical history that will be easily available in cases of emergency where it is critical to know about it.

2.2 AIMS

The concept of having the medical history at the relevant practitioner's disposal could help save many lives in cases of emergency. Many hospitals are linked under one network where they can share information with other health care facilities within their network, but that is limited to facilities within the network. If the patient were to visit or brought to a hospital outside of that network, the hospital would have no record of that patient's medical history. The aim of this project is to design a platform where a patient can keep all of their medical history no matter which health care institution they have visited in the past. This can be achieved by providing a centralized system where a user can keep all their medical history in one place and share with other people/practitioners as required. In case of an emergency, a practitioner on scene (like a paramedic) who has access to the special practitioner's account, can access the patient's complete history using their personal identification number. Everyone carries some sort of identification document with them, and a user can use any sort of documentation number to register an account on the platform, which can be later used by a practitioner to access that patient's entire history.

2.3 OBJECTIVES

The objectives of this project are:

- Provide an accessible way for people to keep track of their medical history for free without having to rely on individual medical facility.
- Have a way to keep track of patient's medical records which can be accessed by any health care professional using an individual's identification number they used to register on the platform.
- Maintain a comprehensive medical record which can be shared by any health care professional when necessary.
- Provide a source to be checked by health care personnel in case of an emergency where there is no one else to provide any information regarding the patient. This can be done by a practitioner on the scene which will have a designated practitioner's account, which can allow them to access the patient's medical records using his/her's personal identification document that they are carrying with them.
- Have a centralized EMR system which can be independent from a single organization monopoly and is free to use for everyone from patients to health care professionals.

2.4 RELEVANCE

Most people have often visited many different hospitals for various reasons throughout their lives.

Whether it be a common cold or a regular checkup. The patient records are usually saved in the hospital's database and they are kept confidential for that hospital's use. Any other health care facility that the patient visits other than that hospital will have no record of that patient.

2.5 OVERVIEW OF THE STRUCTURE

This section will shed light on the topics we are going to discuss in this document.

Introduction: Summarization of this project, as well as aims and objectives along with the project scope and relevance is discussed in this section.

Literature Review: We are going to briefly describe what our project is and discuss the technical research about what other alternatives are available similar to this project. We will also critically analyze each research projects to figure out the differences between them and our system.

Requirements Analysis: Details regarding the project use case scenarios and functional/non-functional requirements. We are along going to conduct MoSCoW analysis on the requirements to further categorize them into meaningful structure.

Evaluation Strategy: In this section, we are going to be talking about the strategic plan of progression and discussing our policies to achieve the goal.

Project Management: In this section, we are going to be going through risk analysis of the project, project plan code of conduct.

3 LITERATURE REVIEW

3.1 BACKGROUND

In this subchapter, we are going to describe what medical history is and discuss the problems related to current method of storage for them. Then we are going to introduce our approach to tackle these problems.

3.1.1 Medical History:

Medical history is a traditional paper document or electronically stored variant of it which represents a narrative or total record of a patient's past diseases, injuries, treatments, and other relevant medical facts. It is used to identify the patient as well as containing the data of the patient's visitor at a particular hospital or clinic. It is part of a patient's history which can help the practitioner in determining the risk factors like allergies against certain medications among other things. Traditionally, there are two ways the medical history for a patient can be recorded by a practitioner:

- Medically relevant complaints recorded as described by the patient, which are known as symptoms.
- Observations made by the practitioner to come up with the diagnoses based on symptoms present, also known as medical signs. [1]

The way medical history of a patient is recorded can depend on the practitioner and situation. For example, a paramedic on the field would only record patient's name, currently observable injuries or complaints, and record as much information about the patient as they can on the spot regarding allergies etc. A general physician would do a physical examination and try to come up with the diagnoses based on symptoms to formulate a plan for treatment. While a psychiatric doctor generally tends to

have a long session with the patient to discuss about their life before coming up with a plan for psychiatric illnesses. Based on these practices, the practitioner can record the observations made and management strategies they have come up with into their hospital/clinic's local database either electronically or on paper.

For the most part, medical history is recorded in a traditional paper-based format which is subject to misplacement, degradation, and sometimes disorganization. The practitioner collects the data from the patient which contains the diagnoses and observations made from that visit and any other follow-ups regarding those symptoms. Usually it contains the checkup summary, vaccination records, medicines prescribed, and any sort of non-invasive tests results like ultrasound tests, x-rays etc. Unless it is a case where more than one practitioner is required to check the patient like a specialist doctor at the same clinic, the medical record kept by the practitioner only exist at the relevant practitioner's clinic/hospital and cannot be accessed by the other clinics/hospitals.

Modern advancements in computer technology has helped with the storage and management of medical history management. Most if not all hospitals and clinics have their own online or local database of medical records system which is used to collect and store patient's medical history in. However, as stated in the previous paragraph, these records are limited to that facility's own database and is not accessible by other institutions. In case of a patient who has visited multiple health care providers, or a patient who has had different health care insurances throughout their life or patients with no medical insurance at all, there is no system to have their centralized medical history throughout all the health care institutions. [2] As such, a practitioner does not have a full view of their unified medical history which is necessary to initiate a proper diagnosis and its treatment. This is particularly critical in cases where the patient is not responsive and does not have any other guardian or relative around, for example in cases of a car accident, or a case where patient has had a heart attack in public and the patient was assisted by strangers who called the emergency services. In such cases, the emergency

services on the scene will have no idea of the patient's medical history and will have to resort to treatment with extreme caution.

3.1.2 Medical History Database:

Database management system (DBMS) is a program that gives user the ability to design a structure of a database, and allow them to create, retrieve and modify data in that database as well as the ability to control access to it. [3] In the field of health services, there are two different kinds of approaches available for implementing a centralized medical history DBMS. The first approach requires all the equipment and software related to the system to be managed by a single manufacturer. This approach can help centralize the system throughout all the health care institutions, but it has a major problem with its implementation; it requires a complete re-equipment of all the current infrastructure of every establishment which is a major expenditure. Another approach is to integrate a common DBMS with each individual health care institution's infrastructure. This approach has a compatibility problem as not every system is going to be the same. Different infrastructures rely on different equipment and software such as different operating system and different structure of data recording etc.

3.1.3 Global Health Database System:

The approach we have created avoids both major problems discussed in the previous subsection by relying on a database that is completely isolated and independent of every health care institution's database. The main principle of this approach is that the patient will be responsible for updating their medical records which are independent from the hospital's record. Currently many health care institutions provide a detailed summary of the checkup to the patient on every visit. So, when a patient visits a hospital and gets their treatment plan, they will be able to scan those details from the summary provided by the doctor using their smart phones, and the data will automatically be collected by the application and uploaded to our centralized database. The patient will also have the option to manually

write any information missing from the details scanned if required before uploading it to the database.

This approach assures that all the patient details are stored in one location and it can be accessed by the relevant health care institution in case the patient is in a critical condition and no other details can be found about them. The patient will also be able to send this history to any doctor they are visiting to provide the doctor with a comprehensive data of their past medical history for accurate diagnosis. This approach has minimal expenditure cost for the existing health care infrastructure as it is an independent system from the current implementation in every health care institute.

3.2 RELATED WORK

In this subchapter, we are going to discuss the research done towards similar concepts discussing the relevance and comparing them with our solution to figure out the relevance of this project.

3.2.1 Electronic medical records (EMR) system by Arzon Inc.:

Jae A. Evans [4] proposed an idea about a system that can convert all the paper-based work in a hospital into an electronic system. This idea was later patented by Arzon Inc. and implemented in the hospitals which were under the umbrella of Arzon Inc. The system comprised of a small touch screen device which could be controlled with a provided pen. The device was used by doctors to record data on it when performing medical examination, prescribing treatment, or recording test results to eliminate the use of traditional paper-based approach of recording data. This data was then recorded into a centralized database which could be accessed by any doctor working within the Arzon Inc.' network of hospitals if necessary. The product claimed to be easy to use which could help eliminate or supplement the traditional way medical records were created and managed.

- **Critical Analysis:**

Although this system was revolutionary at the time, it came with a major expenditure cost of implementing the hardware into each individual hospital. Another drawback of the system was expandability and integration with the current running infrastructure. Because it was based on Windows NT, it was not compatible with other software of that generation that generally were based on MS Dos. Unless the hardware itself was bought, there was no other way to integrate it with the current running infrastructure. It also lacked the ability to share the medical records with health care facilities outside of Arzon Inc. which did not adopt to this system. So, unless the health care institution was using this system, they would not have the ability to create, manage, or share patients' medical record with the centralized database. The main drawback of this system was the inexperience of doctors when it came to using computers as it was not a norm in that era. The doctors wanted to examine many patients as fast as they could and using a system they were not familiar with caused more harm than it did good. [5]

3.2.2 Patient Power™:

Elliot Segal, Mark Klein, and Ernest Kinchen [6] designed a system in 2001 which they called Patient Power. The system was designed to keep a self-managed medical record of the patients with the help of their physicians. The patients would be able to create, manage and share their medical records such as x-ray images, electrocardiogram (EKG) results, ultrasound images etc. within the system and will have the ability to share it with their doctors when necessary. The system would require the use of a scanner-like machine built for this system to scan the results of the paper document and upload it to the relevant medical institute's database. The system would also have the capability to either store the document into the image server or display it on a screen somewhere near-by. To transmit the data over the internet, the system required an ISP facilitated web server for communication. To facilitate the patients

in organizing and keeping the most relevant data in the database, the medical practitioners would be encouraged by monetary incentives to engage in two to three short annual sessions with the patients.

- **Critical Analysis:**

Much like the system discussed previously, this system also uses a special kind of hardware to perform its tasks. This presents us with the same expenditure cost on the side of hospitals and requires a complete overhaul of their current infrastructure. The records availability is also another issue with the system as the records are kept in the hospital's local database, they are only accessible by personnel working in that hospital and there is no way to share these medical records with anyone outside of the said hospital's network. A major drawback of this system is that the practitioner is required to facilitate the patient in recording their medical records which could be a burden on a practitioner's already packed schedule, and they might feel discouraged to use the system at all.

3.2.3 OpenMRS; A Global Medical Records System Collaborative:

The work on OpenMRS began in 2004 [7] to support the development of a new EMR system for an initiative launched in Kenya to strengthen the HIV/AIDs health systems. [8] This later converted the project into an open-source community after their collaboration with another similar initiative called Partners-in-Hand. [9] The original EMR software for HIV/AIDs initiative was deployed in 2006 under the name of AMPATH Medical Record System – AMRS and holds over 300,000 patients' records. [10] Today, OpenMRS has become one of the largest opensource EMR systems in the world and it is freely available for any health care institution to use. What initially began to support an initiative has turned out to be one of the largest open-source communities in the field of health care services. It operates under the Mozilla Public License version 2.0 with an added "Healthcare Disclaimer" term, which means that the users who want to use the software can freely access the source code under the same license to modify and add functionalities to it as they require. OpenMRS is designed on a conceptual database structure

which does not limit the user to an actual type of data collection method or format, which means it can be customized for various uses based on the requirements. It also allows the system administrators to make their own “dictionary” which is a set of different predefined options to optimize the data collection form.

- **Critical Analysis:**

This system is one of the best ERM systems currently active at the time of writing, and it is even open source which means it can further build upon by developers who might want to use it. While the system does provide online database for centralized medical records, these records are usually for a few limited fields of study like HIV or Tuberculosis (TB) patients. For example, it will keep the record of an HIV patient that is currently undergoing treatment with a health care institution that is running this system.

Another major drawback of this system is that it can be very slow in retrieving data from the database if the dataset is big, as the system uses EAV (entity attribute value) data model for it. EAV models are used to provide the user a way of adding new variables to the system in a “dictionary”, but this approach becomes very expensive to query the database due to its inefficiency and complexity. [11] EAV stores the data vertically as showing in fig.1 as opposed to clinical data which is usually a vertical list of key-pair values. Data is consumed in a traditional key-pair value where attributes are the column names and values are the rows in table, hence the design of the EAV requires the data to be pivoted every time the data is retrieved. This approach may not affect smaller datasets, but in a larger dataset this can be a very taxing operation for the CPU as it is the nature of EAV that its table grows exponentially over time attribute is stored in an individual row. And in OpenMRS, there can be multiple columns depending on the system set up by administrators, which means a single value is stored across different columns.

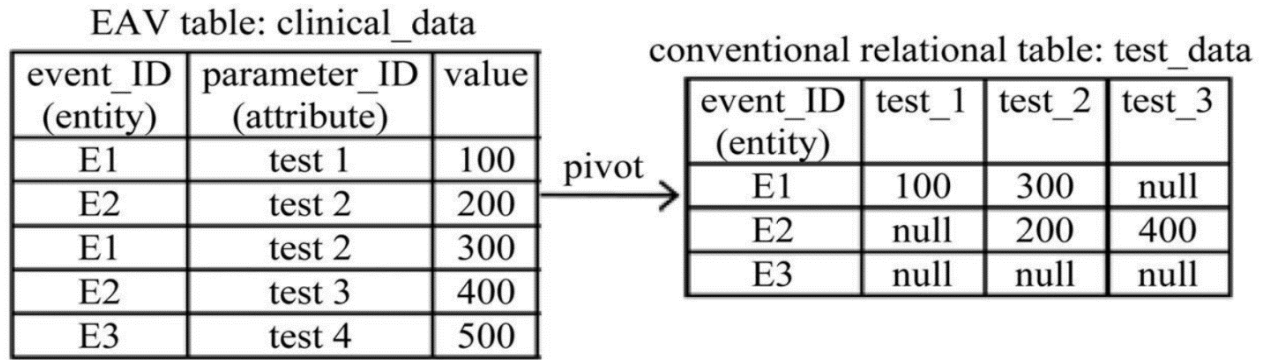


Fig.1: Demonstrates the vertical EAV table on the left and its pivoted counterpart on the right to obtain “test_1”, “test_2” and “test_3” parameters [11].

3.2.4 Personal Medical Database Device:

Thomas Alten [12] designed a system based on a small physical device to which could be used to store patient’s medical history. The device would consist of many different smaller components such as memory module estimated to be around 200kB to hold the medical records, connector to link with another reading/writing device that would be situated at the practitioner’s facility, and a controller which would contain the device firmware etc. To retrieve medical data store in the device, the patient would have to visit the health care institution and give the device to a staff member who would connect it with the read/write device module located at the practitioner’s office. Once the device has done its operations, the data is automatically synced with that health care institution’s database. Similarly, to write the data after medical services have been rendered to the patient, a staff member would have to connect the medical database device with their own read/write module to download the new data into the device which will save it in its onboard memory.

- **Critical Analysis:**

The system does keep the medical records of a patient on its onboard memory, but it comes with its own problems. The major one being the ability to operate only when it is linked with its counterpart at the health care facility, without it, this device has no other efficient way of retrieving or updating the data. So, if a health care facility that the patient is visiting does not have the module to link with the

device, it cannot get any data on the patient's medical history. And since this device needs to be operated in the presence of a facility's staff member, it limits its usage to physical visits only without any capability of using the device online. The device itself is very outdated according to current standards. In the age where everything is online, a system where you must carry a device with you for every time you have to use it seems bizarre, while the 200kB memory is severely insufficient for data storage specially if it has been used several times.

3.2.5 Summary Care Records (SCR):

In United Kingdom, National Health Service (NHS) introduced a system called Summary Care Records (SCR) which creates a global record of patient when they visit any NHS approved health care facility and uploads it on a central database to be accessed by other facilities. These records generally contain the patient's personal information such as their age, address, name, lifestyle information such as if the patient is a smoker or not, if the patient drinks alcohol frequently etc. It also contains information regarding patient's health conditions, diagnoses and medicines prescribed in that visit and any tests conducted and their results. [13]The users also have an option to opt into adding additional information to their electronic records such as their personal information, immunizations etc. The patients also have an option to share their special needs with other health care facilities such as identifying that the patient is crippled and may require a wheelchair, or if the patient is blind and require additional help reaching the facility.

- **Critical Analysis:**

This system is a perfect example of what we are trying to achieve with our system. It allows the patient's medical history to be readily available to every health care facility in the United Kingdom and contains a good amount of information about the patient and their needs. The only downside is that the system is available only for UK and is not available for any other country. What we are trying to achieve can be

implemented and used in every country as the patient's records are not limited to the country they are living in and the patient can manage their own records where in SCR, the patient records are updated by the NHS health care facility that the patient visits. So, if a user visits any medical facility outside of NHS's coverage (for example a hospital in a foreign country), the user records will not be updated in SCR.

3.2.6 Amazon Comprehend Medical:

Amazon Comprehend Medical is a natural language processing (NLP) introduced as part of Amazon Web Services (AWS) package which uses machine learning to extract data related to medical information. It is an API service which requires to prior machine learning experience and it is ready to be used by any one with knowledge of AWS platform. Amazon claims that it can be used to accurately extract information such as medical conditions, tests, diagnoses, medicines prescribed while retaining the context of information extracted. It is a pay-per-use subscription-based service which requires no machine learning models to build, train, or deploy. [14]

- **Critical Analysis:**

The main problem with this system is that it is a service that needs to be utilized rather than being a complete solution. An average end-user with no I.T. knowledge will not be able to make use of it, as it is a service that requires to be utilized by other programs. It is a platform that *can* be used to create an EMR, but it is not one by itself. Another problem with this approach is that it is not free to use. It is part of Amazon's AWS package and requires a monthly subscription fee to use. That alone separates our idea from it as it is free to use by everyone and does not require any I.T. knowledge to setup or use, and can be utilized as a complete user-friendly package by everyone.

4 REQUIREMENTS ANALYSIS

Requirements analysis is a vital part of this report to better understand what our system is going to be like. To figure out how our system is going to be built and how it should work, we need to map out what is the most necessary part of it in terms of core functionality and what is not. That is where requirements analysis comes in. Requirements can be broken down into two categories:

Functional requirements: These are the functionalities that the system *must* have to work properly. Without it, the system will not work as expected. For example, “a smart phone device *must* have the functionality to make phone calls”. Without it, it is not a phone anymore.

Non-Functional Requirements: These are the features of the system that would bring quality of life changes to the system, but the system would work even without these features. For example, “a smart phone *could* have expandable storage”. Having this feature in a smart phone is helpful, but not having it does not have any effect on its performance in any way.

We are going to use MoSCoW analysis to further breakdown the requirements into two categories:

- **Must:** These are the most important requirements, and they are essential for building our system. Without them, our system would not function as we intend it to.
- **Should:** These are the important functionalities that the system should have but they do not have to be satisfied in the way mentioned in our analysis. Even if these requirements are not met, it would not have a huge impact on system performance like missing a *must* functionality.
- **Could:** These functionalities are the additional features of the system which do not impact the system’s ability to function. These are the quality-of-life improvements to our system.

4.1 FUNCTIONAL REQUIREMENTS:

ID	Description	MoSCoW
FR-1	The system must be able to access and use a smartphone device's camera.	Must
FR-2	The system must have user authentication system.	Must
FR-2.1	The system must have different authentication systems for practitioners and patients.	Must
FR-3	The system must be able to extract data from an image taking through user's smartphone camera.	Must
FR-4	The system must be linked to user's personal identification document(s).	Must
FR-5	The system should allow the user to send their medical history to a practitioner.	Should

4.2 NON-FUNCTIONAL REQUIREMENTS:

ID	Description	MoSCoW
NFR-1	The system could be optimized to handle querying a large database.	Could
NFR-2	The system must be secure and protect personal data.	Must
NFR-3	The system should have user management system so individual users can modify their details.	Should
NFR-4	The system's GUI must be user friendly.	Must
NFR-5	The system must give user the opportunity to review and modify results extracted from the image scanned before uploading them to the database.	Must

5 SYSTEM DESIGN AND IMPLEMENTATION

The system made for this research is a mobile application that is easily accessible for everyone. There are two major mobile application platforms; Android and Apple based applications. For the purpose of this research, the Flutter SDK was used to build the Android application only as iOS development requires iOS operating system which we did not have access to, and we could not make it work on a virtual machine.

5.1 TOOLS USED:

For Android application development, we used Flutter version 2.0.4 with Dart 2.12.4 targeting Android SDK version 29. For the IDE, we opted to use IntelliJ IDEA Ultimate 2019.4 rather than Android Studio as it is light-weight and easy to use.

5.1.1 Flutter:

Flutter SDK is an open-source mobile UI framework developed by Google for building Android, iOS, Windows, Linux, and web applications using the same code base. Which means a single project can be compiled for both Android and iOS and other platforms. It uses Dart programming language which is similar to Kotlin and Swift, and it is mainly used for the development of 2D applications. It uses the concept of “widgets” which are reusable blocks of code to build interactive designs. There are several widgets provided by flutter such as a collection of buttons, containers, and forms etc., but user has the option to build their own custom widgets tailored to their needs. Widgets can be single entity widgets which are sufficient on their own such as buttons or can be widgets that require one or more widgets to be a part of it such as container widget which is the main body of a screen, and requires a child widget to be used with it. Every widget has their own properties and using these properties, we can style or modify them.

```

import 'package:flutter/material.dart';
import 'package:ghms/constants.dart';

class TextFieldContainer extends StatelessWidget {
  // Properties.
  final Widget child;
  // Constructor.
  const TextFieldContainer({Key key, this.child}) : super(key: key);
  // Build method that is called every time the page is loaded.
  @override
  Widget build(BuildContext context) {
    // Get device screen size.
    Size size = MediaQuery.of(context).size;

    return Container(
      margin: EdgeInsets.symmetric(vertical: 10),
      padding: EdgeInsets.symmetric(horizontal: 20, vertical: 5),
      width: size.width * 0.8,
      // Background color and border for the widget.
      decoration: BoxDecoration(
        color: kPrimaryLightColor,
        borderRadius: BorderRadius.circular(29),
      ), // BoxDecoration
      // Child widget passed to this class.
      child: child,
    ); // Container
  }
}

```

Fig.2: A simple custom widget class in Flutter used to create a static Form's background design.

5.2 SYSTEM ARCHITECTURE:

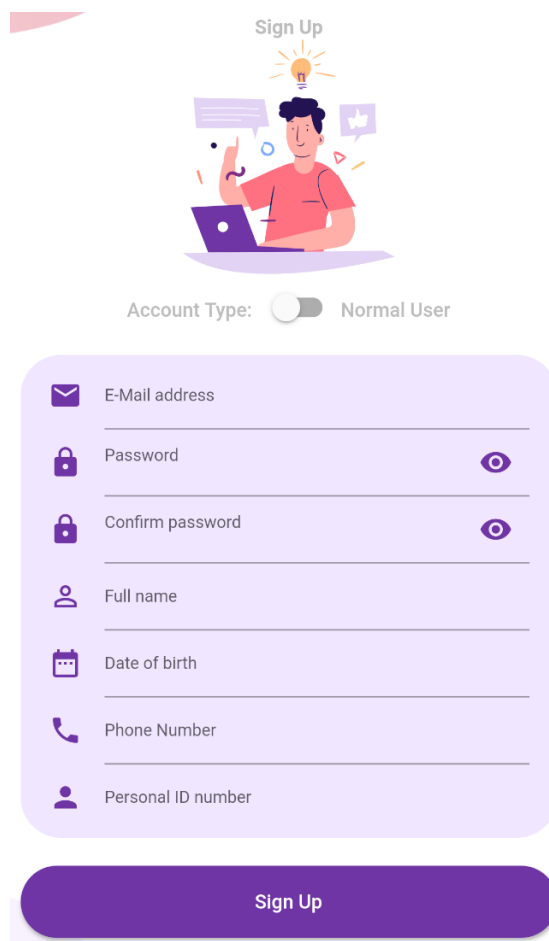
In order to use the application, the user must have an account on the platform that they must use to log in. If a user does not have an account, they can get one by using the sign-up option and fill out all the required fields on screen. The user will also have the option to change their password in case they forget their password.

5.2.1 User accounts:

There are two types of accounts that a user can get: User and Practitioner. A User account is a normal account which a user can use to upload and manage their medical records. To sign up for a User account, the user must sign up with a valid e-mail address as it is required to verify the e-mail address and it is used as their username for login. The user must also provide a valid personal identification

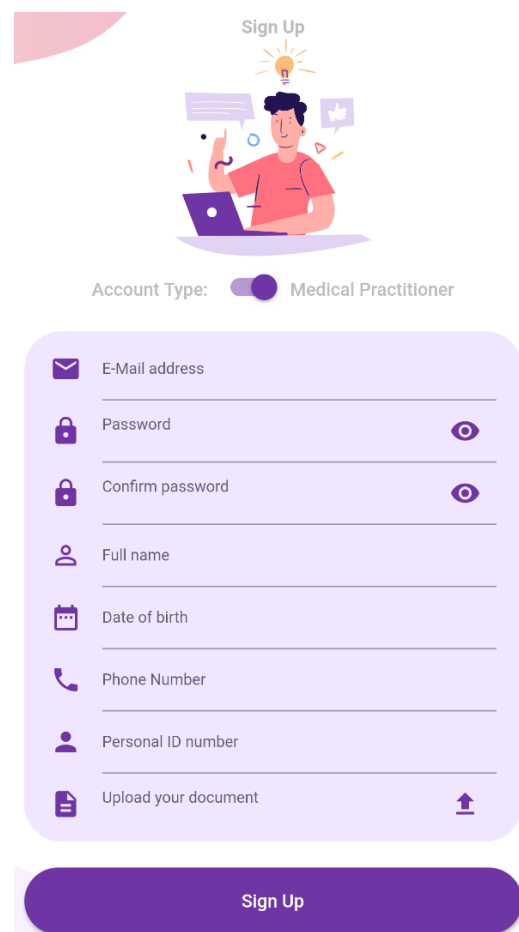
document number such as passport number or driving license number etc. This is required as it is used to filter and search for this particular user's medical records by practitioners.

Another type of account is Practitioner. The signup procedure is the same as User account, except the user must fill out an additional field for verification by uploading a document. This document must be a document that validates that the person is indeed a medical professional. Upon receiving the document in system, the system's staff members must manually check and validate the authenticity of the document. Upon verification, the user is given access to the Practitioner account where they can search for other users' medical history by using a user's personal identification number.



The image shows a 'Sign Up' page for a 'Normal User'. At the top, there is an illustration of a person with a lightbulb idea, a speech bubble, and a thumbs up icon. Below the illustration, the 'Account Type' is set to 'Normal User' with a toggle switch. The form fields are: E-Mail address, Password (with an eye icon to toggle visibility), Confirm password (with an eye icon to toggle visibility), Full name, Date of birth, Phone Number, and Personal ID number. A large purple 'Sign Up' button is at the bottom.

Fig.3: A User account sign up page.



The image shows a 'Sign Up' page for a 'Medical Practitioner'. At the top, there is an illustration of a person with a lightbulb idea, a speech bubble, and a thumbs up icon. Below the illustration, the 'Account Type' is set to 'Medical Practitioner' with a toggle switch. The form fields are: E-Mail address, Password (with an eye icon to toggle visibility), Confirm password (with an eye icon to toggle visibility), Full name, Date of birth, Phone Number, Personal ID number, and Upload your document (with an upload icon). A large purple 'Sign Up' button is at the bottom.

Fig.4: A Practitioner account sign up page.

5.2.2 Home screen:

Normal User account: Once the normal User has logged in, they are taken to the homepage screen where all their previously submitted medical records are populated from the database and shown in their respective cards. Each card contains the date of visit, hospital name and diagnoses. The user can scroll through the list of medical records and can tap onto any card to show the list of medicines prescribed for that medical record. On the same page, the user has option to add a new medical record by pressing the “Add new record” button. This will take them to a page with a form that they must fill before submitting to add a new record. The user must fill out all the fields before submitting the medical record, otherwise the system will throw out an error and prevent user from submitting the medical record. Users also have the option to send individual medical records through email by pressing the “send” button next to the hospital name, which brings in a dialog to enter email address.

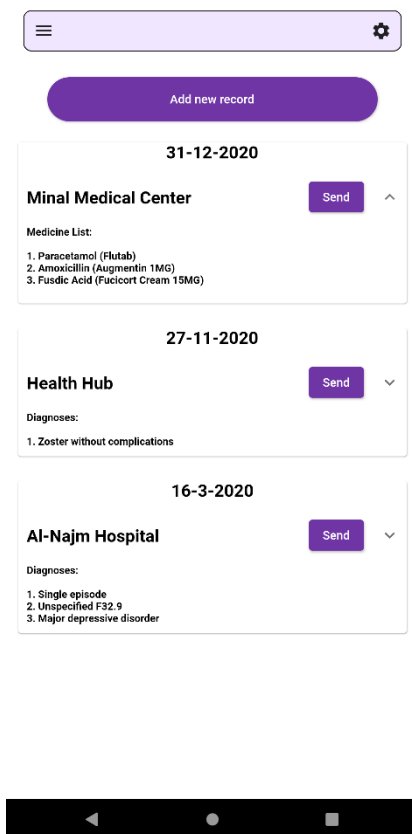


Fig.5: Homepage screen where user can add or check their medical records.



Fig.6: Add new record screen where user can add a new medical record.

Practitioner account:

For the practitioner's account, once the user logs in, they are given a search bar where they can enter a patient's personal ID number (which can be any identification document they used to register with), and they will get to see all of patient's medical records and patient details such as patient's name, phone number and date of birth. Just like the normal User's account, the user is able to scroll through the records and see the diagnoses, medicines list and hospital name along with the date of visit for each record. This account only has read access to these records and the records cannot be modified in any way using this account. If an incorrect patient's personal ID is given, the system will notify the user that no records were found using that personal ID.

The screenshot displays the interface for a practitioner's account. At the top, there is a purple header bar containing a hamburger menu icon on the left and a settings gear icon on the right. Below the header, the text 'VT4117241' is visible. A prominent purple rounded rectangle contains the word 'Search'. Underneath this is a light purple box displaying patient information: 'Patient's name: Naeem Khan', 'Patient's phone #: 0526698242', and 'Patient's DOB: 16-7-1990'. The main content area lists three medical records, each in a white box with a light purple header. The first record is dated '31-12-2020' and from 'Minal Medical Center', with diagnoses '1. Zoster without complications' and '2. Cellulitis of trunk'. The second record is dated '27-11-2020' and from 'Health Hub', with diagnosis '1. Zoster without complications'. The third record is dated '16-3-2020' and from 'Al-Najm Hospital', with diagnoses '1. Single episode', '2. Unspecified F32.9', and '3. Major depressive disorder'. At the bottom of the screen is a black navigation bar with three icons: a back arrow, a home circle, and a square.

Fig.7: Practitioner's account homepage screen where they can search for patient records.

5.2.3 Profile page:

Once the user signs up, all the details they provide are recorded into the Firestore database including their full name, date of birth, phone number, personal ID etc. In the profile page, these records are fetched from the Firestore database and are populated on screen for the user to check and modify. Both User and Practitioner accounts can access this page and can modify any entry using the edit button on top right corner of the screen. Once the modification is done, the edited fields are updated in the database and the user is shown a popup notification, informing them about the successful update before the page is refreshed to show the new user details. The only thing users cannot edit is their e-mail address as it is used as the username for login, and it is only displayed in this screen as read-only text.

The image shows a mobile application interface for a user's profile. At the top, there is a header bar with a hamburger menu icon on the left and a settings gear icon on the right. Below the header is a white card with rounded corners. The card has a title 'Personal Information' in bold black text, followed by a small edit icon (a pencil). The card contains five form fields, each with a label and a value: 'Full Name:' with 'Naeem Khan', 'DOB:' with '7-4-2021', 'Phone Number:' with '55269824', 'ID Number:' with '24trwgasdg', and 'E-mail:' with 'naeemukhan89@gmail.com'. The card is set against a light purple background. At the bottom of the screen, there is a black navigation bar with three white icons: a back arrow, a home circle, and a recent apps square.

Field	Value
Full Name:	Naeem Khan
DOB:	7-4-2021
Phone Number:	55269824
ID Number:	24trwgasdg
E-mail:	naeemukhan89@gmail.com

Fig.8: Profile page where user can check or update their details.

5.2.4 Navigation:

To navigate between pages, the users can use the hamburger menu located on the top-left side of the screen. The users can navigate to medical records screen, profile screen or sign out of the application using this menu. The menu also contains a welcome message for user displaying their full name which is retrieved from the Firestore database.

5.2.5 Back-end:

Our application uses the BLoC design pattern. BLoC stands for **B**usiness **L**ogic **C**omponent and it was introduced by Google in DartCon2018 [15]. The purpose of this design strategy is to create UI elements independent from the business logic, and that is what we have done with our application. Since flutter uses Widgets to create UI elements which are immutable, the data handling part is left out to streams/listeners which control the flow of data by passing it onto next widget when requested.

Authentication:

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        Provider<AuthenticationService>(
          create: (_) => AuthenticationService(FirebaseAuth.instance),
        ),
        StreamProvider(
          create: (context) =>
            context.read<AuthenticationService>().authStateChanges,
          initialData: null,
        )
      ],
      child: ...
      home: AuthenticationWrapper(),
    ),
  );
}
```

Before we talk about how authentication is handled, let us first look into how the root widget which is called by the `main()` method is defined. It is a stateless widget which returns two different streams: *Provider* which is the stream handler we use for Firebase authentication. This stream handler initializes our *AuthenticationService* class and makes it available to be used throughout the application. Next, we use the *StreamProvider* to call the initialized *AuthenticationService* class and listen to the `authStateChanges` method provided by the Firebase library, which notifies all the listeners in case an event happens in the database, for example, a user login or sign out. We use these listeners throughout our project to get the user information such as user's unique ID which is part of our Firestore database where other user information such as medical records are stored, and so on.

Then we have the *AuthenticationService* class which is responsible for calling the `Firebase_auth` library to sign in, sign up users after processing the data it receives by the user input. The class is initialized when the application is loaded by the *Provider* stream, and then later called in the login and sign-up screens when the user wishes to login and sign up. The main benefit of using `firebase_auth` library is that it handles all the authentication and error handling on Google's Firebase and we just have to worry about input sanitization and not worry about authentication errors. All the methods in this class return a string of message returned by Firebase and we just display those errors on the screen. For example, if a user tries to sign up with an invalid e-mail address, Firebase returns an error with "Invalid e-mail format, please use a correct e-mail address", and we simply display that message to user on screen giving them the information about the error and how to fix it.

The *MyApp* class which is the root widget called by the `main()` function and it is first executed when the application starts; calls the *AuthenticationWrapper* class which checks the *emailVerified* flag in Firebase for the current logged in user. If the user did not verify their email, they will be redirected to a page prompting them to verify their email before they can use the application. Otherwise, if the *emailVerified*

flag is set to true, this class will redirect user to the home screen of the application according to their account type. If the user is not logged in at all, it redirects them to the welcome screen where they have the option to login or sign up. This is to prevent anonymous access to the application and prevent users from signing up using invalid email addresses which might prompt people with malicious intent to create spam accounts to overload the database.

Data privacy:

For user authentication, we use Firebase which contains all the user authentication data such as username, sign-in date etc. Our rules structure for Firebase is to allow read, write, create, or update by our application only, which prevents anyone else from making any changes to the database in case they break into the system. For Firestore, which hosts all the user's personal data such as full name, personal ID number, phone number and medical records, we use the following rule set:

```

1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      match /{document=**} {
5        //Allow read, write only if signed in
6        allow read: if isSignedIn();
7        allow write: if isSignedIn();
8        // Allow update only if the uid matches (same user)
9        allow update: if isSignedIn() && request.auth.uid == resource.data.uid;
10       // Allow delete only if the uid matches (same user)
11       allow delete: if isSignedIn() && request.auth.uid == resource.data.uid;
12     }
13   }
14 }
15
16 function isSignedIn() {
17   return request.auth.uid != null;
18 }

```

Fig.9: Firestore rule set which restricts the user access to all the documents/collections.

The ruleset show above allows users to read and write only if they are signed in. Which means anyone who is not signed in will not be able to see or create any medical records, and they will not be able to see any user profile details or edit them. The other set of rules described in the document shown above

is to allow database entries to be updated or deleted only if a user is signed in, and they have the same document/collection ID as their user ID. Since each user's collection's ID is the same as their unique user ID generated from Firestore, this prevents users from updating or deleting data that does not belong to them. For example, if a user signs up on the system, a unique user ID is generated for them in Firebase which is linked with their Firestore data. Which means they can only update or delete that data.

5.2.6 Libraries Used:

Firebase_core & firebase_auth: These two libraries are the controller libraries which lets our application connect to the Firebase authentication service. This library is implemented in this project to take care of user authentication and access logged in user's details.

Cloud_firestore: Just like the libraries mentioned above, this is a controller library for Firestore database which lets us integrate Firestore database with our application.

Provider: In Flutter, the widgets are immutable. That means if we go from screen A to screen B, we will lose all the information related to screen A as it is destroyed before the screen B is rendered on screen. Provider is a wrapper around InheritedWidget class which makes it easy to pass information or states between different screens or classes. For example, in our application, the provider library helps us preserve the Firebase.instance variable which is invoked the moment user is logged in so we can access current logged in user's information throughout the application.

Expandable: This library is a widget which can be collapsed or expanded. This library is used in the *ResultsCard* class which creates the widget in which the user's medical records are shown on the main page. This library was used because of the customization options provided by it which made the *ResultsCard* widget easy to use.

Image_picker: Image picker is library used to select an existing image from phone's gallery or take a new picture with the phone's camera. The main benefit of this library is that it does all the error handling and stream handling for us and disposes all the data after it has been used to avoid any memory leaks.

Flutter_svg: This library is responsible for drawing SVG images and some Android VectorDrawable files in a widget. We have used this library in the welcome screen to draw .svg assets on the screen.

Flutter_email_send: A simple library that sends email using the native e-mail application. This library is used in the home screen for normal user accounts where they can send their records through e-mail to anyone they specify.

5.3 DATABASE DESIGN:

Traditional relational schema-based databases scale vertically, meaning the more data they have, the higher time it would take to query it. To upgrade these sorts of databases, their system needs to be shut down before more servers are added and linked to them which results in a down time for the application that is utilizing them. Since our application's operation is critical and cannot afford to have any downtime, we have opted to use Firestore database which is part of Google's Firebase application development platform.

Firestore is a NoSQL database that uses "documents" and "collections" to store data rather than the traditional NoSQL way of using key-value pair to store data in json objects. Each database has a root collection, under which there can be many documents. Each document must have a unique ID and can have different fields such as String, Integers etc., and they may even contain collections under them. The reason Firestore was chosen as our database is due to its reliability and scalability. It scales horizontally and supports indexing of the documents and collections, which makes the queries faster compared to schema-based databases as querying in Firestore will get you only the data you requested, meaning if

you were to query a field in a document, it will only get that field and not go through any sub-collections under that document unless specifically queried for it. “Queries scale to the size of the result set, not the data set” as it is mentioned by the developers of Firebase, [16] this is due to the hierarchical structure of this database which makes it easy to grab the data that you want without going through the entire table like it would in a schema-based database. So, if we were to query our database in Firestore to get us top 10 pizzas in our country, it would take us the same amount of time whether we have 10 entries in our database or 10,000,000 entries. [16] This makes Firestore an ideal candidate for choice of database as we expect our database to grow exponentially and Firestore is the best choice for us as it is more reliable, secure, and scalable.

5.3.1 Database structure:

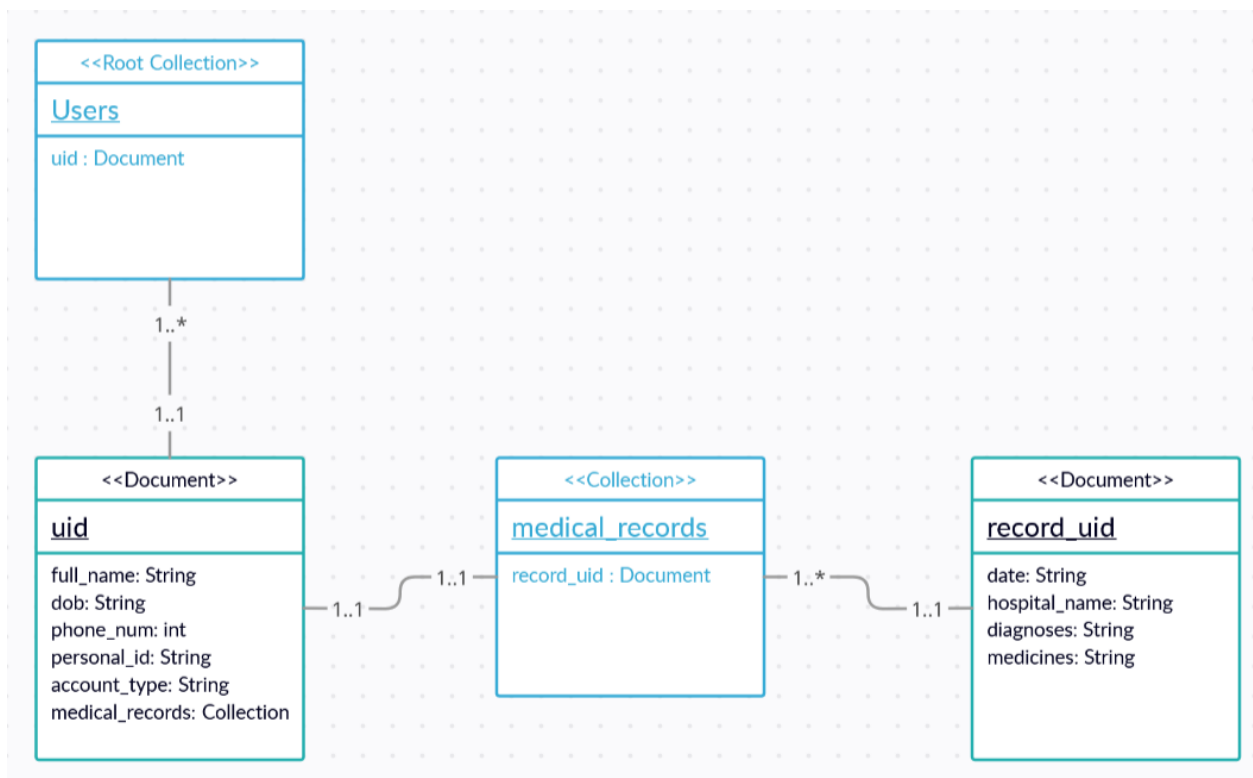


Fig.10: ER diagram of Firestore database for the system.

For user authentication, we are using the Firebase authentication system for our application as it provides us a way to sign up, login and sign out users using their own library in Flutter, and it handles all

the security and error handling from Google's platform. Whenever a user signs up successfully on the platform, a user token is generated and returned as an object in Flutter which also contains the unique user ID randomly generated for this user by Firebase auth. Using this uID (user ID generated in the previous step), we create a document in Firestore which is named after the uID of this new user. Under this document, we store all this user's information such as their full name, phone number, personal ID number etc., along with a collection named "medical_history". Whenever this user adds a new medical record, it will be stored as a new document under the medical_history collection for this user. Each new document under medical_history collection will have a randomly generated ID from the system to make sure there are no duplicates.

6 EVALUATION

In this chapter, we are going to discuss the results acquired from testing the system and the methods used to perform these tests. Due to Covid-19 and social distancing, we could not perform tests involving human subjects and thus could not collect user input on the system design, but we tried to make up for it using unit testing and other evaluation strategies.

6.1 REQUIREMENTS EVALUATION:

In this section, we are going to discuss the which requirements mentioned in the requirements analysis chapter were achieved and which requirements we could not implement.

6.1.1 Functional Requirements:

ID	Description	Implemented
FR-1	The system must be able to access and use a smartphone device's camera.	False
FR-2	The system must have user authentication system.	True
FR-2.1	The system must have different authentication systems for practitioners and patients.	True
FR-3	The system must be able to extract data from an image taking through user's smartphone camera.	False
FR-4	The system must be linked to user's personal identification document(s).	True
FR-5	The system should allow the user to send their medical history to a practitioner.	True

6.1.2 Non-Functional Requirements:

ID	Description	Implemented
NFR-1	The system could be optimized to handle querying a large database.	True
NFR-2	The system must be secure and protect personal data.	True
NFR-3	The system should have user management system so individual users can modify their details.	True
NFR-4	The system's GUI must be user friendly.	True
NFR-5	The system must give user the opportunity to review and modify results extracted from the image scanned before uploading them to the database.	False

All the requirements were met except requirements related to OCR. This was mostly due to time constraints to make sure all the other requirements are met. The reason we could not make OCR work with our project was because of the limitations of the libraries found for Flutter. Although there are several OCR libraries built for Flutter, we could not make them work with our project and we are going to discuss the ones we tried and why we could not implement them successfully.

Flutter_mobile_vision:

This library was the only one close to working with our project, but there were several limitations that we could not overcome to successfully make it work. The main one being that it requires the user to extract a single piece of text from a picture. So, if we have a medical record which usually have a lot of text, it can only extract one word or one line of text. Another limitation with this library was the fact that it tries to extract text before taking the picture, so it has trouble working with a lot of text at once. What we wanted was something that can take a picture and then extract results from it, which would have been much more reliable.

Tesseract_OCR:

While this seemed like an optimal library that would work with our project, as it scans the text from an image rather than try to extract text from live camera feed like the previous mentioned library, it had the same limitations as the flutter_mobile_vision. It could not extract large amounts of text optimally nor it could recognize text in smaller sample of image text.

Scanbot_SDK:

This library outright did not work for us because of multiple conflicts with other libraries in our project. It had conflicts with Firestore and Firebase libraries and we could not fix those issues in time to try out this library.

6.2 SYSTEM COMPATIBILITY TESTING:

To test the system compatibility and see if it can be run on multiple devices without any problems, we made use of AVD (Android Virtual Device) as well as two physical devices on hand. For the physical devices, the system was tested on Samsung Galaxy Note 9 running Android version 10 with device resolution of 2220 x 1080. The second physical device we tested the system on was One Plus 8T running Android version 11 with device resolution of 1080 x 2400. The application was built and side-loaded onto these devices using the IntelliJ IDEA which was our choice of IDE for this project's development. We did not find any widgets or elements overflowing the device screen in our physical device testing and all the elements were rendered as we intended them to.

For virtual device testing, we created a few different virtual devices in an emulator using AVD. Different devices were selected for testing based on their android version and screen size to make sure the application runs smoothly on both big and small screen size and is compatible with older version of android devices.

Device Name	Android Version	Screen Resolution
Google Pixel XL (Device used for development testing)	10.0	1440 x 2960
Google Nexus One	10.0	480 x 800
Google Pixel 4 XL	11.0	1440 x 3040
Google Pixel 3	9.0	1080 x 2160
Google Pixel 2	8.0	1080 x 1920

6.3 DATABASE SECURITY TESTING:

To ensure that the rules we have specified in our Firestore database work as intended, we have created unit testing for them using the Firebase emulator running with node.js. In the emulator, we are running Firebase auth and Firestore instance as our rules rely on users being logged in, for which we need the Firebase authentication protocol. The test files are located in the “test” directory in the project’s root folder which is provided with the Github repository. The rules we have tested are as follows:

1. **Only logged in users can read database:** This is essential because we want the practitioner account users to be able to read data of normal users.
2. **Only logged in users can write to database:** This rule is to ensure that no unauthorized writes happen to the database.
3. **Only logged in users can update their *own* entries:** We do not want users to be able to edit each other’s entries. So even in the unlikely case of a user getting access to our database, they will not be able to write anything to someone else’s data as authentication is handled by Firebase, not Firestore.

4. **No user can delete another user's data:** Same reason as the previous point. To ensure the user data is protected and is only managed by them.

```
> tests@1.0.0 test E:\Uni\Year 4\Semester 2\Dissertation\ghms\test\firestore\tests
> mocha --exit

GHMS
  ✓ Logged in users can read database (102ms)
  ✓ Anonymous users cannot read the database
  ✓ Logged in users can write to database (41ms)
  ✓ Anonymous users cannot write to the database
  ✓ Logged in users can update their data in database
  ✓ A user cannot update another user's data
  ✓ A user cannot delete another user's data

7 passing (283ms)
```

Fig.11: Unit testing results for the Firestore ruleset.

6.4 UI TESTING:

The UI testing was already done when we tested the application on different devices as mentioned in section 6.2, as the main concern for UI was the widgets overflowing out of the screen and making the application crash. In cases where the screen would normally overflow, we have used SingleScrollWidget to make that screen scrollable to avoid widget overflow exceptions which would crash the application. For input fields testing, we have relied on the Form widget's validator function which automatically sanitizes the user inputs and if there are any errors in the input field, it displays an error on screen for user instead of proceeding. For example, if user inputs random string in an e-mail input field, the system will show an error next to the e-mail field stating: "invalid e-mail address". This behavior is true for every other input field and all of them throw different error message depending on the input field's type and the error message defined by us.

7 CONCLUSION:

7.1 PROJECT ACHIEVEMENTS:

We successfully managed to create a working prototype of the application and managed to meet almost all the requirements we set out to do in the beginning of the project. It was our first time working with a NoSQL database, but we managed to implement a secure database architecture as it has been made evident section 6.3: Database Testing. Even after a few limitations that we faced, the project goal of creating a database where user can store and share their records, and health care practitioners can access them, was met. Our goal was to create a platform that is not limited to one health care facility and that is what we managed to accomplished in this project.

7.2 LIMITATIONS:

There were two big limitations that we encountered:

1. **OCR:** We failed to meet two functional requirements and one non-functional requirement because we could not implement OCR due to incompatibility of the libraries with our application. Because of that we had to make the system register entries in database using user input.
2. **iOS incompatibility:** Since we did not have access to any iOS device or Mac operating system, we could not test our code on iOS devices. It should work on devices running iOS such as iPhones and iPads if compiled from Mac OS as mentioned previously in Flutter introduction, but since we could not perform or compile for it, we will assume that it does not work for iOS. The limitation comes from Swift being unable to run on Windows machines, and we could not make a virtual desktop running MacOS.

7.3 FUTURE WORK AND DEVELOPMENT:

The project could be further improved by making it easy and efficient to use by adding image recognition tools to extract data from a medical prescription. This will make it easier for the user to record their medical data in the system as it might take too much time to depending on the prescription size to manually enter data in the input fields. The database can also be secured even more by implementing stricter rule set and manually indexing the documents and collections. The UI can be further improved by adding more widgets and customizing the overall layout scheme of the application.

8 REFERENCES

- [1] K. H. Benedicte Iversen Scheel, "Symptoms, signs, and tests," US National Library of Medicine, 07/2015.
- [2] D. H. A. (DHA), "Guidelines For Managing Health Records," Government of Dubai, Dubai, 2019.
- [3] G. D. Patricia Ward, Database Management Systems, 1st ed., Int. Cengage Business Press, 2006.
- [4] J. A. Evans, "Electronic medical records system". Canada Patent US5924074A, 1996.
- [5] A. Gasch, *Anatomy of an EMR company's failure*, NJ: reliasmedia, 1999.
- [6] M. K. a. E. K. Elliot Segal, "System and method for managing patient medical records (Patient Power)". USA Patent US20010041991A1, 2001.
- [7] D. C. S. B. M. P. B. a. B. N. D. Nareesa A. Mohammed-Rajput, "OpenMRS, A Global Medical Records System Collaborative," Roudebush VA Medical Center, Indiana, 2004.
- [8] J. K. R. T. J. H. A. M. S. P. G. B. e. William M Tierney, "The AMPATH medical record system: creating, implementing, and sustaining an electronic medical record system to support HIV/AIDS care in western Kenya .," Stud Health Technol Inform, 2007.
- [9] B. P. W. B. F. H. J. D. A. C. e. a. Mamlin BW, "Cooking Up An Open Source EMR For Developing Countries: OpenMRS – A Recipe For Successful Collaboration," AMIA Annu Symp Proc, 2006.
- [10] A. Kenya, "AMPATH," [Online]. Available: <https://www.ampathkenya.org/our-results>. [Accessed 05 12 2020].
- [11] G. L. a. L. J. Frey, "Efficient Execution Methods of Pivoting for Bulk Extraction of Entity-Attribute-Value-Modeled Data," National Institute of Health, Salt Lake City, 2017.
- [12] T. Alten, "Personal Medical Database Device". Idaho Patent US20020128865A1, 2001.
- [13] N. H. S. -. D. (NHS), "Summary Care Records (SCR)," 2018. [Online]. Available: <https://digital.nhs.uk/services/summary-care-records-scr>. [Accessed 15 04 2021].
- [14] Amazon, "What is Amazon Comprehend Medical?," 2021. [Online]. Available: <https://docs.aws.amazon.com/comprehend/latest/dg/comprehend-med.html>. [Accessed 15 04 2021].
- [15] G. Developers, "Flutter / AngularDart – Code sharing, better together (DartConf 2018)," Google, 26 January 2018. [Online]. Available: <https://www.youtube.com/watch?v=PLHln7wHgPE>. [Accessed 17 04 2021].
- [16] Firebase, "Cloud Firestore Data Modeling (Google I/O'19)," Firebase, 09 05 2019. [Online]. Available: <https://www.youtube.com/watch?v=IW7DWV2jST0&t=1165s>. [Accessed 18 04 2021].