A CROSS PLATFORM MOBILE APPLICATION: HEALTH BLOOM

A Thesis

Submitted in fulfillment of the

Requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING
By

SYED SHIREEN - 17191A0553
N. DIVYA SREE - 17191A0541
K. JAYA PRAKASH - 17191A0532

Under the guidance of

T. NIRANJAN BABU M.Tech

Assistant Professor (Adhoc)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

COLLEGE OF ENGINEERING

(*Autonomous*)

PULIVENDULA-516 390

ANDHRA PRADESH-INDIA

2021

A CROSS PLATFORM MOBILE APPLICATION:

HEALTH BLOOM

A Thesis

Submitted in fulfillment of the

Requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING
By

SYED SHIREEN - 17191A0553
N. DIVYA SREE - 17191A0541
K. JAYA PRAKASH - 17191A0532

Under the guidance of

T. NIRANJAN BABU M. Tech

Assistant Professor Adhoc

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
COLLEGE OF ENGINEERING
(*Autonomous*)
PULIVENDULA-516 390
ANDHRA PRADESH-INDIA

2021

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

COLLEGE OF ENGINEERING (AUTONOMOUS) PULIVENDULA-516 390

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

STUDENT DECLARATION

We hereby declare that this submission is our own work and that to the best of our knowledge and belief, it contains no material previously published or written by another person or material which has been accepted for the award of any degree or diploma of any university or institute of higher learning.

S.Shireen
(17191A0553)

N. Divya Sree
(17191A0541)

K.Jaya Prakash
(17191A0532)

CERTIFICATE

This is to certify that the project entitled "A **CROSS PLATFORM MOBILE APPLICATION: HEALTH BLOOM**" is being submitted by

SYED SHIREEN - 17191A0553
N. DIVYA SREE - 17191A0541
K. JAYA PRAKASH - 17191A0532

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer science & engineering to the **Jawaharlal Nehru Technological University, Anantapur college of Engineering (Autonomous) Pulivendula**, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

T. Niranjan Babu M. Tech
Assistant Professor (Adhoc)

Head of the Department
Dr. S. Jessica Saritha M. Tech, Ph.D.
Assistant Professor & Head
Department of CSE
JNTUACE, Pulivendula

Signature of the External Examiner

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I now have the opportunity to express my guidance for all of them.

I wish to thank **prof.G.Ranga Janardhana,** Honourable Vice Chancellor, JNTUA. His motivation words that created burning desire in us and that eventually led to successfully completing this project work.

The man who has helped us a lot in times of trouble, and who helped us in recovering from intricate problems is Prof**. G. Shankara Sekhar Raju,** principal, JNTUA College of Engineering, Pulivendula. I am thankful to him.

It's my pleasure to say thanks to Prof.**G.V. Subba Reddy**, Vice Principal, JNTUA College of Engineering, Pulivendula for motivating us to have vision and persistence to work in spite of many obstacles.

Foremost, we gratefully acknowledge my sincere gratitude to our respected **Dr. S. Jessica Saritha** M.Tech, Ph.D, Assistant Professor, Head of the Department, Department of Computer Science and Engineering, JNTUA College of Engineering, Pulivendula for his valuable guidance, timely suggestions, and lively intensity throughout the work. We are always thankful to her for providing me the opportunity to do work.

The real mentor and motivator of this project **T. Niranjan Babu** M. Tech**,** Assistant Professor Adhoc of Computer Science and Engineering Department, JNTUA College of Engineering, Pulivendula. His wide knowledge and logical way of thinking have made a deep impression on me. His understanding, encouragement and personal guidance have provided the basis for this thesis. His source of inspiration for innovative ideas and his kind support is well to all his students and colleagues.

Last but far from least, I also thank my family members, my friends, staff and faculty members of CSE Dept for their moral support and constant encouragement. I am very much thankful to one and all that helped me for the successful completion of the project.

With gratitude

SYED SHIREEN - 17191A0553
N. DIVYA SREE - 17191A0541
K. JAYA PRAKASH - 17191A0532

Abstract

The demand for doctors in our country is high and ever growing and this fact is crystal clear when one sees the statistics of 13 lakh doctors for 136 crores of population. And this gap is nowhere near to be filled with 1 lakh doctors being produced every year. This scarcity of doctors has increased apparently because of the covid-19 pandemic. whereas treating patients also by maintaining social distance has become the primary concern. The best possible solution seems to be virtual consultations. So, our main aim is to make it possible by developing a mobile app for patients to get treated by the certified physicians right from where they are. This may sound cool, but it's not always as efficient as meeting the doctor in person. So, our app would be containing main features of a virtual meet with the doctor and also making appointments for consultation and reminders of the appointment and also reminders for prescribed timings of medicine. Also, the keeping track of the health of an individual provides more insights for both the patient and the physician. Locating a nearby hospital and also getting the no. of vacant beds is the prime feature and it exploits the google API to make it possible. In case of accidents or emergencies, close associates, of the person can be alerted with messages that notify urgency and show the location of the user to them.

List of Contents

List of figures

Chapter 1

Introduction

There are things that go wrong if not under constant care and there are some other things that don't need chronical care but once gone wrong there should be appropriate steps taken to become normal. Health is such an issue. It can go wrong on many levels and also there are corresponding treatments. Ample of time is wasted in the process by just waiting to meet the doctor. And for a doctor some small and regular illnesses like cough, headache, cold etc. can be prescribed just by talking to the patient. A patient with these kind of illnesses waiting at a hospital for such long time is really demotivating and make them feel "ain't there a better way". when prescribed with a course of drugs intake, there are pretty good chances to forget them at any point of time because of the busy lifestyles of people now a days. When in a city or a new place getting to know where the nearby hospital is present is also an important task as generally people are prone to wander in the wrong streets even when there are some kind individuals present to direct, which cannot be guaranteed always. Stating that, the above situations are proven to be more expensive in terms of time and remembering things could be more difficult in the middle of the hectic schedules. With the advancements of telecommunication and with the changed ways of interaction the aforementioned issues can be handled better by implementing an app. people now a days with all those hectic schedules to keep up with and living under constant stress are leading to a lot more medical issues and illnesses than we are supposed to. With increased problems and not so increase in the number of physicians to treat patients, the queues in the clinics and the waiting time for a troubled individual to get treated are increasing. Although visiting a doctor when feeling ill is advisable but with lots of important things at stake the idle time that's Infront of it is not recommendable. After examining these issues virtual meet with the doctor is more appealing that saves a person from both tiring commute to hospital and waiting there.

1.1 Motivation and Scope:

Do you know…???

- The global medical app market has tremendous growth potential expected to generate a revenue of around USD 111.1 billion. Adoption of modern technology in the form of healthcare apps is the way to go. It's truly a win-win for both healthcare providers and patients.
- An important factor to consider is that the digitization of healthcare and technology can never completely replace the human touch, but there are many operations that can make significant changes in a person's health and life.
- The main reason for creating the app is to help people by intimating the availability of beds in the nearby hospitals so that they come to know where they have to go for efficient treatment in difficult situations.
- The other reason is, this app shows the growth of health by the feature "Health-Tracker" added in the app. And also reminding them to take the pills at a particular time under the doctor's prescription.

1.2 Need for Study:

So, creating an app for remotely contacting doctors is the next booming idea that has already proven its worth. And we are doing so with the flutter framework powered by dart language and exploiting the available open-source APIs and make it possible for a person to contact remotely and securely an authorized reliable individual.

What's even more fascinating is that even healthcare professionals are using smartphone health applications. Doctors can now remotely access patient's health data, monitor their progress, and provide real-time guidance.

1.3 Literature Survey:

To conduct this literature survey, we followed the guidelines used by previous literature (Najaftorkaman et al., 2013), including three main discrete activities: (i)searching the initial list of papers; (ii)appraising relevant papers; (iii) extracting and analyzing data.

Initial Search:

We identified initial search resources informed through prior work (Najaftorkaman al., 2013). We considered that the relevant papers might distribute in the areas of research in Information and Computing Sciences, Biomedical Engineering, and Medicine and Health Sciences. Hence, we selected a set of journals ranked in CORE1(CORE, 2016b) referring to these three domains, and we only screened journals ranked A*, A, or B in CORE in this survey.

Accordingly, the studies from these journals are more reliable, and the research quality is assessed through CORE ranking and impact factors.

Firstly, we found 8 journals from the journal list of Information and Computing Sciences (FoR code2 08) overlapping bioinformatics, medical informatics, medicine and health sciences (UNSW, 2013). Secondly, we screened 7 journals from the journal list of Biomedical Engineering (For code 0903) covering information and computer sciences, and 3 of them were duplicate with those found in the first round. Thirdly, in the journal list concerning Medicine and Health Sciences (FoR code 11), we only collected 1 journal in the list involving information systems, but this journal was already included. Eventually, 12 journals remained as our initial search resources. Furthermore, we measured the latest impact factor of each journal from Journal Citation Reports (Thomson Reuters, 2016), and most impact factors are greater than 1.000. In addition, we checked the Google citation of each selected journal (Google Scholar, 2017).

These journals are indexed in different databases such as IEEE Xplore, ScienceDirect and Scopus.

Table 1 - List of journals selected in this review							
No.	**Title**	**Ranking in CORE**	**Impact factor**	**h5-index**	**Database**	**Identified papers**	**Relevant papers**
1	IEEE Transactions on Information Technology in Biomedicine	A*	2.493	46	IEEE Xplore (IEEE, 2016)	2	1
2	Computer Methods and Programs in Biomedicine	A*	1.862	39	ScienceDirect (Elsevier, 2016a)	15	0
3	International Journal of Medical Informatics	A	2.363	43	ScienceDirect (Elsevier, 2016a)	43	12
4	BMC Bioinformatics	A	2.435	66	Scopus (Elsevier, 2016b)	0	0
5	Artificial Intelligence in Medicine	A	2.142	28	ScienceDirect (Elsevier, 2016a)	2	0
6	Computer Methods in Biomechanics and Biomedical Engineering	A	1.850	27	Scopus (Elsevier, 2016b)	0	0
7	Medical and Biological Engineering and Computing	A	1.018	-	Scopus (Elsevier, 2016b)	0	0
8	BMC Medical Informatics and Decision Making	B	2.042	33	Scopus (Elsevier, 2016b)	10	7
9	Computers in Biology and Medicine	B	1.521	32	ScienceDirect (Elsevier, 2016a)	7	1
10	Journal of the American Medical Informatics Association	B	3.428	61	Scopus (Elsevier, 2016b)	8	1
11	International Journal of Bioinformatics Research and Applications	B	-	8	Scopus (Elsevier, 2016b)	0	0
12	Journal of Biomedical Informatics	B	2.447	44	Scopus (Elsevier, 2016b)	12	2
	Total number of papers					99	24

Fig 1.3: List of journals selected in this review

1.4 Research Gap:

After observing many health care apps that are available in the AppStore and play store, which have ways of facilitating appointments, reminders, check-ups, supervision on the go, offer a host of other benefits, and video calls, medicine delivery at home, etc.

We realized that all those apps lack a premier function which is fetching the availability of beds in the nearby hospitals, which could be a lifesaver in severe conditions, .so we intend to implement it along with some of the aforementioned features.

This feature would be more efficient when coupled with routing and navigation to that nearby hospital, which is an addon in fulfilling the gap.

Chapter 2

2.1 Problem Statement:

People nowadays with all those hectic schedules to keep up with and living under constant stress are leading to a lot more medical issues and illnesses than we are supposed to. With increased problems and not so increase in the number of physicians to treat patients, the queues in the clinics and the waiting time for a troubled individual to get treated are increasing.

The personal health monitoring of each individual is considered very important because of the rise in health problems in today's world. The increasing stressful lifestyle is taking a maximum toll on public health.

Majority of the minor issues can be solved by just following a good diet, proper sleep patterns, and regular exercising. But how does a patient know what diet is good or what exercise he/she should follow and more importantly whether the plan that he is following is working effectively for him? The absence of such a mechanism makes the task of the patient a bit difficult. Thus, introducing them to this virtual meeting with the doctors in this mobile health care apps.

Although visiting a doctor when feeling ill is advisable but with lots of important things at stake, the idle time that's in front of it is not recommendable. After examining these issues, a virtual meet with the doctor is more appealing that saves a person from both a tiring commute to the hospital and waiting there.

We are implementing a project of creating an app that acts like a interface between doctors and patients, which helps a person in getting necessary advice from a doctor right from where they are and if not sufficient making appointment with doctor. Also getting the information of the nearby hospitals and the number of beds vacant in them is the prime feature which has proven to be more important information in this pandemic. Also, to alert the close associates is also a concern that is going to be addressed in this project.

2.2 Objectives:

The objectives of our project are

- Doctor Appointments:
  - To make appointments with an authenticated and authorized health professional.
- Appointment Remainder:
  - A reminder that prompts the appointments made.
- Medicine Reminder :
  - A reminder that prompts the taking the medicines prescribed the doctor at prescribed time.
- Locate Nearby Hospitals :
  - Searching for a nearby hospitals and routing and navigation to it, along with the number of vacancies at those hospitals.
- Alert Emergency contacts with location:
  - Sending notifications to the close associatives in times of emergency.

- Emergency Video Calls:
  - Video chat feature to converse and get instructed or explain the condition in emergency times.
- Health Tracker :
  - A graph that tracks the important health vitals like glucose level, blood pressure etc for analysis purposes.
- Notes :
  - An inbuilt scribble pad to make note of important points.

2.3 Hypothesis:

A user would have to register his credentials and create a profile. All the features would work with the inbuilt APIs irrespective of the user's data. Features that would depend on user details are the health trackers and notes, Emergency contacts are alerted given the emergency of the situation and tapping alert contacts.

Chapter 3

3.1 Research Methodology:

After identification of the relevant papers, we designed a form to extract data from included studies based on our research questions. There are three criteria we applied in our form for data extraction. These criteria were derived from our three RQs. **Healthcare records** – We classified the forms of healthcare records in the included studies into three groups: electronic records, paper-based records, and hybrid records. We assigned each paper to the corresponding group after identification of the form of healthcare records. **Definition and measure** – We summarized several examples from those papers that explicitly provided definitions and measures of data completeness in their studies. In this way, we could address our fundamental questions about definition and measure of data completeness raised in the section of Introduction. **Study themes** – In order to identify the study themes per paper, we emphasized the objectives and content of the study to determine the theme. If the objective of one study is to design or develop tools or methods to address data completeness, this kind of study could be categorized into 'design and development' theme. If the studies conducted an assessment of data completeness, they could be invited into the theme of 'evaluation'. While other studies investigated the challenges or barriers to achieving data completeness in healthcare, they could be classified into 'determinants' theme, since these difficulties encountered by practitioners in achieving data completeness also can be seen as factors impacting the achievement of completeness. Some paper may have multiple objectives and therefore, such research could address two or more than two study themes. We reviewed each paper thoroughly keeping the three main criteria in focus. We applied thematic analysis and frequency analysis on the text of each paper to extract and record explicit response from publications for each criterion (healthcare records, definition and measure, and study themes). After finishing the data extraction, the first author labelled the dataset content with explicitly mentioned terminology in the literature and grouped them into several categories based on the criterion in the first round of classification. In the second and third rounds, we revised, merged or split some of the categories. In order to resolve discrepancies and achieve the final classification, we acted in two groups: literature survey and assessment. In this manner, the extracted data was assessed and re-examined against the literature for the validity of classification.

Teleconsultation is made possible with the doctor and based on necessity an appointment is made with the doctor. For scheduled appointments a reminder is present to prompt the appropriate timings. After consultation with the doctor a medicine reminder is implemented to notify the timings for prescribed medicine. In case of emergency or at extreme situations friends and family can be alerted with current location. And locating and navigating to the nearest hospitals is easier. One can make videocalls either to a doctor or friends in times of emergency and get instructions of first aid or the primitive steps. Track the progress of the important health vitals like blood pressure, glucose levels, body weight etc. and analyze the fluctuations from time to time. Make note of important points in the app itself and also prioritize them according to the necessity.

3.2 Modules:

3.2.1 Doctor Appointments:

Provide more accessibility to the patients. Focuses on helping patients to easily find the doctor. This includes patients an option to 'Book Anytime, anywhere' and also allows easy access to doctors. Furthermore, the appointment booking feature makes the lives of patients, and those who are suffering from life-long ailments or are bed-ridden more comfortable.

Work Flow of a patient:

Sign in - -> Search and select Doctor - -> Book an Appointment - -> Receive Notification - -> Consult with doctor.

Users should be allowed to upload their photos and any other information that they want to share.

Patients should be able to choose and select the Doctors according to their search results after the filters. Once they are done shortlisting, they should be able to check the doctor's profile and his specialty, visiting hours an etc.

Work Flow of a Doctor:

Sign in - -> Complete profile - -> Share Timeslot - -> Approve Appointment - -> Communication with patient - -> Prescription.

However, in the case of doctors, all the required information should be available like their specialty, experience, contact details along with their registered practitioner numbers.

The doctors should get authorization to accept or deny consultation requests if, for some reason, only after the doctor confirms or accepts the request, the customer should get notification.

Here, in this feature patients either book an appointment just by mentioning their problem or also there is a facility to add the previous checked reports while booking the appointment, this helps the doctor in a significant way.

3.2.2 Appointment Remainder:

Appointment reminders are sent to patients in the form of notifications before their schedule. It shows the upcoming appointments and also records all the past appointments of the user. It is a very useful feature for the patients who might forget the timing of their appointment with the doctor, in that case, it plays a huge role.

Work Flow:

Click on the Floating button - -> Enter name of the doctor - -> Enter place to visit - -> Enter type of the specialist - -> Set time and Date - -> Click on Add Appointment.

The present added appointment will be displayed in the upcoming events and shows the notification at that particular time and once it is completed it displays under the past appointments.

3.2.3 Medicine Remainder:

There are plenty of tips and tricks to help remember your medications, but perhaps the most helpful are mobile health care apps that provide daily prescription reminders via notifications. These apps help to automate and track doses, so you're less likely to miss a pill.

This feature reminds the patient to stay on schedule for their daily medicines. The reminder hits an alarm and requests confirmation if the person has taken the specific medicine at the times prescribed. If not responded to, or responded negatively, the app sends a notification.

Work Flow:

Click on the Floating button - -> Enter Medicine Name - -> Enter type of Medicine - ->Select how many times a day - -> Set the time - -> Click save.

3.2.4 Locate Nearby Hospitals:

This is the most important feature of this app. This feature is to help people by locating the nearby hospitals in their current location providing information on the distances and directions for you and intimating the availability of beds in the hospitals so that they come to know where they have to go for efficient treatment in difficult situations.

Work Flow:

Click on the locate Nearby Hospitals - -> Directly Navigate to the Google Maps - -> Shows the Nearby Hospitals with availability of beds.

3.2.5 Contact Relatives:

This feature adds the contacts and if any emergencies, that shares your current location to the contacts if want them to reach you as soon as possible. Helps to access you in an emergency, or to coordinate care.

Work Flow:

Click on Add Relative - -> Edit relative details (Name, Mobile number) - -> Save changes - -> Click on Link Relative.

Whenever clicked on contact relative it directly navigates to the SMS page and sends the message to the contact.

3.2.6 Urgent Video Calls:

Patients use this feature to connect quickly with a medical professional, instead of waiting weeks for an appointment. This feature helps you to contact with the doctors in the time of appointment and also it helps to connect with loved ones on emergency needs. This offer video, phone, and text chat between patients and doctors.

This virtual care uses jisti meet services that are designed to give you immediate access to a doctor so getting medical attention is more convenient and accessible for everybody. This means no more sitting around urgent care waiting rooms full of germs and sick people just to get seen for a minor cold. It also means you don't have to take a big chunk of time out of work or take a sick child down to an office to see a doctor. Instead, you can get on your mobile device and easily have a doctor see you virtually from the comfort of your home or office.

Work Flow:

Click on the Video call - -> Check the required options- -> Start Now - -> Can share the link via many social sources.

You can connect the video by checking the required boxes mentioned such as Audio only, Audio muted, Video muted and once if you are okay with the options then you can start the video. And here you can invite the people by sharing the link.

3.2.7 Health Tracker:

This feature in this health bloom app shows the growth of health by the feature "Health-Tracker". Helps in organizing all personal documents you are challenged to manage - from all personal IDs and journals to medical contacts and insurance information. Secure upload and storage.

This feature helps the patient to enter the information in to a health category, collects the data you mentioned daily and shows the graph of your health status. So, you can view all your progress in one convenient place. It shows your Health over time, so you can see how you're doing overall.

Work Flow:

Open Health tracker - -> Click on Sleep Tracker - - > Give data.

Open Health tracker - -> Click on Weight Tracker - - > Give data.

Open Health tracker - -> Click on Blood Glucose - - > Give data.

Open Health tracker - -> Click on Blood Pressure - - > Give data.

3.2.8 Notes:

This feature helps person to enter all important notes and to record observations in the app itself. Notes will be arranged in the priority specified.

Work Flow:

Click on Floating button - -> Add Title and write the notes.

Chapter 4

Software Components:

4.1. Firebase:

Firebase is a platform developed by google for creating mobile and web applications. It provides backend as a service with variety of tools. It is categorized as NoSQL database program which stores JSON-like documents. It is used for the authentication of the user and access firestore that contain the native SDKs. Data taken from the user is stored in this NoSQL database in hierarchical structure of subcollections. Firebase REST APIs and tools make it simple to work on code and apps simple. Building a website or an app it made easy by using firebase.

In our app it is used in working with the login page and signing up. Also, the doctor authentication while making an appointment

ADDING FIREBASE TO THE APP:

It involves four steps;

Step 1: creating a firebase project:

To add firebase to the flutter app, firstly we have to add create a firebase project and then add it to the app. When implementing the flutter app in both iOS and android then a single firebase project is created and linked to both versions.

Step 2: Registering the app with firebase:

- Launch the workflow in the project overview console page.
- Enter the bundle ID in iOS like com.mycompany.ios-app-name and add appstore ID to get redirected to appstore in apple devices.
- For android devices add android package name which is also, the application ID. It is generally present in the gradle file at app/build.gradle.
- Firebase uses dynamic links for authentication and SHA-1 hash is required for the user authentication when using the google sign-in or phone number.
- Register

Step 3: Add firebase configuration file:

- For android setup, obtain your firebase android configuration file from google-services.json. Move the config file to the android/app directory in flutter app and enable the firebase services for android devices in by adding google services plugin to the gradle file in android/build.gradle
- For apple devices, obtain your firebase configuration file from GoogleServices-Info.plist. Move fille to Runner/runner directory of flutter app.

Step 4: add FlutterFire plugins:

- Flutter uses plugins to provide access to a wide range of platform-specific services, such as Firebase APIs. Plugins include platform-specific code to access services and APIs on each platform.
- Firebase is accessed through a number of different libraries, one for each Firebase product (for example: Realtime Database, Authentication, Analytics, or Cloud Storage). Flutter provides a set of Firebase plugins, which are collectively called FlutterFire.
- Since Flutter is a multi-platform SDK, each FlutterFire plugin is applicable for both iOS and Android. So, if you add any FlutterFire plugin to your Flutter app, it will be used by both the iOS and Android versions of your Firebase app.

We have implemented authentication of the user using the firebase and it is implemented adding user sign-in (RSVP) button

Provider package is used to make a centralized state object available throughout the application's tree of flutter widgets.

The user starts off unauthenticated, the app shows a form requesting the user's email address, depending on whether that email address is on file, the app will either ask the user register, or request their password, and then assuming everything works out, the user is authenticated.

Lib/main.dart file contains the code for implementing the user authentication and it happens in many statelful widgets that act based upon the state which they are present. The following is the list of those used states,

1. email
2. loginState
3. startLoginFlow
4. verifyEmail
5. signInWithEmailAndPassword
6. cancelRegistration
7. registerAccount,
8. appState.signOut

4.2 SQFlite Data Base:

Sqlite is c-library that is implements a fast, small, self-contained, high reliable full featured sql database. It' s not a client-server database engine, rather its embedded in the end program. Hence it is used for the offline storage of the data. To implement it in our project we add sqflite and add the path of the packages to the project. Now creating a db/lib file is enough to get it run along with the code.

Sqflite is a package for SQLite plugin for flutter. It supports iOS and Android and MacOS.

1. It supports the transactions and batches.
2. Automatic version management during open.

3. Helpers for insert/query/update/ delete queries.
4. DB operation executed in a background thread in Android and iOS.

As mentioned above sqflite database is used to store locally. Persistent data is stored local in general because, persisting data is very important for users since it would be inconvenient for them to type their information every time or wait for the network to load the same data again.

SQLite is one of the most popular ways to store data locally. We used the package sqflite to connect with SQLite. Sqflite is one of the most used and up to date packages for connecting to SQLite databases in Flutter.

First of all, we have to Add dependency to the project:

It is done by adding the latest version of sqflite and path_provider under dependencies in the pubsedc.yaml file. We use the path_provider package to get the commonly used location such as TemporaryDirectory and ApplicationDocumentsDirectory.

Creating a DB client:

We create a new database.dart file which is singleton. We use the singleton pattern to ensure that we have only one class instance and provide a global point access to it. Now we setup a database, if there is no object assigned to the database, we use the initDB function to create the database. In this function, we will get the path for storing the database and create the desired tables.

Opening database:

An SQLite database is a file in the file system identified by a path. If relative, this path is relative to the path obtained by getDatabadePath(), which is the default database directory on Android and the documents directory on IOS.

Creating the model class:

The data inside the database is first converted inti Dart maps. So, we need to create the model classes with tomap and frommap methods.

CRUD operations stand for the create, read, update, delete:

The SQFlite package provides two ways to handle these operations using RawSQL queries or by using a table name and a map that contains the data. There is a basic migration mechanism to handle schema changes during the opening. Many applications use one database and would never need to close it although it will be closed when the application is terminated. If you want to release resources, you can close the database.

Results of maps are read-only. you need to create a new map if you want to modify it in the memory. For a transaction don't use the database but instead only use the Transaction object in a transaction to access the database. A transaction is committed if the call-back does not throw an error. If an error is thrown, the transaction is cancelled. So, to rollback a transaction one way

is to throw an exception. To avoid the changing from dart to native and again native to dart for each transaction we used batch. During a transaction, the batch won't be committed until the transaction is committed.

4.3 Shared Preferences:

Shared preferences allow you to store small amounts of primitive data as key/value pairs in a file on the device. To get a handle to a preference file, and to read, write, and manage preference data, use the SharedPreferences class. Shared preferences can be read privately or publicly. Changes in the memory object are written and updated asynchronously. Data could also be written synchronously but when using this calling it from the main thread would stop the UI rendering so we only used asynchronously updating disks.
Wraps platform-specific persistent storage for simple data (NSUserDefaults on iOS and macOS, SharedPreferences on Android, etc.). Data may be persisted to disk asynchronously. Since data is added Asynchronously there is no guarantee that writes will be persisted to disk after returning, so this plugin must not be used for storing critical data.
The plugin could be added by adding the shared preference line to pubspec.yml in the dependencies section.

```
dependencies:
flutter:
sdk: flutter
shared_preferences: '>=0.5.12+4 <2.0.0'
```

In the file that we intend to use shared preferences we import the them by using

```
import 'package:shared_preferences/shared_preferences.dart';
```

Reading and writing data:

To get the shared preferences object you can do the following:

```
final prefs = await SharedPreferences.getInstance();
```

This will be used for all of the following examples.

1. int

```
// read
final myInt = prefs.getInt('my_int_key') ?? 0;// write
prefs.setInt('my_int_key', 42);
```

2. double

```
// read
final myDouble = prefs.getDouble('my_double_key') ?? 0.0;// write
prefs.setBool('my_bool_key', true);
```

3. bool

```
// read
final myBool = prefs.getBool('my_bool_key') ?? false;// write
prefs.setBool('my_bool_key', true);
```

4. string

```
// read
final myString = prefs.getString('my_string_key') ?? ';// write
prefs.setString('my_string_key', 'hello');
```

5. stringList

```
// read
final myStringList = prefs.getStringList('my_string_list_key') ?? [];// write
prefs.setStringList('my_string_list_key', ['horse', 'cow', 'sheep']);
```

apart from using the primitive data types one can also implement the complex data types, even though this is not a default property one can manipulate and manage to get this functionality by first converting them into strings and then assigning them. First you convert the Dart object to a map. Then you convert the map to a JSON string using jsonEncode. Finally, you can save the JSON string to shared preferences as you would any other string.

```
Eg:// import 'dart:convert';Person person = Person('Mary', 30);Map<String, dynamic> map = {
 'name': person.name,
 'age': person.age
};String rawJson = jsonEncode(map);
prefs.setString('my_string_key', rawJson);
```

Getting a custom Dart object back from shared preferences is just the same but in the other direction.

```
final rawJson = prefs.getString('my_string_key') ?? ';
Map<String, dynamic> map = jsonDecode(rawJson);
final person = Person(map['name', map['age']);
```

4.4. JISTI MEET:

Jisti is a collection of free and open-source multiplatform voice, video conferencing and instant messaging applications for the web platform, Windows, Linux, macOS, iOS and Android. The Jitsi project began with the Jitsi Desktop.

This is used to implement the urgent video call feature. Meets can be created with by customizing like audio only or both audio and video muted etc. The link can be sent to appropriate persons who are to be present at the meet. It can also be launched in web browser

in a device when local app isn’t present and mostly used by international professionals. Jisti videobridge was the implementation of the multi -party video calling for the first time.

Jisti Meet is an open-source JavaScript WebRTC application used primarily for video conferencing. In addition to audio and video, screen sharing is available, and new members can be invited via a generated link. The interface is accessible via web browser or with a mobile app. The Jitsi Meet server software can be downloaded and installed on Linux-based computers. Jitsi owner 8x8 maintains a free public-use server for up to 50 participants at meet.jit.si.

Key features of Jisti Meet:

Encrypted communication (secure communication):

As of April 2020, 1-1 calls use the P2P mode, which is end-to-end encrypted via DTLS-SRTP between the two participants. Group calls also use DTLS-SRTP encryption, but rely on the Jitsi Videobridge (JVB) as video router, where packets are decrypted temporarily. The Jitsi team emphasizes that "they are never stored to any persistent storage and only live-in memory while being routed to other participants in the meeting", and that this measure is necessary due to current limitations of the underlying WebRTC technology.

Features:

Jitsi supports multiple operating systems, including Windows as well as Unix-like systems such as Linux, Mac OS X and BSD. The mobile apps can be downloaded on the App Store for iOS and on the Google Play Store and F-droid platform for Android. It also includes:

1. Attended and blind call transfer
2. Auto away
3. Auto re-connect
4. Auto answer and Auto Forward
5. Call recording
6. Call encryption with SRTP and ZRTP
7. Conference calls
8. Direct media connection establishment with the ICE protocol
9. Desktop Streaming
10. Encrypted password storage using a master password
11. File transfer for XMPP, AIM/ICQ, Windows Live Messenger, YIM
12. Instant messaging encryption with OTR (end-to-end encrypted)

13. IPv6 support for SIP and XMPP

14. Media relaying with the TURN protocol

4.5. Tomtom API:

Tomtom API contains cross platform REST APIs that are useful to locate the nearby hospitals. Searching, routing, mapping, traffic and navigation to those hospitals is also done by exploiting this API. It is used in our app to searching. search for an address, business or place. Geocode to validate and structure addresses globally.

Tomtom consists of a suite of APIs developed and maintained by the tomtom community for rendering the maps, search, routing, mapping, traffic and navigation.

4.5.1 Map API:

The Amp Display API web services suite offers the following APIs with their endpoints:
It again has two sub-APIs:

1.Raster API: The Maps Raster API renders map data that is divided into gridded sections called tiles. Tiles are square images (png or jpg format) in various sizes which are available at 23 different zoom levels, ranging from 0 to 22. For zoom level 0, the entire earth is displayed on one single tile, while at zoom level 22, the world is divided into 2^{44} tiles.

2.Vector API: Similar to the Maps Raster API, the Maps Vector API serves data on different zoom level ranging from 0 to 22. For zoom level 0, the entire earth is displayed on one single tile, while at zoom level 22, the world is divided into 2^{44} tiles. The Maps Vector Service delivers geographic map data packaged in a vector representation of squared sections called vector tiles. Each tile includes pre-defined collections of map features (points, lines, road shapes, water polygons, building footprints, etc.) delivered in one of the specified vector formats. The format of the tile is formally described using the protobuf schema.

4.5.2 Search API:

The Search API consists of the following services with their corresponding endpoints:

Search: This is a fuzzy single-line search process for query-based relevant addresses and POIs, with the option to restrict the search within a certain geometry.

Autocomplete: Autocomplete enables you to make a more meaningful Search API call by recognizing entities inside an input query and offering them as query terms.

Geocoding: Geocoding only looks up locations of addresses with the ability to provide single line or structured addresses as input.

Reverse Geocoding: Reverse Geocoding turns latitudes / longitudes into human readable addresses with the ability to look up cross streets.

Filters: Filter your own or 3rd party POIs (Points of Interest) within a certain geometry.

Additional Data: Returns sets of coordinates that represent the outline of a city, country, or land area.

POI Categories: Provides a full list of POI categories and sub-categories, together with their translations and synonyms.

Batch Search: Batch Search is a subset of the Search API and dispatches batches of requests to endpoints from the Search API suite with ease. You can call Batch Search endpoints to run either Asynchronously, or Synchronously.

Search API Market Coverage: Provides a list of all countries supported by the Search API engine.

Extended Search API: The Extended Search API consists of the following endpoints:

EV Charging Stations Availability: The EV Charging Stations Availability endpoint provides information about the current availability of charging spots.

Points of Interest Details: The Points of Interest Details endpoint returns rich content of the POI, which includes ratings, price range, reviews, and photo IDs of the POI.

Points of Interest Photos: The Points of Interest Photos endpoint returns (at this point up to 5) photos of the POI. This API uses the photo IDs provided in the Points of Interest Details endpoint.

Extended Search API Market Coverage: Provides a list of all countries supported by the Extended Search API engine.

4.5.3. Routing API:

TomTom Routing is a suite of web services designed for developers to use our latest scalable routing engine. Independent tests have established that the TomTom routing engine is the best in the industry. Our routing engine uses IQ Routes and TomTom Traffic. It consists of two web services calculating the route and calculating the range. Routing can be of two types

Batch routing: Batch Routing works as a subset of the Routing API suite. This service enables you to dispatch batches of Calculate Route and Calculate Reachable Range Requests with ease.

Asynchronous and Synchronous batch processing: You can call Batch Routing APIs to run either asynchronously or synchronously.

Asynchronous API overview: The Asynchronous API is appropriate for processing big volumes of relatively complex routing requests. It allows the retrieval of results in a separate call (multiple downloads are possible). The Asynchronous API is optimized for reliability and is not expected to run into a timeout. The number of batch items is limited to 700 for this API.

Data retention period notice please be aware that batches processed by an Asynchronous API are available for download for 14 days, after which a results download will return a HTTP 404 (Not Found) Response.

Sequence of Asynchronous API client actions: A client sends a Request to the Asynchronous Batch Submission endpoint. The server will respond with one of the following:

HTTP 202 or HTTP 303: This depends on the redirectMode parameter with the Location header, which points to the Asynchronous Batch Download endpoint.

Another HTTP error: See the Asynchronous Batch Submission HTTP status codes. After getting a HTTP 202 or HTTP 303 Response, the client should follow the redirect to the Asynchronous Batch Download endpoint which is a blocking long poll Request. When a client calls the Asynchronous Batch Download endpoint the possible scenarios are:

Batch Response is calculated before timeout (by default this is 120 seconds; it can be changed by using the waitTimeSeconds parameter). The client receives HTTP 200. Batch Response is ready and it gets streamed to the client. Batch response is not ready before timeout. The client receives HTTP 202. Batch Request is accepted for processing. The client downloads batch results from the URL specified by the Location header (see point 3).

Synchronous API overview:

The Synchronous API is recommended for lightweight routing requests with quickly-expiring results (e.g., due to changing traffic conditions).

In such a case, when the service receives a Request, it will respond as soon as the routes of the batch are calculated and there will be no possibility to retrieve the results later on. The Synchronous API will return a timeout error if the request takes longer than 60 seconds. The number of batch items is limited to 100 for this API. Sequence of Synchronous API client actions. A client sends a Request to the Synchronous Batch endpoint.

The server will respond with one of the following:

1) **A HTTP 200 Batch processing result:** The calculation is finished before timeout and the client downloads results straight away.

2) **A HTTP 408 Request timeout error:** This occurs if the Request takes longer than 60 seconds and cannot be finished in this timeframe.

3) **Another HTTP error:** See the Synchronous Batch Response codes.

Matrix routing:

The Matrix Routing service enables the calculation of a matrix of route summaries for a set of routes defined with origin and destination locations. For every given origin, this service calculates the cost of routing from that origin to every given destination. The set of origins and the set of destinations can be thought of as the column and row headers of a table, while each cell in the table contains the costs of routing from the origin to the destination for that cell.

The following costs are computed for each route:

1) Travel times
2) Distances

Use the computed costs to determine which routes to calculate using the Routing API suite.

THE ASYNCHRONOUS API: The Asynchronous API is appropriate for processing big volumes of relatively complex routing requests. It allows the retrieval of results in a separate call (multiple downloads are possible). The Asynchronous API is optimized for reliability and is not expected to run into a timeout. The maximum size of a matrix for this API is 700 (the number of origins multiplied by the number of destinations), so examples of matrix dimensions are: 5x10, 10x10, 28x25 (it does not need to be square). Data retention period notice Please be aware that matrices processed by the Asynchronous API are available for download for 14 days, after which a Request for results download will return a HTTP 404 (Not Found) Response. Sequence of Asynchronous client actions A client sends a Request to the Asynchronous Matrix Submission endpoint.

The server will respond with one of:

HTTP 202 or HTTP 303 (depending on redirectMode parameter) with Location header, which points to the Asynchronous Matrix Download endpoint.

A HTTP error. See the Asynchronous Matrix Submission Response codes. After getting a HTTP 303 Response, the client should follow the redirect to the Asynchronous Matrix Download endpoint which is a blocking long poll Request. The client waits for a matrix Response.

Possible scenarios are:

The matrix Response is calculated before timeout (by default 120 seconds, can be changed by waitTimeSeconds parameter). The client receives HTTP 200. The matrix Response is ready and it is being returned to the client. The matrix Response is not ready before timeout. The client receives HTTP 202. The matrix Request is accepted for processing.
The client downloads the matrix results from the URL specified by the Location header.

Synchronous API overview:

The Synchronous API is recommended for lightweight routing requests with quickly-expiring results (e.g., due to changing traffic conditions). In such a case, when the service receives a Request, it will respond as soon as the route summaries of the matrix are calculated, and there will be no possibility to retrieve the results later on. The Synchronous API will return a timeout error if the Request takes longer than 60 seconds. The maximum size of a matrix is limited to 100 for this API (the number of origins multiplied by the number of destinations), so examples of matrix dimensions are: 5x10, 10x10, 20x5 (it does not need to be square). Sequence of Synchronous Matrix API client actions. A client sends a Request to the Synchronous Matrix endpoint. The server will respond with one of the following:
A HTTP 200 Matrix processing result. The calculation is finished before timeout and the client downloads the results straight away.

A HTTP 408 Request timeout error. If the Request takes longer than 60 seconds and cannot be finished in this timeframe.

Another HTTP error. See the Synchronous Matrix HTTP status codes section.

4.5.4 Traffic API:

The Traffic Incidents service is a suite of web services designed for developers to create web and mobile applications around real-time traffic. These web services can be used via RESTful APIs. The Traffic Incidents APIs are based on real-time traffic data from TomTom Traffic.

Features of the tomtom traffic API are:

1) Is updated every minute with very latest traffic incident and delay information.
2) Returns detailed information about traffic jams and traffic related incidents. Details include: start-location, end-location, road-name, type of delay, length (in time) of the delay, significance, and distance.
3) The Incident Tile API provides traffic incident and delay information for display on your map view.

4.5.5 Opencage API:

Opencage geocoding API is used to send text messages of the latitude and longitude details to the emergency contacts. It works in both directions, that is, when clicked by the receiver it opens into maps and shows the location of the person who tried to notify. Alerting emergency contacts feature exploits this API. The message sent would have the address of the location and when clicked it would open in maps. If maps is not present in the device then it would also open in browser.

The open cage geocoder is an aggregator API. Every query at the API generates calls to the various back-end geocoders have been installed. The bulk of the geocoder code is written in Perl, apart from the back-end geocoders which are written in a whole plethora of languages. It relies on different set of tools and services to operate and maintain the business solutions. The API is served by HAProxy and Apache, and the whole lot runs on top of servers running ubuntu, a version of Linux. The default API response includes the HTTP header access-control-allow-origin: * which thus allows all cross-origin requests. Many people confuse forward geocoding with geosearch. Geocoding expects an address or placename as an input. Geosearch takes any string of text as input and tries to extract locations from that string or to expand the string and return a list of possible location matches. This geocoding API is designed for geocoding, and does NOT perform "fuzzy" matching. As an example, a request with query par will NOT return Paris, France as a result. The auto-suggest or also known as typehead popularly is not yet included in this API. The OpenCage geocoding API will always attempt to find a match for as many parts of a query as it can, but this isn't always possible. Where a partial match is made, for example a street name can be matched but not the specific house number, then we will still return a result.

Geocoding confidence:

The precision of a match is returned in the confidence field. Please note: confidence is NOT used for ranking of the results. It does not tell you which result is more "correct", nor what you type of thing the result is - for that please check the components portion of the result.

The confidence score is an integer value between 0 and 10, where 0 reflects inability to determine a confidence (due to lack of a bounding box), 1 indicates low precision, and 10 indicates high precision.

Confidence is calculated by measuring the distance between the southwest and northeast corners of each result's bounding box. Then an adjustment may be made to reflect the ambiguity of the underlying geocoder. Please note, confidence is not the way to determine the type of place that was matched, for that please use the _type field of the components portion of the response.

Ambiguous Results:

When forward geocoding one may find multiple valid matches for a query. Many places have the same or similar names. In this case, multiple results ranked by relevance are returned.

No Results:

In cases where the geocoder cannot find any match for a request, we will return a successful status (a response code of 200) but the number of results in the response will be zero. You can test this situation by sending a request with the query NOWHERE-INTERESTING which will return a valid response with 0 result.

4.6 Dart:

Dart is an open-source, scalable programming language, with robust libraries and runtimes, for building web, server, and mobile apps. It is a client optimized language for fast apps on any platform. It is optimized for UI around the needs of user interface creation. We used dart native as our app is for mobile devices. It includes dart VM for just–in–time compilation and Ahead time compiler for machine code.

Asynchronous operations let the program complete work while waiting for another operation to finish. Here are some common asynchronous operations:

1) Fetching data over a network.
2) Writing to a database.
3) Reading data from a file.

To perform asynchronous operations in Dart, you can use the Future class and the async and await keywords.

Synchronous operation: A synchronous operation blocks other operations from executing until it completes.

Asynchronous operation: Once initiated, an asynchronous operation allows other operations to execute before it completes.

Synchronous function: A synchronous function only performs synchronous operations.

Asynchronous function: An asynchronous function performs at least one asynchronous operation and can also perform synchronous operations.

The Dart ecosystem uses packages to manage shared software such as libraries and tools. To get Dart packages, you use the pub package manager. You can find publicly available packages on the pub.dev site, or you can load packages from the local file system or elsewhere, such as Git repositories. Wherever your packages come from, pub manages version dependencies, helping you get package versions that work with each other and with your SDK version.
To use a package, do the following:

1. Create a pubspec (a file named pubspec.yaml that lists package dependencies and includes other metadata, such as a version number).
2. Use pub to get your package's dependencies.
3. If your Dart code depends on a library in the package, import the library.

4.7 Flutter:

Flutter is an open-source UI software development toolkit developed by google .it can be used to craft beautiful, natively compiled applications for mobile, web and desktop from a single codebase. From the release of its first version 'sky' its applications have cross platform and could be run on Android, IOS, Mac, Windows, Google Fushia from a single codebase. It has many built in widgets which makes developing an app so much easier and it working on different platforms doesn't mess with the UI design because it has the control of pixels over the screen and they could be customized as of will with built-in widgets or by creating new customized widgets. It is powered with the DART programming language.

What is the difference between a package and a plugin? A plugin is a type of package - the full designation is plugin package, which is generally shortened to plugin.

Packages:

At a minimum, a Dart package is a directory containing a pubspec file. Additionally, a package can contain dependencies (listed in the pubspec), Dart libraries, apps, resources, tests, images, and examples. The pub.dev site lists many packages—developed by Google engineers and generous members of the Flutter and Dart community— that you can use in your app.

Plugins:

A plugin package is a special kind of package that makes platform functionality available to the app. Plugin packages can be written for Android (using Kotlin or Java), iOS (using Swift or Objective-C), web, macOS, Windows, Linux, or any combination thereof. For example, a plugin might provide Flutter apps with the ability to use a device's camera.

Ensuring apps are accessible to a broad range of users is an essential part of building a high-quality app. Applications that are poorly designed create barriers to people of all ages. The UN Convention on the Rights of Persons with Disabilities states the moral and legal imperative to ensure universal access to information systems; countries around the world

enforce accessibility as a requirement; and companies recognize the business advantages of maximizing access to their services.

Flutter is declarative. This means that Flutter builds its user interface to reflect the current state of your app.

When the state of your app changes (for example, the user flips a switch in the settings screen), you change the state, and that triggers a redraw of the user interface. There is no imperative changing of the UI itself (like widget.setText)—you change the state, and the UI rebuilds from scratch.

Ephermal State:

Ephemeral state (sometimes called UI state or local state) is the state you can neatly contain in a single widget. This is, intentionally, a vague definition, so here are a few examples.

Current page in a PageView.

Current progress of a complex animation.

Current selected tab in a BottomNavigationBar.

Other parts of the widget tree seldom need to access this kind of state. There is no need to serialize it, and it doesn't change in complex ways. In other words, there is no need to use state management techniques (ScopedModel, Redux, etc.) on this kind of state. All you need is a StatefulWidget.

App state:

State that is not ephemeral, that you want to share across many parts of your app, and that you want to keep between user sessions, is what we call application state (sometimes also called shared state).

Examples of application state:

1) User preferences
2) Login info
3) Notifications in a social networking app
4) The shopping cart in an e-commerce app
5) Read/unread state of articles in a news app

For managing app state, you'll want to research your options. Your choice depends on the complexity and nature of your app, your team's previous experience, and many other aspects. There is no clear-cut, universal rule to distinguish whether a particular variable is ephemeral or app state. Sometimes, you'll have to refactor one into another. For example, you'll start with some clearly ephemeral state, but as your application grows in features, it might need to be moved to app.

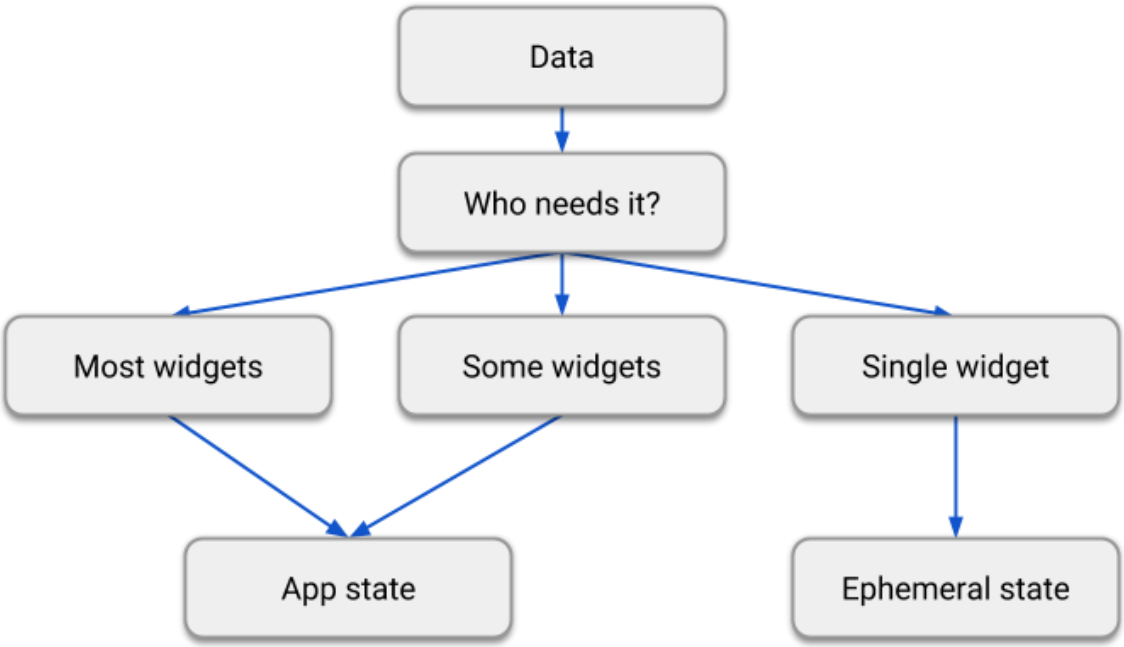


Fig 4.7: Ephemeral state

In summary, there are two conceptual types of state in any Flutter app. Ephemeral state can be implemented using State and setState(), and is often local to a single widget. Both types have their place in any Flutter app, and the split between the two depends on your own preference and the complexity of the app.

Chapter 5

5.1 Implementation:

The health care app is developed successfully as mentioned in previous chapters. And the features are working fine .doctor appointment is implemented successfully and the reminders for medicine and the appointment are working fine .they show the features of notifying when it is due time. finding the nearby location of the nearby hospital works fine and the routing and navigation is easier and shows real time traffic too. the beds inthehospital are shown accurately with the data uploaded in the datatable . the graphical representation of the health trackers are successful and insightful. note making is even simpler now as it can br written in the app itself. prioritization is working fine. alerting contacts at emergency situations is working fine with the intricate featuring process.

5.2 Models:

Widgets are temporary objects, used to construct a presentation of the application in its current state.

A widget's main job is to implement a build () function, which describes the widget in terms of other, lower-level widgets. The framework builds those widgets in turn until the process bottoms out in widgets that represent the underlying RenderObject, which computes and describes the geometry of the widget.

Flutter has one core principle: everything is a widget. There is a readily available widget for pretty much everything you will use in your app, including:

1. Stylistic elements like fonts and colors
2. Layout elements like margins and paddings
3. Graphic components like buttons, lists, images, menus, etc.

And also, for animations, user interaction, etc.

There are two types of widgets you should look out for: StatelessWidget and StatefulWidget. When you define your own widgets, you will extend from one or the other, depending on your needs.

When You want some static functionality in the app, Then StatelessWidget is meant for static things, like text, images, fonts, paddings and so on.When the screen is changing overtime then we use stateful widgets.

To integrate the widgets with the functionality desired models are used. Models are nothing but dart files that do the working of the functioning of the app.

5.2.1 Appointment.dart

This model is used to implement making appointments with the doctors. It takes name, place, address, date and time, notification id as strings.

```
class Appoinment {
  int _id;
  String _name;
  String _place;
  String _address;
  String _dateAndTime;
  int _notificationID;
  bool _done;
```

And a done variable of bool type to verify the appointment is booked or not.

```
Appoinment.fromMapObject(Map<String, dynamic> map) {
  this._id = map['id'];
  this._dateAndTime = map['date_time'];
  this._address = map['address'];
  this._place = map['place'];
  this._name = map['name'];
  this._notificationID = map['notification_id'];
  this._done = map['done'] == 1;
}
```

Appointment is confirmed with the input mapped to objects using this keyword.

5.2.2 Hospital.dart

This model is used to implement the getting the nearby location of the hospital. We use the user made packages developed by us i.e; other dart models.

```
import 'package:health_bloom/models/location.dart';
import 'package:health_bloom/others/constants.dart';
import 'package:health_bloom/others/network.dart';


class HospitalData {
  UserLocation userLocation;
  List<Hospital> hospitalList;
  HospitalData();
  getNearbyHospital() async {
    // ignore: deprecated_member_use
    this.hospitalList = List<Hospital>();
    userLocation = UserLocation();

    await userLocation.getLocation().then((value) {
      this.userLocation = value;
    });
```

this uses the user location and maps with the nearby hospitals.

```
String url =
    'https://api.tomtom.com/search/2/nearbySearch/.JSON?key=$kTomsApiKey&lat=${userLocation.latitude}
NetworkHelper networkHelper = NetworkHelper(url);
```

The input is taken in the form of string of the latitudinal and longitudinal locations, using the tomtom API.

```
String uri =
    'https://api.opencagedata.com/geocode/v1/json?q=$locationLat+$locationLon&key=f29cf18b10224e27b8931981
NetworkHelper _networkHelper = NetworkHelper(uri);
```

This is used get the vacant beds in the hospitals exploiting the opencage API.

5.2.3 Location.dart:

This model is the sub collection of the hospital.dart . This model gets the location of the user latitude and longitude with the geolocator.

```
import 'package:geolocator/geolocator.dart';

class UserLocation {
  double latitude, longitude;
  UserLocation({this.latitude, this.longitude});
  Future<UserLocation> getLocation() async {
    UserLocation userLocation = UserLocation();
```

This data gotten id then parsed as the Json string in double format.

5.2.4 Note.dart

This model is used to implement the notes feature of the application.

It contains the fields like id, title, description, datecreated, priority in different forms like int, string.

```
class Note {
  int _id;
  String _title;
  String _description;
  String _dateCreated;
  String _date;
  int _priority;

Map<String, dynamic> toMap() {
  var map = Map<String, dynamic>();
  if (id != null) {
    map['id'] = _id;
  }
  map['title'] = _title;
  map['description'] = _description;
  map['priority'] = _priority;
  map['dateCreated'] = _dateCreated;
  map['date'] = _date;
  return map;
}
```

This snippet converts the note object into map object.

```
Note.fromMapObject(Map<String, dynamic> map) {
  this._id = map['id'];
  this._title = map['title'];
  this._description = map['description'];
  this._priority = map['priority'];
  this._dateCreated = map['dateCreated'];
  this._date = map['date'];
}
```

This code snippet extracts a note object from a map object.

5.2.5 Relative.dart

This model implements the feature of alerting the emergency contacts. This module does the primary part of taking the data and storing in the database.

```
class Relative {
  String name, email, phoneNumber, uid, documentID;
  Relative();
  Relative getData(var data) {
    this.phoneNumber = data['phoneNumber'];
    this.email = data['email'];
    this.uid = data['uid'];
    this.name = data['name'];
    this.documentID = data.documentID;
    return this;
  }
```

It takes the name,email id,phone number and sets user id and document id to the contact.

```
Map<String, dynamic> toMap() {
  Map<String, dynamic> data = Map<String, dynamic>();
  data['phoneNumber'] = this.phoneNumber;
  data['email'] = this.email;
  data['uid'] = this.uid;
  data['name'] = this.name;
  return data;
}
```

This converts the relative objects into map objects.

5.2.6 Remainder.dart

This model implement the sending notifications to remind the appointments and the medicine prescriptions.

```
import 'dart:convert';
```

Encoders and decoders for converting between different data representations, including JSON and UTF-8. In addition to converters for common data representations, this library provides support for implementing converters in a way which makes them easy to chain and to use with streams.

```
class Reminder {
  int _id;
  String _name;
  String _type;
  int _times;
  String _time1, _time2, _time3;
  int _notificationID;
  Map<String, dynamic> _intakeHistory;

  Reminder(this._name, this._type, this._time1, this._time2, this._time3,
      this._times, this._notificationID, this._intakeHistory);
```

It takes the id, name, type, no. of times, notification id.

And also maps the intake history of the reminder object into map object.

```
Map<String, dynamic> toMap() {
  var map = Map<String, dynamic>();
  if (_id != null) {
    map['id'] = _id;
  }
  map['name'] = _name;
  map['type'] = _type;
  map['times'] = _times;
  map['time1'] = _time1;
  map['time2'] = _time2;
  map['time3'] = _time3;
  map['notification_id'] = _notificationID;
  String temp = jsonEncode(intakeHistory);
  map['intake_history'] = temp;
  return map;
}
```

Converting to map objects.

```
Reminder.fromMapObject(Map<String, dynamic> map) {
  this._id = map['id'];
  this._time1 = map['time1'];
  this._time2 = map['time2'];
  this._time3 = map['time3'];
  this._times = map['times'];
  this._type = map['type'];
  this._name = map['name'];
  this._notificationID = map['notification_id'];
  Map<String, dynamic> temp = jsonDecode(map['intake_history']);
  this._intakeHistory = temp;
}
```

Retrieving from the map objects.

5.2.7 Tracker.dart

This model is used to implement the health tracker feature in the app. We take objects into for many cases like blood pressure readings, sugar readings etc. and convert them into map objects. So that we can later retrieve and present them in the screen widget.

```
import 'package:cloud_firestore/cloud_firestore.dart';
```

Cloud_ firestore is imported to operate on it and make the inserting and deleting ,updating etc.

```
class TrackerModel {
  Sleep sleepTracker;
  WeightTracker weightTracker;
  BloodPressureTracker bloodPressureTracker;
  BloodSugarTracker bloodSugarTracker;
  TrackerModel();
}
```

Implementing different trackers in the model like for sleep, weight, bloodpressure and blood sugar.

a) sleep

```
class Sleep {
  int hours, minutes;
  String notes;
  DateTime dateTime;

  Sleep({this.hours, this.minutes, this.notes, this.dateTime});
}
```

Sleep class with inputs as hours or minutes slept, along with the notes describing the sleep or sleeping conditions, and dateTime of the period.

```
Map<String, dynamic> toMap() {
  Map<String, dynamic> map = Map<String, dynamic>();

  map['dateAndTime'] = this.sleepData.dateTime.toString();
  map['hours'] = this.sleepData.hours.toString();
  map['minutes'] = this.sleepData.minutes.toString();
  map['notes'] = this.sleepData.notes;

  return map;
}
```

Converting the above inputs into map objects.

```
fromMap(Map<String, dynamic> map) {
  Sleep sleepTracker = Sleep();
  sleepTracker.dateTime = DateTime.parse(map['dateAndTime']);
  sleepTracker.hours = int.parse(map['hours']);
  sleepTracker.minutes = int.parse(map['minutes']);
  sleepTracker.notes = map['notes'];
  return sleepTracker;
}
```

Retrieving from the map objects.

```
List<Sleep> loadData(QuerySnapshot snapshot) {
  List<DocumentSnapshot> documents = snapshot.documents;
  List<Sleep> sleepList = [];
  for (var data in documents) {
    Map map = data.data;
    sleepList.add(this.fromMap(map));
  }
  return sleepList;
}
```

Depicting the data in the form of list as they dynamic and used to represent pictorially later in screen.

b) weight

```
class WeightTracker {
  bool isTracking;
  Weight weightData;
  WeightTracker();
```

Implementing the weight class as a list of inputs.

```
Map<String, dynamic> toMap() {
  Map<String, dynamic> map = Map<String, dynamic>();

  map['dateAndTime'] = this.weightData.dateTime.toString();
  map['weight'] = this.weightData.weight.toString();
  map['notes'] = this.weightData.notes;

  return map;
}
```

Converting into the map objects.

```
fromMap(Map<String, dynamic> map) {
  Weight weightTracker = Weight();
  weightTracker.dateTime = DateTime.parse(map['dateAndTime']);
  weightTracker.weight = int.parse(map['weight']);

  weightTracker.notes = map['notes'];
  return weightTracker;
}
```

Retrieving from the map objects.

```
List<Weight> loadData(QuerySnapshot snapshot) {
  List<DocumentSnapshot> documents = snapshot.documents;
  List<Weight> weightList = [];
  for (var data in documents) {
    Map map = data.data;
    weightList.add(this.fromMap(map));
  }
  return weightList;
}
```

Representing the data in the form of the list.

c) glucose

```
class BloodSugar {
  int bloodSugar;
  String notes;
  DateTime dateTime;
  BloodSugar({this.bloodSugar, this.notes, this.dateTime});
}
```

Implementing the class of BloodSugar to track the glucose levels in the blood.

```
Map<String, dynamic> toMap() {
  Map<String, dynamic> map = Map<String, dynamic>();

  map['dateAndTime'] = this.bloodSugar.dateTime.toString();
  map['blood_sugar'] = this.bloodSugar.bloodSugar.toString();
  map['notes'] = this.bloodSugar.notes;

  return map;
}
```

To convert the objects into map objects.

```
fromMap(Map<String, dynamic> map) {
  BloodSugar bloodSugar = BloodSugar();
  bloodSugar.dateTime = DateTime.parse(map['dateAndTime']);
  bloodSugar.bloodSugar = int.parse(map['blood_sugar']);

  bloodSugar.notes = map['notes'];
  return bloodSugar;
}
```

To retrieve from the map Objects.

```
List<BloodSugar> loadData(QuerySnapshot snapshot) {
  List<DocumentSnapshot> documents = snapshot.documents;
  List<BloodSugar> bloodSugarList = [];
  for (var data in documents) {
    Map map = data.data;
    bloodSugarList.add(this.fromMap(map));
  }
  return bloodSugarList;
}
```

Listing the objects for the later retrieval on tap in the screen widget.

d) Blood pressure:

```
class BloodPressure {
  int systolic, diastolic, pulse;
  String notes;
  DateTime dateTime;

  BloodPressure(
      {this.systolic, this.diastolic, this.pulse, this.notes, this.dateTime});
}
```

Creating a class for bloodpressure with systole, diastole pulses along with notes and time of the datetime.

```
Map<String, dynamic> toMap() {
  Map<String, dynamic> map = Map<String, dynamic>();

  map['dateAndTime'] = this.bloodPressure.dateTime.toString();
  map['diastolic'] = this.bloodPressure.diastolic.toString();
  map['systolic'] = this.bloodPressure.systolic.toString();
  map['pulse'] = this.bloodPressure.pulse.toString();
  map['notes'] = this.bloodPressure.notes;

  return map;
}
```

Converting the objects into map objects.

```
fromMap(Map<String, dynamic> map) {
  BloodPressure bloodPressure = BloodPressure();
  bloodPressure.dateTime = DateTime.parse(map['dateAndTime']);
  bloodPressure.diastolic = int.parse(map['diastolic']);
  bloodPressure.systolic = int.parse(map['systolic']);
  bloodPressure.pulse = int.parse(map['pulse']);

  bloodPressure.notes = map['notes'];
  return bloodPressure;
}
```

Retrieving from the map objects.

```
List<BloodPressure> loadData(QuerySnapshot snapshot) {
  List<DocumentSnapshot> documents = snapshot.documents;
  List<BloodPressure> bloodPressureList = [];
  for (var data in documents) {
    Map map = data.data;
    bloodPressureList.add(this.fromMap(map));
  }
  return bloodPressureList;
}
```

Listing the values and to be represented on the screen of widget.

5.2.8 User.dart

This model is used to store the profile data of the user.

```
class UserProfile {
  String allergies,
      userName,
      age,
      gender,
      bloodGroup,
      bloodPressure,
      bloodSugar,
      email,
      height,
      weight,
      phoneNumber,
      picture,
      uid;
```

The above parameters are taken username,age,gender,blood group,blood pressure,blood sugar,email, height,weight,phone number,pic ,uid.

Data is imported from the relative.dart model also.

```
getAllRelatives(var data) {
  // ignore: deprecated_member_use
  this.relatives = List<Relative>();
  for (var relative in data) {
    Relative _relative = Relative();
    _relative.getData(relative);
    this.relatives.add(_relative);
  }
}
```

To add to the relatives.

```
deleteRelative(String documentID) {
  Relative _relative = Relative();
  bool found = false;
  for (var r in this.relatives) {
    if (r.documentID == documentID) _relative = r;
  }
  if (found) {
    this.relatives.remove(_relative);
  }
}
```

To delete the data of the relatives i.e. emergency contacts.

5.3 Other Models:

5.3.1.Auth.dart:

This model is used to the authentication of the app,

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/services.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:shared_preferences/shared_preferences.dart';
```

The list of packages imported to make authentication successful.

Cloud_firestore package is imported to use map to store, shared preferences package to implement shared preferences, sign_in dart package to perform the google sign-in operation

```
class User {
  User({@required this.uid});
  final String uid;
}
```

User class declaration making the user id as not null attribute.

```
abstract class AuthBase {
  Stream<User> get onAuthStateChanged;
  Future<User> currentUser();
  Future<User> signInWithGoogle();
  Future<void> signOut();
}
```

An abstract class is defined and the methods are declared beforehand for taking the values to be returned later.

```
class Auth implements AuthBase {
  final _firebaseAuth = FirebaseAuth.instance;

  User _userFromFirebase(FirebaseUser user) {
    if (user == null) {
      return null;
    }
    return User(uid: user.uid);
  }
```

To check whether the user is present in the database or not, and if present returns user id.

Implementation of the sign- in with google is done asynchronously.

```
SharedPreferences prefs;
prefs = await SharedPreferences.getInstance();

if (!userExits) {
  prefs.setBool('first', true);
  try {
    await Firestore.instance
        .collection('profile')
        .document(authResult.user.uid.toString())
        .setData({
```

Wraps NSUserDefaults (on iOS) and SharedPreferences (on Android), providing a persistent store for simple data. Data is persisted to disk asynchronously.

```
@override
Future<void> signOut() async {
  final googleSignIn = GoogleSignIn();
  await googleSignIn.signOut();

  await _firebaseAuth.signOut();
}
```

Sign out is also an asynchronous function. It is a delayed computation so future is used.

```
resetPasswordLink(String email) {
  FirebaseAuth.instance.sendPasswordResetEmail(email: email);
}
```

To reset a password by sending verification through email.

5.3.2 Constants.dart:

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

It is a package that contains all the methods that are used to make material design in Flutter.

```
fontSize: 25.0,
```

The size of glyphs (in logical pixels) to use when painting the text. During painting, the [fontSize] is multiplied by the current textScaleFactor to let users make it easier to read text by increasing its size. [getParagraphStyle] will default to 14 logical pixels if the font size isn't specified here.

```
color: Colors.white,
```

The color to use when painting the text. If [foreground] is specified, this value must be null. The [color] property is shorthand for Paint().color = color.In [merge], [apply], and [lerp], conflicts between [color] and [foreground] specification are resolved in [foreground]'s favor - i.e. if [foreground] is specified in one place, it will dominate [color] in another.

```
const String kOpenCageApiKey = 'f29cf18b10224e27b8931981380b747a';
const kTomsApiKey = 'vA9uQILIGUAG86z9xCTSkETjqg7ZCiGa';
```

keys of tomcat API and Opencage API that are used in this project.

5.3.3 Database_helper.dart:

```
import 'package:sqflite/sqflite.dart';
import 'dart:async';
import 'dart:io';
import 'package:path_provider/path_provider.dart';
import '../models/note.dart';
import '../models/reminder.dart';
import '../models/appoinment.dart';
```

Dart:async package is used as support for asynchronous programming, with classes such as Future and Stream.

Dart:io package is for File, socket, HTTP, and other I/O support for non-web applications.

```
DatabaseHelper._createInstance();
```

Named constructor to create instance of DatabaseHelper.

```
Directory directory = await getApplicationDocumentsDirectory();
String path = directory.path + 'app.db';
```

Get the directory path for both Android and iOS to store database.

```
var appDatabase = await openDatabase(path, version: 1, onCreate: _createDb);
return appDatabase;
```

Open/create the database at a given path.

```
Future<List<Map<String, dynamic>>> getNoteMapList() async {
  Database db = await this.database;
```

Fetch Operation: Get all note objects from database

```
Future<List<Map<String, dynamic>>> getReminderMapList() async {
  Database db = await this.database;
```

Fetch Operation: Get all reminder objects from database

```
Future<List<Map<String, dynamic>>> getAppoinmentMapList() async {
  Database db = await this.database;
```

Fetch Operation: Get all appointment objects from database

```
Future<int> insertNote(Note note) async {
  Database db = await this.database;
  var result = await db.insert(noteTable, note.toMap());
  return result;
}
```

Insert Operation: Insert a Note object to database

```
Future<int> insertReminder(Reminder reminder) async {
  Database db = await this.database;
  var result = await db.insert(reminderTable, reminder.toMap());
  return result;
}
```

Insert Operation: Insert a reminder object to database

```
Future<int> insertAppoinment(Appoinment appoinment) async {
  Database db = await this.database;
  var result = await db.insert(appoinmentTable, appoinment.toMap());
  return result;
}
```

Insert Operation: Insert an appointment object to database

```
Future<int> updateNote(Note note) async {
  var db = await this.database;
  var result = await db.update(noteTable, note.toMap(),
      where: '$colId = ?', whereArgs: [note.id]);
  return result;
}
```

Update Operation: Update a Note object and save it to database

```
Future<int> updateReminder(Reminder reminder) async {
  var db = await this.database;
  var result = await db.update(reminderTable, reminder.toMap(),
      where: '$remColId = ?', whereArgs: [reminder.id]);
  return result;
}
```

Update Operation: Update a reminder object and save it to database

```
Future<int> updateAppoinment(Appoinment appoinment) async {
  var db = await this.database;
  var result = await db.update(appoinmentTable, appoinment.toMap(),
      where: '$appoinmentColId = ?', whereArgs: [appoinment.id]);
  return result;
}
```

Update Operation: Update an appointment object and save it to database

```
Future<int> deleteNote(int id) async {
  var db = await this.database;
  int result =
      await db.rawDelete('DELETE FROM $noteTable WHERE $colId = $id');
  return result;
}
```

Delete Operation: Delete a Note object from database

```
Future<int> deleteReminder(int id) async {
  var db = await this.database;
  int result =
      await db.rawDelete('DELETE FROM $reminderTable WHERE $remColId = $id');
  return result;
}
```

Delete Operation: Delete a Reminder object from database

```
Future<int> deleteAppoinment(int id) async {
  var db = await this.database;
  int result = await db
      .rawDelete('DELETE FROM $appoinmentTable WHERE $appoinmentColId = $id');
  return result;
}
```

Delete Operation: Delete an appointment object from database

```
Future<int> getCount() async {
  Database db = await this.database;
  List<Map<String, dynamic>> x =
      await db.rawQuery('SELECT COUNT (*) from $noteTable');
  int result = Sqflite.firstIntValue(x);
  return result;
}
```

Get number of Note objects in database

```
Future<int> getRemCount() async {
  Database db = await this.database;
  List<Map<String, dynamic>> x =
      await db.rawQuery('SELECT COUNT (*) from $reminderTable');
  int result = Sqflite.firstIntValue(x);
  return result;
}
```

Get number of reminder objects in database

```
Future<int> getAppoinmentCount() async {
  Database db = await this.database;
  List<Map<String, dynamic>> x =
      await db.rawQuery('SELECT COUNT (*) from $appoinmentTable');
  int result = Sqflite.firstIntValue(x);
  return result;
}
```

Get number of appointment objects in database

5.3.4 Functions.dart:

This model is used to make the material design of widgets and know about the details of a window object.

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

double getDeviceHeight(BuildContext context) {
  MediaQueryData deviceInfo = MediaQuery.of(context);
  return deviceInfo.size.height;
}

double getDeviceWidth(BuildContext context) {
  MediaQueryData deviceInfo = MediaQuery.of(context);
  return deviceInfo.size.width;
}
```

MediaQueryData is the information about a piece of media (e.g., a window). For example, the [MediaQueryData.size] property contains the width and height of the current window.To obtain the current [MediaQueryData] for a given [BuildContext], use the [MediaQuery.of] function. For example, to obtain the size of the current window, use MediaQuery.of(context).size.

If no [MediaQuery] is in scope then the [MediaQuery.of] method will throw an exception. Alternatively, [MediaQuery.maybeOf] may be used, which returns null instead of throwing if no [MediaQuery] is in scope.

Buildcontext is a handle to the location of a widget in the widget tree.This class presents a set of methods that can be used from [StatelessWidget.build] methods and from methods on [State] objects.[BuildContext] objects are passed to [WidgetBuilder] functions (such as [StatelessWidget.build]), and are available from the [State.context] member. Some static functions (e.g. [showDialog], [Theme.of], and so forth) also take build contexts so that they can act on behalf of the calling widget, or obtain data specifically for the given context.

5.3.5 Network.dart:

```dart
import 'package:http/http.dart' as http;
import 'dart:convert';
```

Dart convert is a package of encoders and decoders for converting between different data representations, including JSON and UTF-8. In addition to converters for common data representations, this library provides support for implementing converters in a way which makes them easy to chain and to use with streams.

```
class NetworkHelper {
  final String url;
  NetworkHelper(this.url);

  Future getData() async {
    http.Response response = await http.get(url);

    if (response.statusCode == 200) {
      var data = response.body;
      return jsonDecode(data);
    } else {
      print(response.statusCode);
    }
  }
}
```

Status code 200 is the accepted string value of a response and all remaining status codes just signify the type of error that could have occurred.

5.3.6 Notification_service.dart:

This model is implemented for various notification services used in the app.

```
scheduleAppoinmentNotification(
    {@required int id,
    @required String title,
    @required String body,
    @required DateTime dateTime}) async {
  var scheduledNotificationDateTime = dateTime;
```

It gives the notification for a scheduled appointment.

Notifications can be scheduled periodically and each periodic function has a method implemented for it.

dailyNotification();
dailyMedicineNotification(param);
weeklyNotification();
notificationDetails();
To implement on receiving functionality.

Chapter 6

6.1 Work flow of Code:

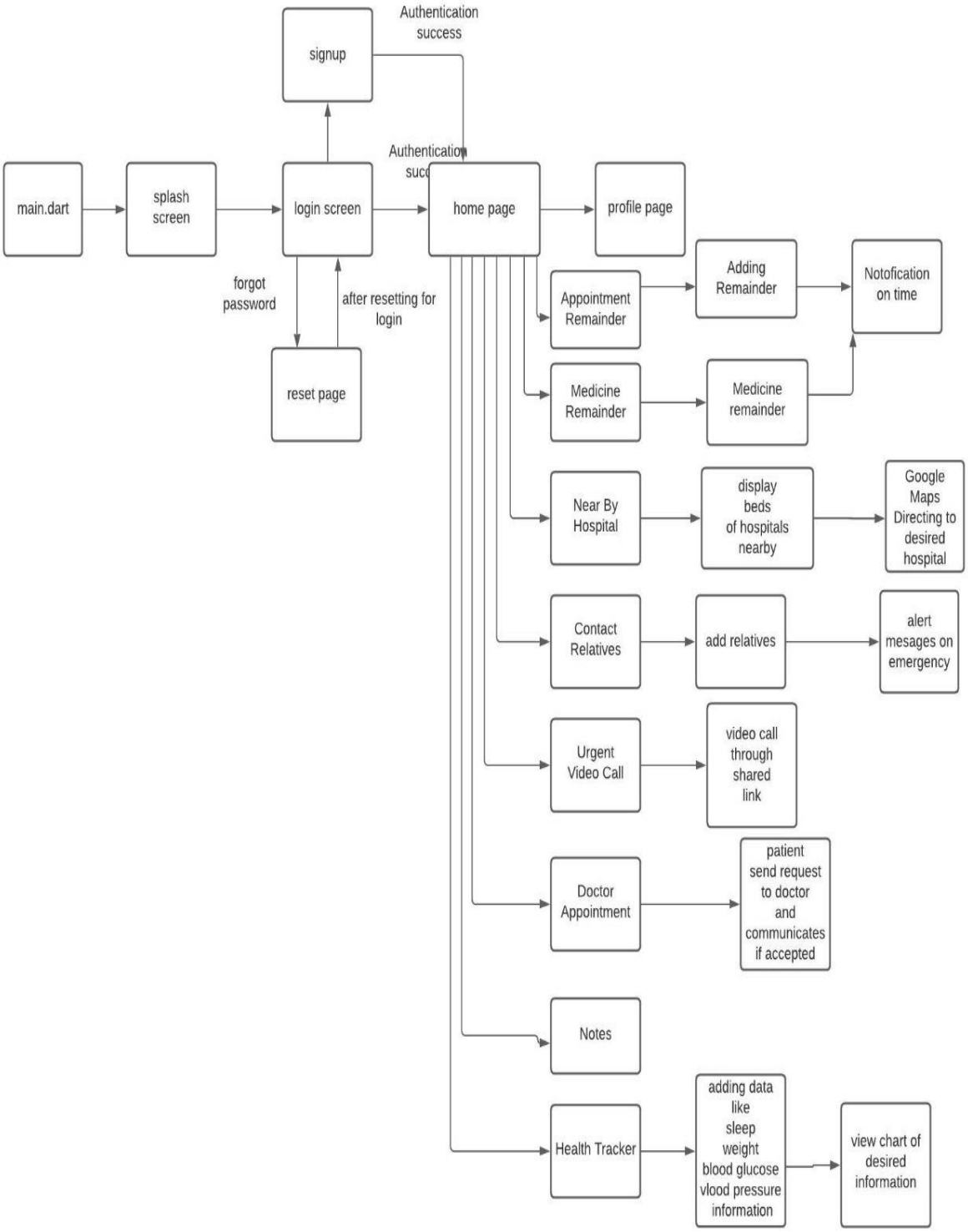
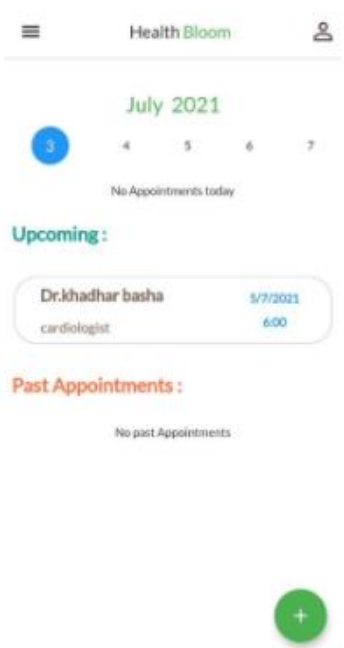


Fig 6.1 Work flow of code

6.2 Result Analysis:

Home page:

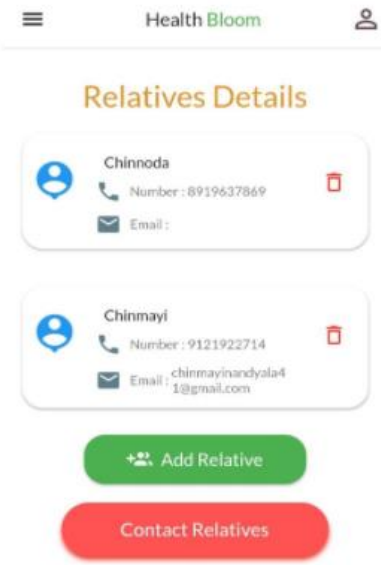
Remainder Appointment:



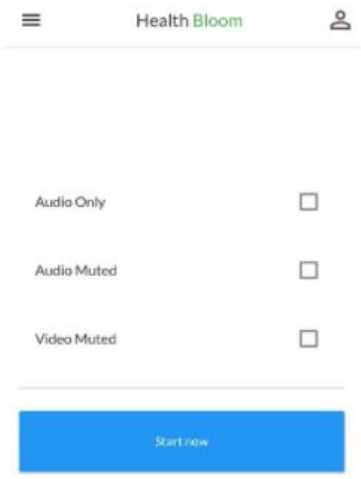
Medicine Appointment:

Locate Near By Hospitals:

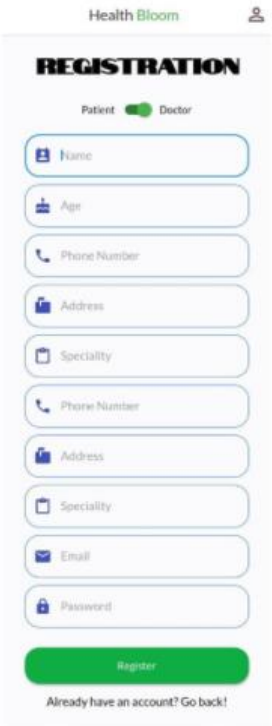
Contact Relatives:



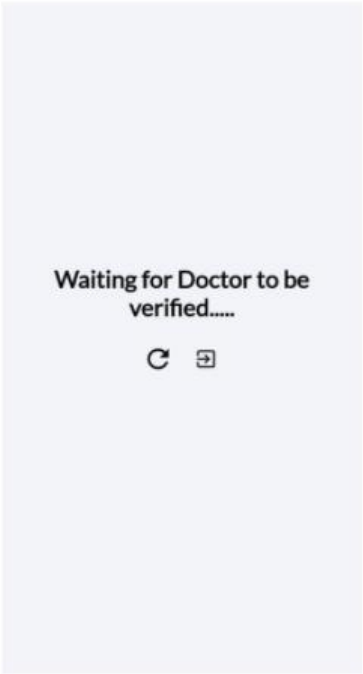
Video Calls:



Doctor Appointment:



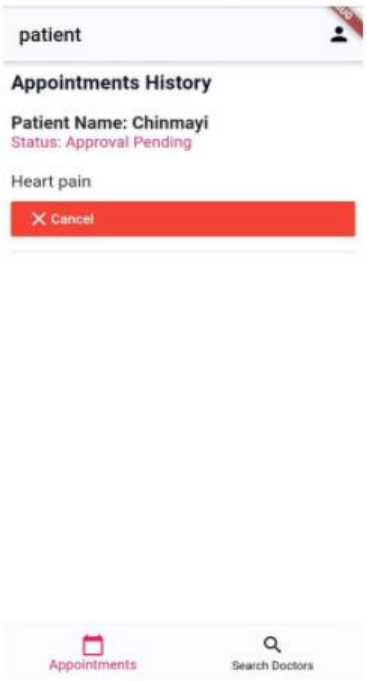
Doctor waiting To accept:



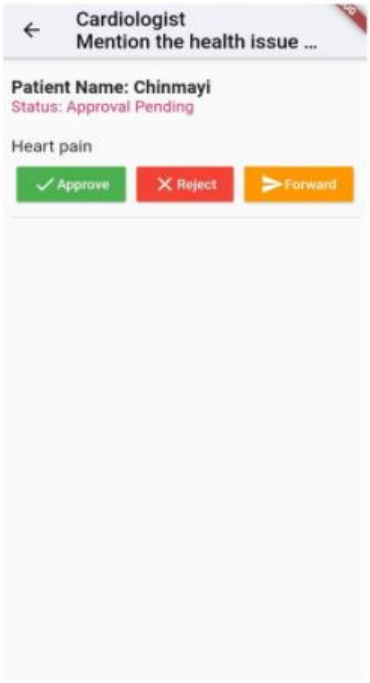
Doctor Appointment Window:

Searching for doctor:

Patient waiting To accept:

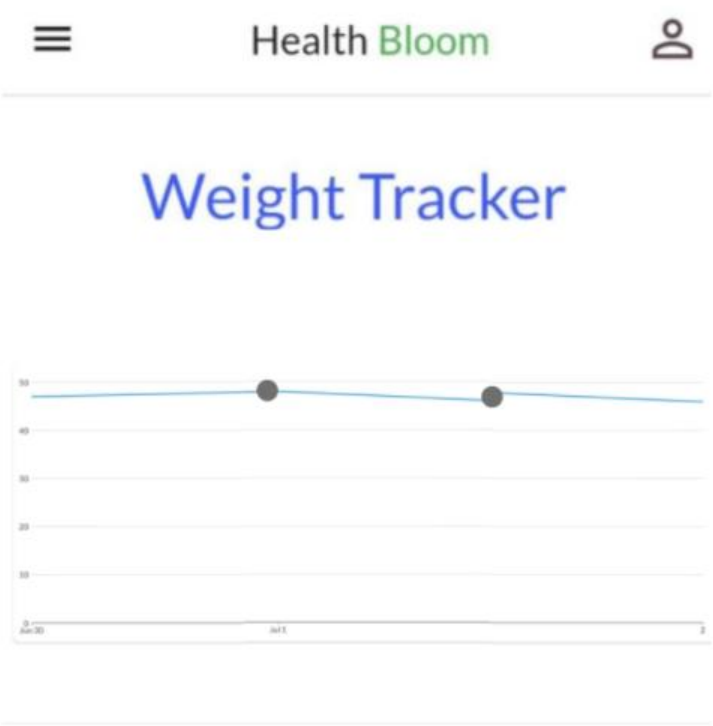


Doctor Appointment Window:



Notes:

Health Tracker:



Chapter 7

7.1 Conclusion:

In toto, as shown in the previous chapters, all these features help to make user friendly UI for our Health Bloom app. So, searching for nearby hospital and locating it and making a doctor appointment are easier. And all the reminders help us not to miss any appointments or prescriptions. We sure of our app would do more service than we expect it to be with all the features embedded.

7.2 Objects justification:

The objectives of making the doctor appointment with authentication and authorization are working fine and appointment reminder and nearby hospital locator and each module has been tested thoroughly and debugging is done carefully checking each and every functionality of the workflow diagram. Each component is implements successfully as shown in the results of the previous chapter.

7.3 Further enhancements:

Even though our application looks exhaustive there are still some features that could be added into the app. Like adding the nearby medical pharma's or including home delivery of medicines etc. As of now including the home delivery of medicines is a sensitive issue that could lead to unforeseen problems, but the model could be implemented once the laws of e commerce are made clearer for drug transfers.