VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

An Internship Project Report

on

Covid-19 Tracker App

Submitted in partial fulfilment of the requirements for the VIII Semester of degree of **Bachelor of Engineering in Information Science and Engineering** of Visvesvaraya Technological University, Belagavi

by

Shreya Uday Hasyagar

1RN18IS101

Under the Guidance of

Mrs. Shwetha GN

Assistant Professor

Department of ISE

Department of Information Science and Engineering

RNS Institute of Technology

Dr. Vishnuvaradhan Road, Rajarajeshwari Nagar post, Channasandra, Bengaluru-560098

2021-2022

RNS INSTITUTE OF TECHNOLOGY

Dr. Vishnuvaradhan Road, Rajarajeshwari Nagar post,

Channasandra, Bengaluru - 560098

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Internship work entitled ***Covid-19 Tracker App*** has been successfully completed by **Shreya Uday Hasyagar (1RN18IS101)** bonafide students of **RNS Institute of Technology, Bengaluru** in partial fulfilment of the requirements of 8th semester for the award of degree in **Bachelor of Engineering in Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during academic year **2021-2022.** The internship report has been approved as it satisfies the academic requirements in respect of internship work for the said degree.

Mrs. Shwetha GN	**Dr. Suresh L**	**Dr. M K Venkatesha**
Internship Guide	Professor and HoD	Principal
Assistant Professor	Department of ISE	RNSIT
Department of ISE	RNSIT	

External Viva

Name of the Examiners	**Signature with Date**
1. _____	**1.** _____
2. _____	**2.** _____

DECLARATION

I, **Shreya Uday Hasyagar [USN: 1RN18IS101]** student of VIII Semester BE, in Information Science and Engineering, RNS Institute of Technology hereby declare that the Internship work entitled ***Covid-19 Tracker App*** has been carried out by us and submitted in partial fulfilment of the requirements for the *VIII Semester degree of* ***Bachelor of Engineering in Information Science and Engineering*** *of Visvesvaraya Technological University, Belagavi* during academic year 2021-2022.

Place: Bengaluru

Date:

SHREYA UDAY HASYAGAR

(1RN18IS101)

ABSTRACT

The World Health Organization has declared the outbreak of the novel coronavirus, Covid-19 as pandemic across the world. With its alarming surge of affected cases throughout the world, lockdown, and awareness (social distancing, use of masks etc.) among people are found to be the only means for restricting the community transmission. In a densely populated country like India, it is very difficult to prevent the community transmission even during lockdown without social awareness and precautionary measures taken by the people.

COVID-19 has become biggest impediment for the survival of the human race, at present. Again, as mobile technology is now an important component of human life, hence it is possible to use the power of mobile technology against the treat of COVID-19. Every nation is now trying to deploy an interactive platform for creating public awareness and share the important information related to COVID-19. Keeping all of these in mind, an attempt to deploy an interactive cross-platform (web/mobile) application COVID-19 TRACKER for the ease of the users.

The application is featured with all the real-time attributes about the novel coronavirus disease and its measures and controls. To achieve all these functionalities, many tools and APIs are used in this application. The system purposely aims to maintain the digital protection of the society, create public awareness, and not create any agitation situation among the individuals of the society.

ACKNOWLEDGMENT

At the very onset I would like to place my gratefulness to all those people who helped me in making the Internship a successful one.

Coming up, this internship to be a success was not easy. Apart from the sheer effort, the enlightenment of the very experienced teachers also plays a paramount role because it is they who guided me in the right direction.

First of all, I would like to thank the **Management of RNS Institute of Technology** for providing such a healthy environment for the successful completion of internship work.

In this regard, I express sincere gratitude to our beloved Principal **Dr. M K Venkatesha,** for providing us all the facilities.

I am extremely grateful to our own and beloved Professor and Head of Department of Information Science and Engineering, **Dr. Suresh L**, for having accepted to patronize me in the right direction with all his wisdom.

I place my heartfelt thanks to **Mrs. Shwetha GN** Assistant Professor, Department of Information Science and Engineering for having guided internship and all the staff members of the department of Information Science and Engineering for helping at all times.

I thank **Mr. Akshay D R, Co-Founder & CEO of ENMAZ**, for providing the opportunity to be a part of the Internship program and having guided me to complete the same successfully.

I also thank our internship coordinator **Dr. R Rajkumar,** Associate Professor, Department of Information Science and Engineering. I thank my friends for having supported me with all their strength and might. Last but not the least, I thank my parents for supporting and encouraging me throughout. I have made an honest effort in this assignment.

SHREYA UDAY HASYAGAR

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

COVID-19	-	Corona Virus Disease 19
WHO	-	World Health Organization
PHEIC	-	Public Health Emergency of International concern
REST API	-	Representational State Transfer Application Programming Interface
UI	-	User Interface
SOAP	-	Simple Object Access Protocol
HTTP	-	Hypertext Transfer Protocol
BLOC	-	Business Logic Component
JSON	-	JavaScript Object Notation
JS	-	JavaScript
JSP	-	Java Server Pages
ASP	-	Active Server Pages
BSD	-	Berkeley Software Distribution
OS	-	Operating System

1. INTRODUCTION

1.1 Background

In Wuhan city of China, due to some unknown causes a local outbreak of pneumonia was notice in December 2019. Originally, the disease got named serious acute respiratory syndrome coronavirus 2 (SARS-COV-2). Corona Virus Disease 19 (COVID-19) caused by the novel coronavirus is confirmed officially later by the World Health Organization (WHO) and they also declare COVID-19 as the Public Health Emergency of International Concern (PHEIC). This ongoing outbreak of pneumonia spread rapidly all over the globe, causing more than 1,83,820 deaths soon got identified as the cause of the novel coronavirus. As of April 25, 2020, more than 22,398 confirmed COVID-19 cases found in INDIA. Social distancing was found to be a promising way to saturate the growth of the disease. Thus, most of the nations declared lockdown of the entire state to set a bar on the growing numbers of the disease. But it was found 42% of the society was not aware of the pandemic yet, and the lockdown caused a huge economic break.

Keeping all of these complex features in mind we have used all the promising technologies to build the mobile application as well as a web application. The application is capable to provide real-time worldwide cases. The application also provides information on different preventive measures to be taken by users and symptoms of COVID-19.

Nowadays, wireless communication in terms of mobile technology is an essential part of our life. This technology is now the guiding force for human life to perform any activities. Again, in this pandemic situation, fake news prevention and distribute genuine information, i.e., information regarding public awareness, test centers, geo-tracking, etc. to everyone in every time with minimum expense, it is perhaps the biggest challenge to any administration all over the world. Mobile technology is the only one stop solution for everything above mentioned problem.

1.2 Existing Systems

In this pandemic condition, a couple of mobile applications are introduced by government institutions and others, which are providing valuable information to the people to aware of the pandemic situation, like the number of confirmed cases, number of recovered, number

of deaths as of now, etc. APOLLO, one of the biggest medical giants in India, developed a risk assessment scanner for COVID-19 outbreak in INDIA and named it “covid. apollo247”.

This risk assessment scanner comes with a tag line “Stay calm amidst the current paranoia surrounding COVID-19” and is successful in reaching a major part of the society during this outbreak. The solution could efficiently predict the chances of risk (Low, Mid, High) of having positive coronavirus tress depending on multiple users given parameters using their ML algorithms. A team of twenty plus members (teachers and students) from Mahindra Ecole Centrale developed a dashboard for smooth user experience and quality visualization of real-time data. The website shows the real-time data of the infected, cured, and deaths in INDIA, though some inconsistency is also observed in their application. Several graphs and charts are used in the website to visualize the data also at the district level, but more accuracy is expected concerning their results.

The Government of India took the initiative and developed a mobile application named AarogyaSetu to connect essential health services with the people of INDIA. Basically, it is a mobile application; still, the use of web views was found in the statistics display dashboard. They have also come up with a geo-tracking facility that helps the system to track the coronavirus affected people and form a cluster. MICROSOFT BING has taken the novel attempt to implement a software solution for the COVID-19 pandemic across the world for data transparency and increasing traceability. Bing map comes with an interactive platform that focused on the world map. When hovering over the different countries, it used to render the COVID-19 statistics (Confirmed cases, Active cases, Deaths, Recovered) of the particular nation.

1.3 Proposed System

An application with a lot of information may be confusing for the user with less knowledge on how to use mobile/web. We propose a simple Covid-19 Tracker which provides most important functionalities. The application uses API to fetch the real-time number of active cases, number of recovered and deaths. It provides means to find any country of his choice and allow the user to set a default country for frequent updates.

The application provides users information on top COVID-19 symptoms and precautions to be taken to prevent from getting affected. It also tries to burst some myths related to COVID-19 and provides important information about the virus.

2. LITERATURE REVIEW

A Literature survey describes the concept of how the concept has emerged, how it has been implemented and what is the current status.

In Paper [1], authors have explained that with the increased awareness of people to the importance of regular health check-up, healthcare-providing facilities such as hospitals and clinics are struggling to cope with the demands of an ever-increasing population. In this paper, a healthcare mobile application prototype was developed successfully and installed on an Android smartphone. The application offers convenient telemedicine services that enable patients to be diagnosed online by medical specialists. The application also facilitates the ordering of medications using online payment. Medications can be delivered directly to the desired location, and in emergency cases, patients can call for ambulance service. The system enables medical doctors and nurses to monitor their patient's health continuously. Pharmacies can also use the application to offer and deliver their medical products to patients. The recent Covid-19 pandemic has made mobile health applications very popular. In this paper, authors have used Flutter to develop an integrated mobile healthcare application that enables reliable, fast, and most importantly affordable healthcare delivery application. The concept of the mobile healthcare application is explained. The prototype of the application was developed in phases. Phase 1 revolved around selecting the system components followed by phase 2 which was system installation and setup. Phase 3 mainly focused on application system architecture followed by phase 4 which was developing and testing the proposed application. The final phase involved installation of application on mobiles and final testing. While discussing the results, authors explain the complete working of the application and provide step by step representation of each screen of the app along with its functionality. This paper discussed a healthcare delivery mobile application prototype that was developed successfully using Flutter, and launched on an Android smart phone. Future enhancement and the final version of the application that can be developed is examined and recorded for further studies.

In Paper [2], the issue faced by Flutter developers with a state management when developing their applications is discussed. This paper proposes a new Flutter architecture based on the Clean Architecture by Uncle Bob. The Flutter Clean Architecture (FCA) proposed in this paper is packaged and released through a Flutter package. The architecture is tested by developing a full application from scratch using the package and documenting the process. The Flutter Clean

Architecture provides a solution to the state management problem as well as a potential overall choice for Flutter mobile application architecture. This paper aims to propose, test, and evaluate an architectural model and a state management technique for Flutter. The state management problem is discussed in this paper. Flutter is composed of widgets, and a widget can either be stateful or stateless. The Flutter framework creates states for its widgets that can be read synchronously when the widget is built and can be modified throughout the lifecycle of the application. The fundamental purpose of the proposed Flutter clean architecture is separation of concerns and scalability. By dividing the system into layers that separate business logic from platform-specific implementation, three crucial principles of Software Engineering are maintained i.e., Framework Agnosticism Testability and Isolation from External Modules. Each of these principles are discussed in detail and FCA flow diagram is presented. While analysing the implementation of the architecture, layers such as Data and Platform Layer, Application Layer and Domain Layer working are explained. The proposed architecture is tested on different platforms and no noticeable lag was present due to the added overhead and testers reported snappy performance. In this paper, the implementation of the Clean Architecture through the use of the Flutter Clean Architecture package proved to be very effective and advantageous. Combined with ease of testing, the Clean Architecture sped up the development process by maximizing the potential for working in parallel with other programmers. The Flutter Clean Architecture is a solution to Flutter's State Management problem, while also being an option for a Flutter mobile application architecture.

In Paper [3], exponential development of versatile application advancement is analysed. To finish this survey paper and learn about the subject an aggregate of four research papers were utilized which comprehended reasonable flow situation of Cross-stage versatile application advancement. MCIDER is fundamentally a working framework similarity engineering that can run applications worked for various versatile biological systems ideally iOS and Android together on the equivalent smartphone or tablet. Examination of most recent cross-stage mobile application improvement approaches which are at present accessible in the market are successfully weighed up in this paper. This exploration paper investigated how unique devices which are as of now accessible in the market. An application is developed on Flutter for farming solution and it contains features that help farmers plan their crop cycles and accordingly decide the optimum time or climate for growing any crop. The main objective of the app is to provide farmers with a complete solution to help maximize their yield and resolve their queries. The paper mentions the key features of the App and the information it provides in the lifecycle of

farming and the system architecture diagram for the same is explained. Various IoT devices used for collecting information about the field conditions and advantages of the proposed system is talked through and this project's future scope are predicted. The main objective of the project mentioned in this paper is to provide farmers with a complete solution to help them obtain the maximum yield and resolve queries. This project aims to revolutionize agriculture in India by introducing farmers to smart and simple solutions much like the proposed app that allows farmers to adapt and implement new methods that help them better manage their crops and obtain a profitable yield.

In Paper [4], Cross-platform mobile application development priority in today's world and generation is considered. Developers are enforced to either construct the same application numerous times for various operating systems or accept a low-quality similar solution that trades native speed and accuracy for portability. In this paper authors explain Flutter and its features and advantages over other technologies. The authors have developed a Flutter based mobile application for Billing and Reward system. The redeem point system emphasizes on the principle that the more money you spend, the more points you get in return. Each time a customer purchases something, they get a definite sum of points depending on how much they've purchased. The paper describes the working of the App and its features involved in it. The objective of the App is that by rewarding the customers with redeemable points, customer's average order value is increased thus, encouraging them to invest in the brand. The authors propose a system where the system consists of two parts client and server. The client side is deployed on an Android or iOS based mobile phone. The server side is deployed on a Windows OS. The paper is concluded with the contribution of the App to the society. The App is helping the retailers and small to medium shop owners to attract new customers, retain existing ones, and motivate increased purchase among current consumers.

In Paper [5], a study has been undertaken to indicate the benefits of using Flutter over other app development platforms. The paper consists of working of different platforms and their role in application development. Cross-platform frameworks that show resemblance to Flutter were discussed and implemented by various companies, yet neither of them suffices to satisfy the requirement of industrial development. In spite of the ineffective precursors, Flutter which is backed up by Google, drew attention and developers found it easier to use as well. The paper discusses the easy-to-use feature of Flutter along with the Flutter architecture in detail. The paper casts light on Flutter/Dart UI management and Flutter compiling. While comparing Flutter with other development platforms, topics such as native mobile applications, cross-

platform mobile application development, use and popularity are considered. A simple App is built using Flutter and other existing development platforms and some interesting differences were observed. Flutter had the lowest amount of code lines and files, that were needed in order to create the application. Native iOS had lower size of project files and app size while the native android had the highest number of files created and required lower amount of code lines than the iOS native. Android native had the longest developing time, followed by the iOS native and lastly the Flutter which had the shortest time. Run time CPU performance comparison is carried out and the result is that the highest CPU performance was more in Flutter iOS as compared to native iOS, while the Native Android had a better highest CPU performance than Flutter Android. But overall Flutter had a better mean CPU performance in both the OS so it beats the natives in both iOS and Android. Authors conclude the paper by stating that Flutter is a tool with a promising feature, if the community continues to grow in the direction that it is right now. Considering that Flutter's strong side is being a cross-platform solution, Flutter still performs well on a single application base if compared to native applications.

3. ANALYSIS

3.1 Introduction

In general, developing a mobile application is a complex and challenging task. There are many frameworks available to develop a mobile application. Android provides a native framework based on Java language and iOS provides a native framework based on Objective-C / Swift language.

However, to develop an application supporting both the OSs, we need to code in two different languages using two different frameworks. To help overcome this complexity, there exists mobile frameworks supporting both OS. These frameworks range from simple HTML based hybrid mobile application framework (which uses HTML for User Interface and JavaScript for application logic) to complex language specific framework (which do the heavy lifting of converting code to native code). Irrespective of their simplicity or complexity, these frameworks always have many disadvantages, one of the main drawbacks being their slow performance.

In this scenario, Flutter – a simple and high-performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system's canvas rather than through native framework.

3.1.1 Flutter

Flutter is a cross-platform software development framework that was presented by Google in 2015 and received its first release in May of 2017. Flutter nowadays has steadily grown and provided possibilities not only for iOS and Android mobile development but also for web and desktop applications as well.

Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

To be specific, Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming. Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive

programming) compare the widget's state (old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

Features of Flutter

Flutter framework offers the following features to developers −

- Modern and reactive framework.
- Uses Dart programming language and it is very easy to learn.
- Fast development.
- Beautiful and fluid user interfaces.
- Huge widget catalog.
- Runs same UI for multiple platforms.
- High performance application.

3.1.2 Flutter REST API

Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. REST API is a way of accessing web services in a simple and flexible way without having any processing.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

Today, most of the apps use remote data using APIs. So, this section will be the important part for those developers who want to make their carrier in Flutter.

Flutter provides http package to use http resources. The http package uses await and async features and provides many high-level methods such as read, get, post, put, head, and delete methods for sending and receiving data from remote locations. These methods simplify the development of REST-based mobile applications.

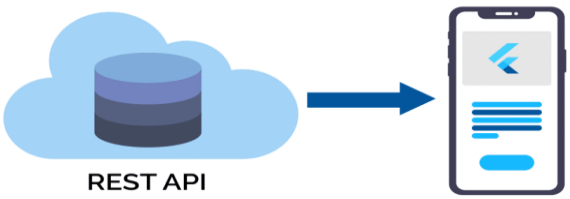

Fig 3.1 REST API

3.2 Software requirement specification

The best thing about using Flutter for creating cross-platform native mobile apps is the fact that you can build those on almost any OS.

Here are some System Requirements for Android Studio which is needed for running an Android simulator.

Windows:

- Microsoft® Windows® 7/8/10 (64-bit)
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based.
- **Disk Space**: 1.64 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter depends on these tools being available in your environment.
    - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
    - Git for Windows 2.x, with the Use Git from the Windows Command Prompt option.

If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

4. SYSTEM DESIGN

4.1 Flutter Framework

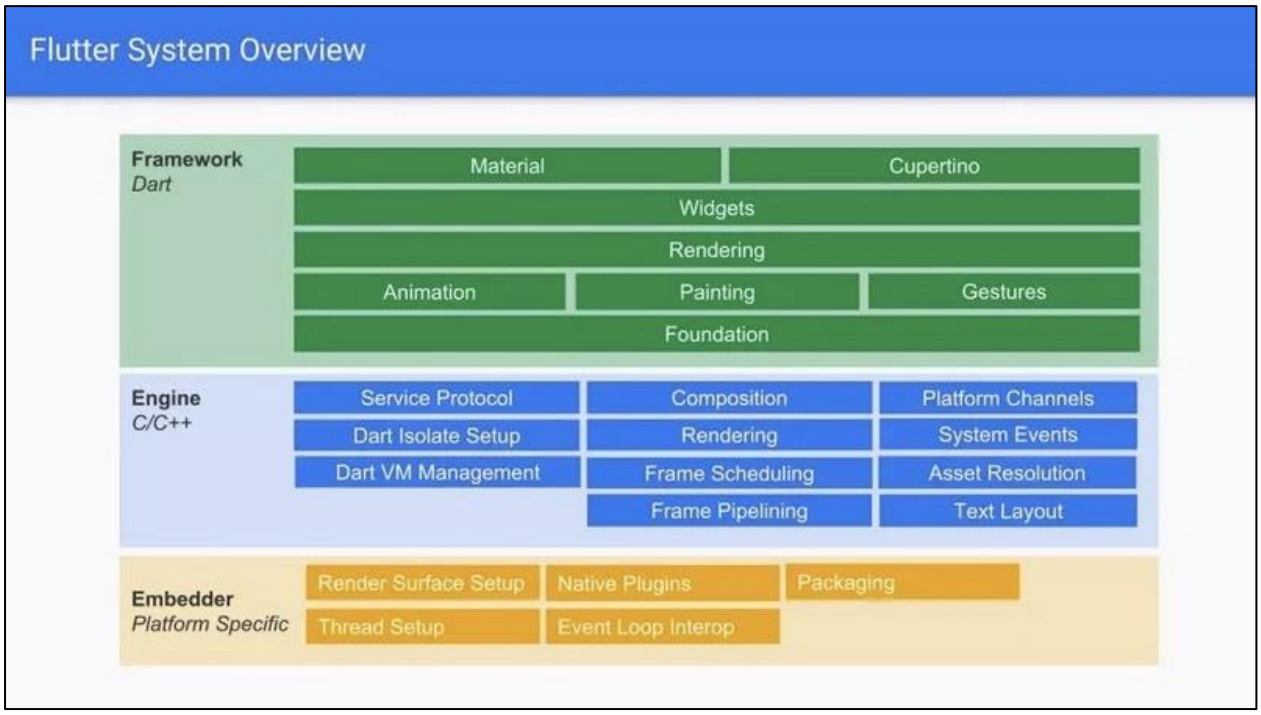

Fig 4.1 Flutter System Overview

Flutter is organized around layers. Each layer is built upon the previous.
From the diagram we can see the low-level part of Flutter is an Engine built in C++. It provides low-level rendering support using Google's Skia graphics library.

The high-level part of the diagram is the Framework written in Dart. It provides libraries to handle animation, gestures, rendering, widgets and more.

With all this layer the developer can do more with less code by using elements on the top or go down to customize some behavior of its app.

Everything is a widget

In Flutter, everything is a widget nested inside another widget. It comes with beautiful, customizable widgets and we can control the behavior of each widget and also styling becomes easy.

All the widget of a Flutter app forms a hierarchy where a widget is a composition of other widgets and each widget inherits properties from its parent.

4.1.1 Basic widgets

Flutter comes with a suite of powerful basic widgets, of which the following are commonly used:

Text:

The Text widget lets you create a run of styled text within your application.

Row, Column:

These flex widgets let you create flexible layouts in both the horizontal (Row) and vertical (Column) directions. The design of these objects is based on the web's flexbox layout model.

Scaffold:

Implements the basic Material Design visual layout structure. This class provides APIs for showing drawers, snack bars, and bottom sheets.

Container:

The Container widget lets you create a rectangular visual element.

Appbar:

A Material Design app bar. An app bar consists of a toolbar and potentially other widgets, such as a TabBar and a FlexibleSpaceBar.

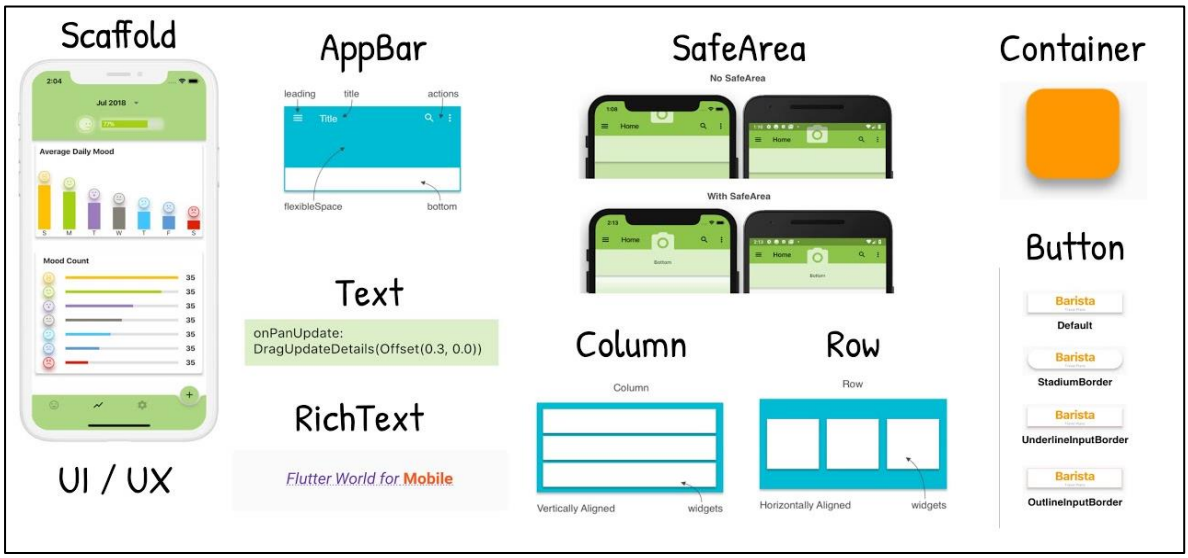

Fig 4.2 Basic Widgets

4.1.2 What is Material design?

Material is an adaptable design system, backed by open-source code, that helps developers easily build high-quality, digital experiences. From design guidelines to developer components, Material can help developers build products faster. Material design guidelines provide best practices and user interface design.

1. **App Structure and navigation**

   It consists of:

   - Appbar
     A Material Design app bar. An app bar consists of a toolbar and potentially other widgets, such as a TabBar and a FlexibleSpaceBar.

   - Drawer
     A Material Design panel that slides in horizontally from the edge of a Scaffold to show navigation links in an application.

   - MaterialApp
     A convenience widget that wraps a number of widgets that are commonly required for applications implementing Material Design.

   - Scaffold
     Implements the basic Material Design visual layout structure. This class provides APIs for showing drawers, snack bars, and bottom sheet.

2. **Buttons**

   It consists of:

   - DropdownButton

     Shows the currently selected item and an arrow that opens a menu for selecting another item.

   - ElevatedButton

     A Material Design elevated button. A filled button whose material elevates when pressed.

   - FloatingActionButton

     A floating action button is a circular icon button that hovers over content to promote a primary action in the application.

3. **Input and selections**

It consists of:

- Checkbox

  Checkboxes allow the user to select multiple options from a set. The Checkbox widget implements this component.

- Slider

  Sliders let users select from a range of values by moving the slider thumb.

- Text Field

  Touching a text field places the cursor and displays the keyboard. The Text Field widget implements this component.

- Date & Time Pickers

  Date pickers use a dialog window to select a single date on mobile.

4. **Layout**

It consists of:

- Divider

  A one logical pixel thick horizontal line, with padding on either side.

- ListTile

  A single fixed-height row that typically contains some text as well as a leading or trailing icon.

- Stepper

  A Material Design stepper widget that displays progress through a sequence of steps.

5. **Information displays**

It consists of:

- Card

  A Material Design card. A card has slightly rounded corners and a shadow

- Icon

  A Material Design icon.

- Image

A widget that displays an image.

- LinearProgressIndicator

  A material design linear progress indicator, also known as a progress bar.

- CircularProgressIndicator

  A material design circular progress indicator, which spins to indicate that the application is busy.

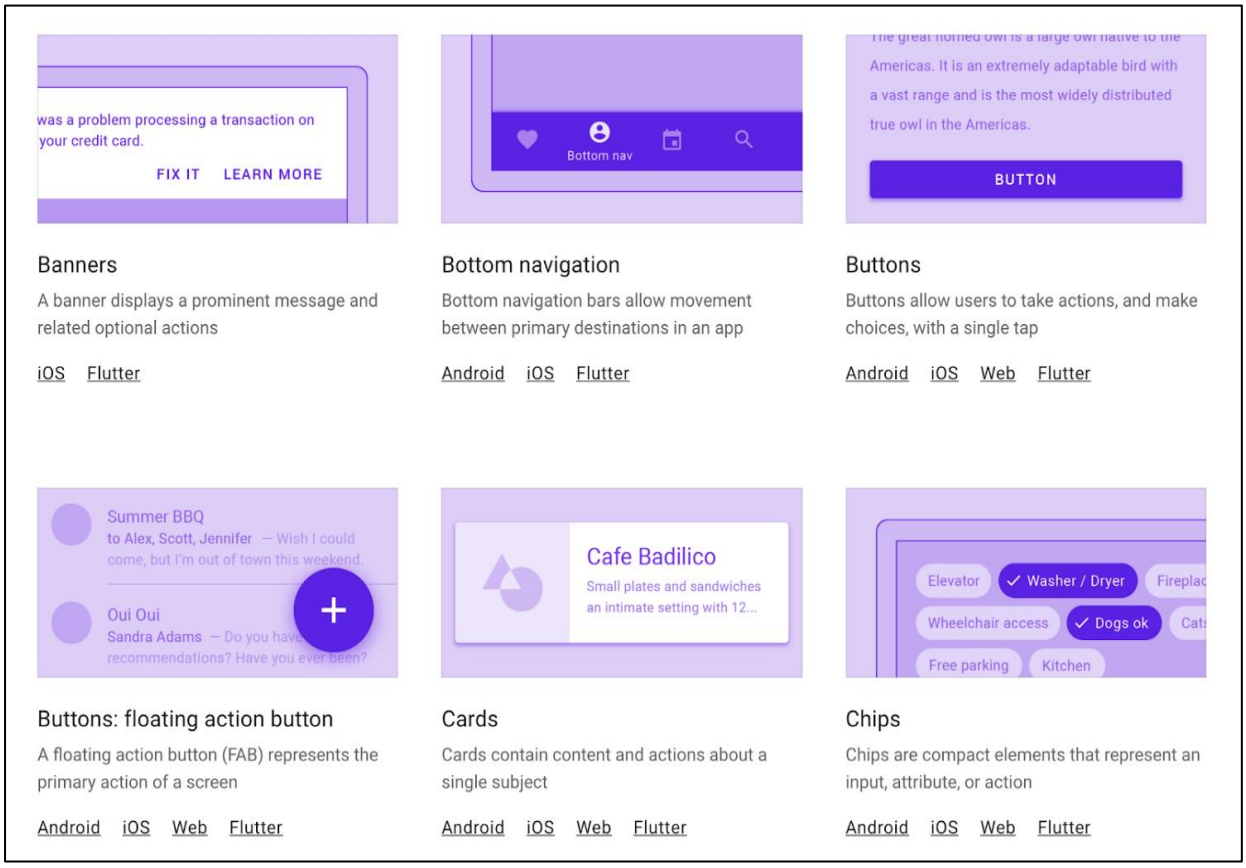

Fig 4.3 Flutter Material Design

4.2 Flutter Architecture

Flutter delivers the basic architecture that can be applied to application and manage its state easily. The architecture that is used in Flutter is called the Business Logic Component (BLOC). It is an event-state based approach that allows you to trigger events and handle state changes based on them. The BLOC is a good approach that separates your business logic from the user interface and oversees business logic key points by testing. The core ideas that were used for BLOC architecture are simplicity, scalability, and testability, and all these goals were definitely

achieved within the BLOC architecture. But this is a separate topic that we may cover at a later date.

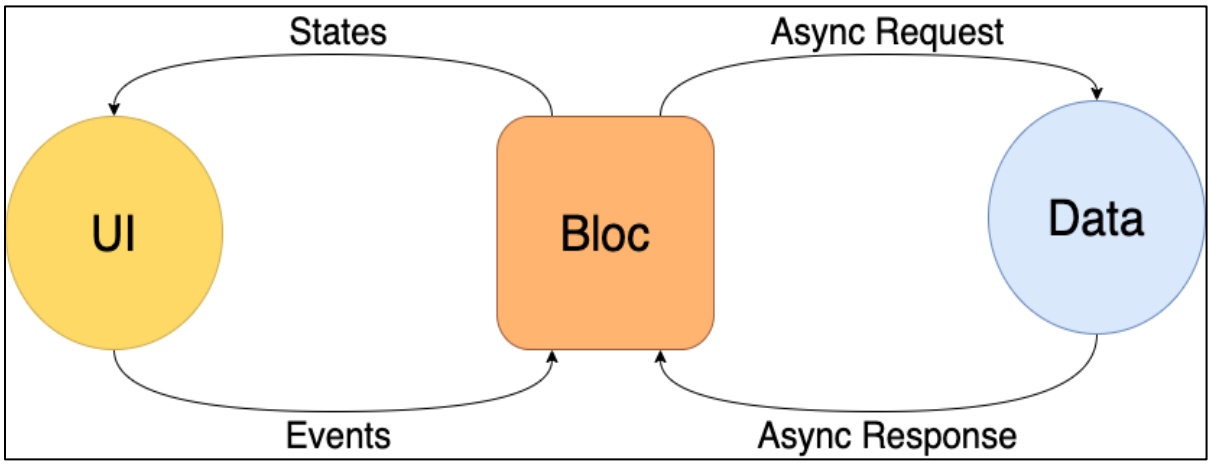

Fig 4.4 Flutter Architecture

5. DETAILED DESIGN

5.1 Process Flow

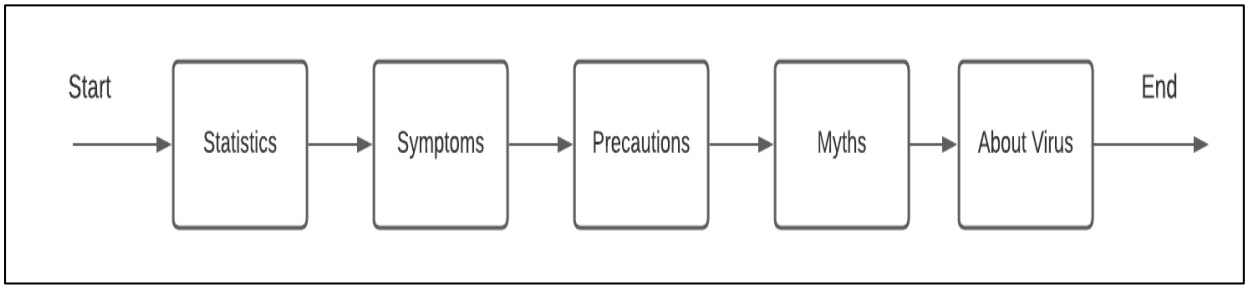

Fig 5.1 (a) Process Flow

The user is provided with many options to choose like View the Statistics, read about the Symptoms of COVID-19, read about Precautions to be taken, Myths regarding the virus and more information About the virus. The user can choose any of the options in any random way.

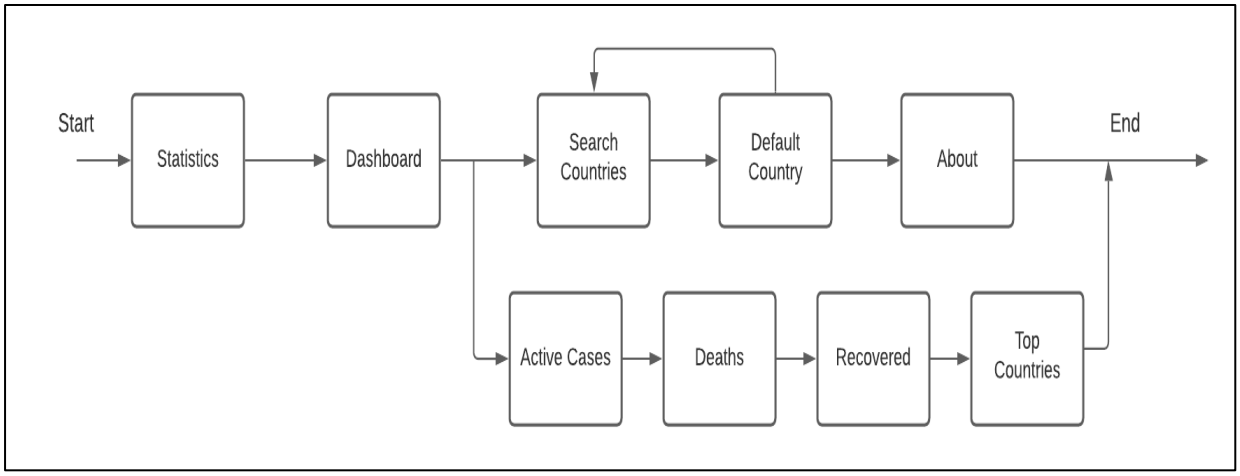

Fig 5.1 (b) Process Flow

Statistics screen consists of many components. The user is provided with the information on global statistics of COVID-19 spread. The user can view the number of active cases, deaths and recovered cases of any country from the dashboard. The dashboard also contains top countries with highest number of active cases. The Search feature allows the user to search any country of his choice to view information. The user can choose a default country. When the default country is to be selected it navigates to search screen. About screen provides brief information on what the app does to the user.

5.2 Screen Designs

The following screen are used in the App.

Fig 5.2 Home Screen

Fig 5.2 shows the main screen of the app. It includes different features like Statistics, Symptoms, Precautions, Myths and About virus represented in card view along with suitable headings and descriptions.

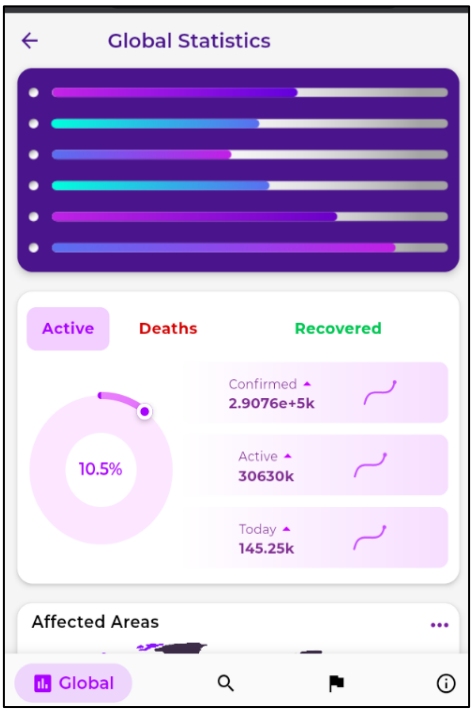

Fig 5.3 Global Statistics Screen

Fig 5.3 represents the global statistics screen which includes Bottom Navigation Bar as a dashboard which initially shows the active, deaths and recovered cases along with affected areas in the world map.

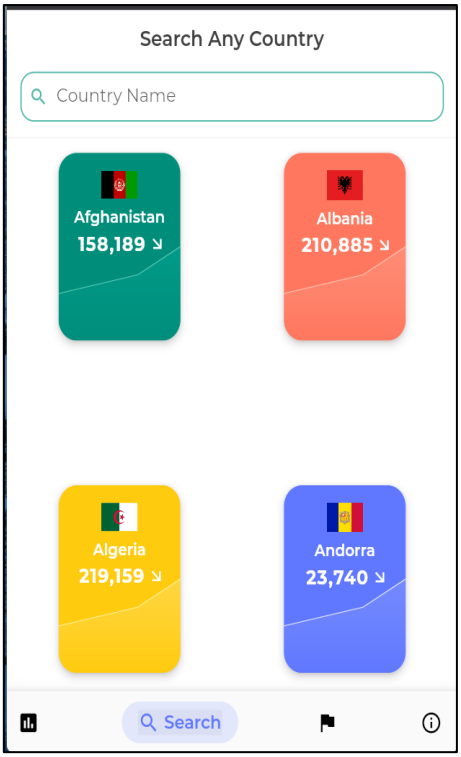

Fig 5.4 Card View of Countries

Fig 5.4 represents the screen that includes many card views for all the countries along with their flags and a search box is provided to find the desired country.

Fig 5.5 Default Country

Fig 5.5 represents the screen that helps the user to choose a default country, for immediate updates. It uses button to navigate the user to next screen.

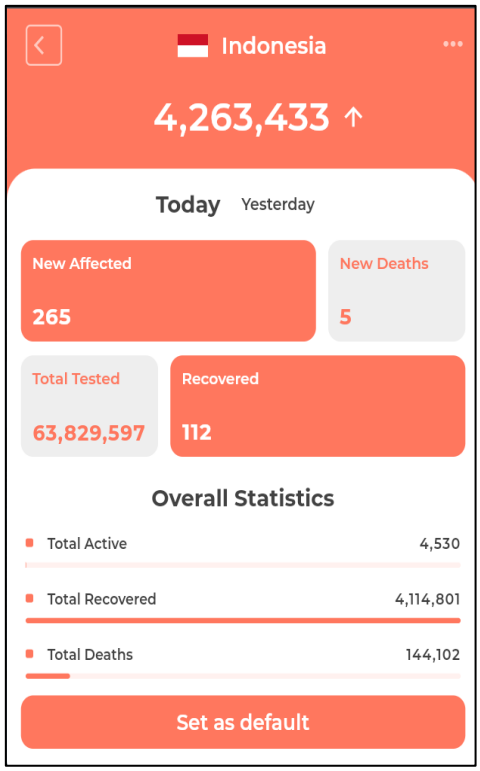

Fig 5.6 Country Statistics

Fig 5.6 is displayed when the user clicks on the button to choose a default country. It shows the user the present and the previous day overall statistics.

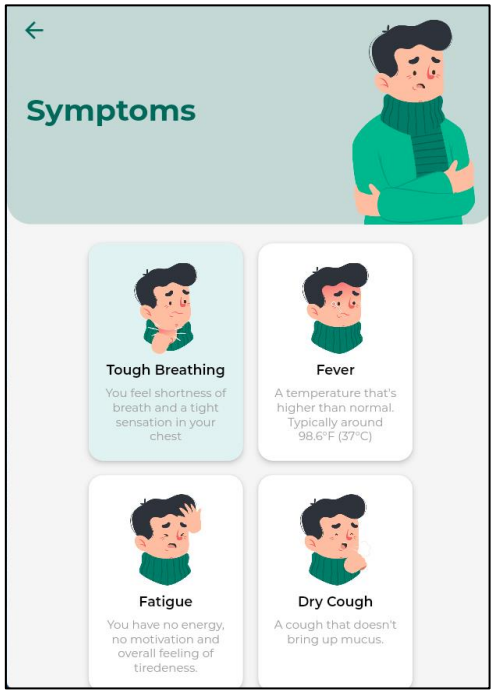

Fig 5.7 Symptoms

Fig 5.7 shows the screen that provides information on various symptoms of COVID-19 designed using card view and suitable images.

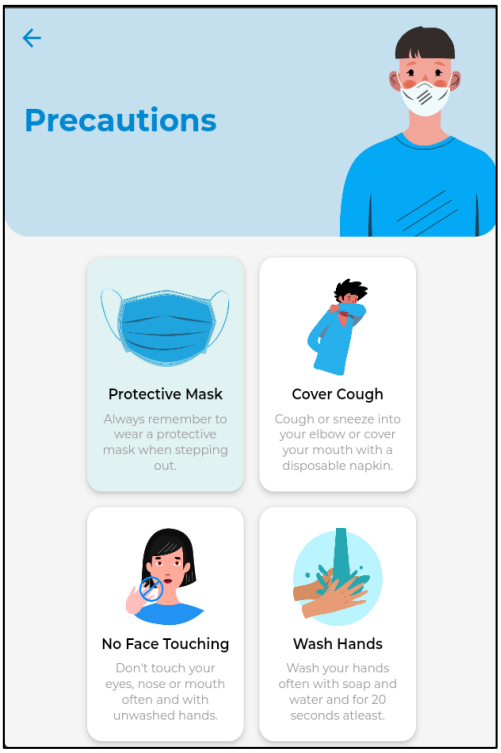

Fig 5.8 Precautions

Fig 5.8 shows the screen that provides information on various preventive measures to be taken to avoid the spread of COVID-19 and is designed using card view and suitable images.

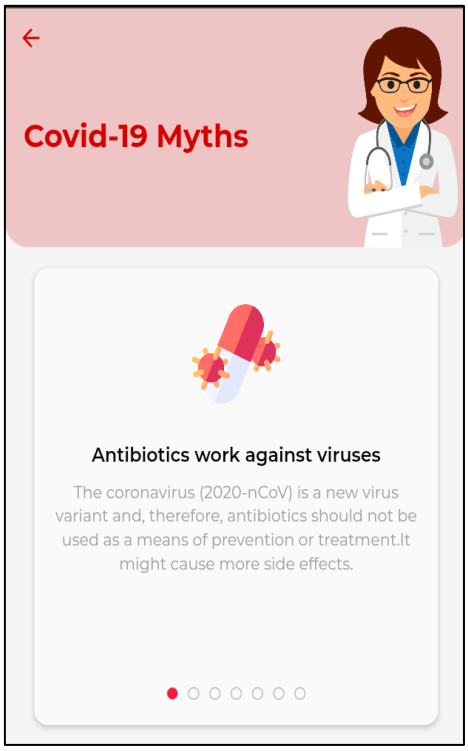

Fig 5.9 Covid-19 Myths

Fig 5.9 displays various myths related to COVID-19 and is designed using a feature to swipe cards, present along with description and suitable images.

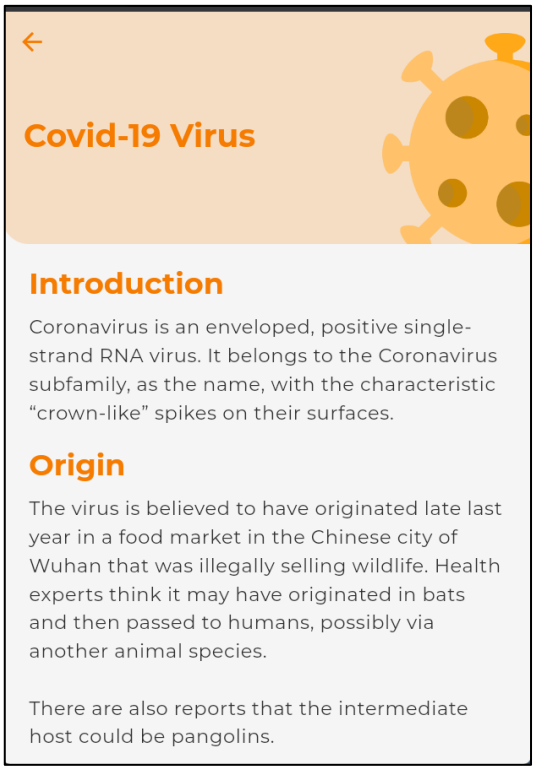

Fig 5.10 Covid-19 Information

Fig 5.10 displays information on what actually COVID-19 is and its history and its effects on human race. It also attempts to provide answers to general questions on the virus.

6. IMPLEMENTATION DETAILS

6.1 Implementation

```
class _HomeScreenState extends State<HomeScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        backgroundColor: Colors.lightBlue[50],
        elevation: 0,
        centerTitle: true,
        title: AutoSizeText(
          "Covid-19 Tracker App",
          style: TextStyle(
            fontSize: 20,
            fontFamily: "Montserrat",
            color: Colors.black,
            fontWeight: FontWeight.w600,
          ), // TextStyle
          minFontSize: 14,
          stepGranularity: 2,
          maxLines: 1,
```

The snippet is used to include basic design to the home page like AppBar, font, colors, and other necessary widgets.

```
static List<Map<String, dynamic>> categoryData = [
  {
    "imgLeft": 5.0,
    "imgBottom": 19.0,
    "imgHeight": 122.0,
    "imgPath": "assets/stats.png",
    "tabName": "Statistics",
    "tabDesc": "How many people are affected in the world",
    "color": Colors.deepPurpleAccent,
  },
  {
    "imgLeft": 15.0,
    "imgBottom": -8.0,
    "imgHeight": 150.0,
    "imgPath": "assets/symptoms/symptoms.png",
    "tabName": "Symptoms",
    "tabDesc": "Top Covid-19 symptoms",
    "color": Colors.teal[800],
  },
```

The snippet is used to display different features that the app provides on the home screen.

```
Function getPage(tabName, context) {
  switch (tabName) {
    case ("Symptoms"):
      return () => Navigator.of(context).push(MaterialPageRoute(
          builder: (context) =>
              SymptomsScreen(color: color, imgPath: imgPath))); // MaterialPageRoute
    case ("Precautions"):
      return () => Navigator.of(context).push(MaterialPageRoute(
          builder: (context) =>
              PrecautionsScreen(color: color, imgPath: imgPath))); // MaterialPageRoute
    case ("Myths"):
      return () => Navigator.of(context).push(MaterialPageRoute(
          builder: (context) => MythsScreen(color: color, imgPath: imgPath))); // MaterialPageRoute
    case ("Virus"):
      return () => Navigator.of(context).push(MaterialPageRoute(
          builder: (context) =>
              VirusDetailsScreen(color: color, imgPath: imgPath))); // MaterialPageRoute

    case ("Statistics"):
      return () => Navigator.of(context)
```

The snippet is used to switch the tabs and fetch the specified screens, when a particular card is selected.

```
class _CountryStatScreenState extends State<CountryStatScreen>{

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      body: SafeArea(
        child: CountryStatWidget(
          color: widget.color,
          onBackArrow: (){
            Navigator.of(context).pop();
          },
          countryCode: widget.countryCode,
          countryName: widget.countryName,
          totalCases: widget.totalCases,
          flagPath: widget.flagPath,
          isIncreasing: widget.isIncreasing,
        ), // CountryStatWidget
      ), // SafeArea
    ); // Scaffold
```

The snippet is implemented to display different statistics of the country and uses mainly Scaffold.

```
class _WorldStatScreenState extends State<WorldStatScreen> {
  PageController _controller;
  int selectedBottomBarIndex = 0;
  List<Widget> pages;
  List<BarItem> barItems;
  Future<bool> future;

  Future<bool> loadPreferences() async{
    SharedPreferences prefs=await SharedPreferences.getInstance();
    var jsonString=prefs.getString('defaultCountry');
    if(jsonString!=null){
      defaultCountry=DefaultCountry().fromJson(json.decode(jsonString));
      return true;
    }
    return false;
```

The snippet uses page controller to choose the initial default country.

```
class _DefaultCountryScreenState extends State<DefaultCountryScreen> {
  @override
  Widget build(BuildContext context) {
    if(defaultCountry.countryName==null){
      return Padding(
        padding: const EdgeInsets.fromLTRB(15, 350, 15, 0),
        child: Column(
          children: <Widget>[
            AutoSizeText(
              "No default country selected yet",
              style: TextStyle(
                fontSize: 18,
                fontFamily: "Montserrat",
                fontWeight: FontWeight.normal,
                color: Colors.black,
              ), // TextStyle
              maxFontSize: 18,
            ), // AutoSizeText
            SizedBox(height: 20),
```

The snippet implements the default country page and displays the message “No default country selected yet” before selection.

```
class _SymptomsScreenState extends State<SymptomsScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[100],
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        leading: IconButton(
            onPressed: () => Navigator.of(context).pop(),
            icon: Icon(
              Icons.arrow_back,
              color: widget.color,
              size: 28,
            )), // Icon // IconButton
        centerTitle: true,
        backgroundColor: Colors.transparent,
        elevation: 0,
      ), // AppBar
      body: Column(
        children: <Widget>[
```

The snippet is used to design the screen using AppBar and Column components to display the necessary information.

```
class _SymptomCardGridState extends State<SymptomCardGrid> {
  int selectedIndex = 0;
  final List<Map<String, String>> symptoms = const [
    {
      "symptom": "Tough Breathing",
      "desc":
          "You feel shortness of breath and a tight sensation in your chest",
      "imgPath": "assets/symptoms/sore_throat.png",
    },
    {
      "symptom": "Fever",
      "desc":
          "A temperature that's higher than normal.\nTypically around 98.6°F (37°C)",
      "imgPath": "assets/symptoms/high_fever.png",
    },
    {
      "symptom": "Fatigue",
      "desc":
          "You have no energy, no motivation and overall feeling of tiredeness.",
```

The snippet implements list to store the information that is to be displayed to the user.

```
class PrecautionsScreen extends StatefulWidget {
  final imgPath;
  final Color color;

  PrecautionsScreen({this.imgPath, this.color});

  @override
  _PrecautionsScreenState createState() => _PrecautionsScreenState();
}

class _PrecautionsScreenState extends State<PrecautionsScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[100],
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        leading: IconButton(
            onPressed: () => Navigator.of(context).pop(),
            icon: Icon(
```

The snippet to design the screen using AppBar and Column components to display the necessary information to prevent the spread of the virus.

```
class PrecautionCardGrid extends StatefulWidget {
  @override
  _PrecautionCardGridState createState() => _PrecautionCardGridState();
}

class _PrecautionCardGridState extends State<PrecautionCardGrid> {
  int selectedIndex = 0;
  final List<Map<String, String>> preventions = const [
    {
      "prevention": "Protective Mask",
      "desc": "Always remember to wear a protective mask when stepping out.",
      "imgPath": "assets/prevention/mask.png",
    },
    {
      "prevention": "Cover Cough",
      "desc":
          "Cough or sneeze into your elbow or cover your mouth with a disposable napkin."
      "imgPath": "assets/prevention/coughCover.png",
    },
    {
      "prevention": "No Face Touching",
```

The snippet implements list to store the information that is to be displayed to the user in an effective way.

```
class MythsScreen extends StatelessWidget {
  final controller = PageController(
    initialPage: 0,
  );

  final imgPath;

  final Color color;

  List<Map<String, String>> myths = [
    {
      "myth": "Antibiotics work against viruses",
      "desc": "The coronavirus (2020-nCoV) is a new virus variant and, therefore, " +
          "antibiotics should not be used as a means of prevention or treatment." +
          "It might cause more side effects.",
      "imgPath": "assets/myths/antibiotics.png",
    },
    {
      "myth": "Parcels from China spreads coronavirus",
      "desc":
```

The snippet initializes the first page using page controller and implements list to store the information that is to be displayed to the user.

6.2 Implementing REST API in Flutter

Flutter uses HTTP package, which provides advanced methods to perform operations. REST API uses simple http calls to communicate with JSON data because (i) it uses await & async features (ii) it provides various methods (iii) it provides class and http to perform web requests.

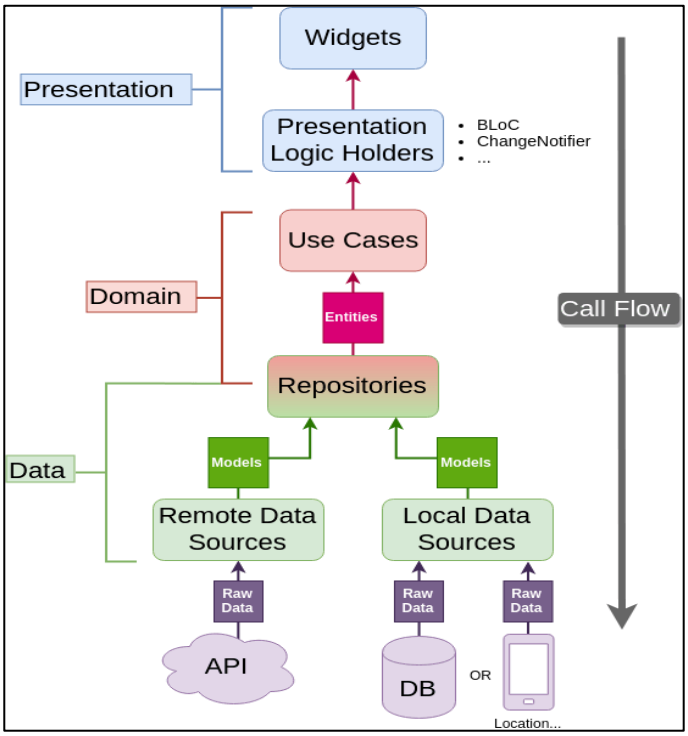

Fig 6.1 Flutter TDD Clean Architecture

The detail explanation of the core methods of the http package are as follows:

1. **Read:**
   This method is used to read or retrieve the representation of resources. It requests the specified URL by using the get method and returns the response as Future<String>.
2. **Get:**
   This method requests the specified URL from the get method and returns a response as Future<response>. Here, the response is a class, which holds the response information.
3. **Post:**
   This method is used to submit the data to the specified resources. It requests the specified URL by posting the given data and return a response as Future<response>.
4. **Put:**
   This method is utilized for update capabilities. It updates all the current representation of the target resource with the request payloads. This method requests the specified URL and returns a response as Future<response>.
5. **Head:**
   It is similar to the Get method, but without the response body.
6. **Delete:**
   This method is used to remove all the specified resources.

To fetch data from the internet, these necessary steps are to be followed:

Step 1: Install the latest http package and add it to the project.

Step 2: Next, make a network request by using the http package.

Step 3: Now, convert the response getting from network request into a custom Dart object.

Step 4: Now, fetch the data with Flutter. You can call the fetch method in the initState(). The following code explains how you can fetch the data.

Step 5: Finally, display the data. You can display the data by using the Future Builder widget. This widget can work easily with async data sources.

The application uses the API from open-source website https://disease.sh/v2/ . The features of disease.sh - Open Disease Data API is that we can build anything from console widgets to mobile applications, freely and is easy to use. The API provides data on current global outbreaks, including COVID-19 and Influenza.

```dart
import 'package:http/http.dart' as http;
import 'exceptions.dart';
import 'package:flutter/material.dart';

class ApiService {
  static const String apiKey = String.fromEnvironment('API_KEY',
      defaultValue: '54ca993b6e8040d883f8da1753bc5f3d');
  final String _newsUrl = "http://newsapi.org/v2/everything?";
  final String _statsUrl = "https://disease.sh/v2/";
  final String _query = "q=covid%2019%20vaccine";
  final String _limit = "pageSize=10&page=1";

  String get query => _query;
  String get newsUrl => _newsUrl;
  String get statsUrl => _statsUrl;
  String get limit => _limit;
}
```

The snippet shows the API Key along with the URL which provide the service.

```dart
      ApiService.apiKey;
  try {
    var response = await http.get(url);
    var json = jsonDecode(response.body);
    if (json['status'] == "ok") {
      return json;
    } else if (json['status'] == "error") {
      throw FetchDataException(json['code'] + json['message']);
    }
  } on SocketException {
    throw FetchDataException('No Internet connection');
  }
}
```

The snippet converts the http response to a Post. It checks for the necessary conditions and throws exceptions if the errors are found.

```
getStatsResponse(StateLocation stateLocation,
    {String code = "", bool yesterday = false}) async {
  String endpoint = _getStatsEndpoint(
      location: stateLocation, code: code, yesterday: yesterday);
  String url = _apiService.statsUrl + endpoint;
  try {
    var response = await http.get(url);
    if (response.statusCode == 200) {
      // ignore: non_constant_identifier_names
      var Json = json.decode(response.body);
      if (stateLocation == StateLocation.TOP_FIVE) {
        return Json.sublist(0, 6);
      }
      return Json;
    } else {
      throw FetchDataException("Failed to load stats");
    }
  } on SocketException {
    throw FetchDataException("No internet connection");
```

The snippet checks for the response received. Based on the status code received, appropriate actions are performed.

```
_getStatsEndpoint(
    {@required String code,
    bool yesterday,
    @required StateLocation location}) {
  if (location == StateLocation.GLOBAL) return "all?yesterday=$yesterday";
  String endpoint = "countries";

  if (location == StateLocation.SPECIFIC) {
    endpoint += "/" + code + "?strict=false&";
  } else if (location == StateLocation.TOP_FIVE) {
    endpoint += "?sort=cases&";
  } else if (location == StateLocation.ALL) {
    endpoint += "?";
  }
  return endpoint + "allowNull=false&yesterday=$yesterday";
}
```

The snippet uses the data received from the API to display the contents on the screen by filtering out the necessary information.

7. TESTING

Testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.

7.1 Common Flutter Errors

1) A Render-Flex Overflowed:

Render-Flex overflow is one of the most frequently encountered Flutter framework errors. When it happens, you'll see yellow & black stripes indicating the area of overflow in the app UI. The error often occurs when a Column or Row has a child widget that is not constrained in its size.

How to fix it?

One way to do it is to wrap the Column in an Expanded widget. Another way is to wrap the Column in a Flexible widget and specify a flex factor. In fact, the Expanded widget is equivalent to the Flexible widget with a flex factor of 1.0.

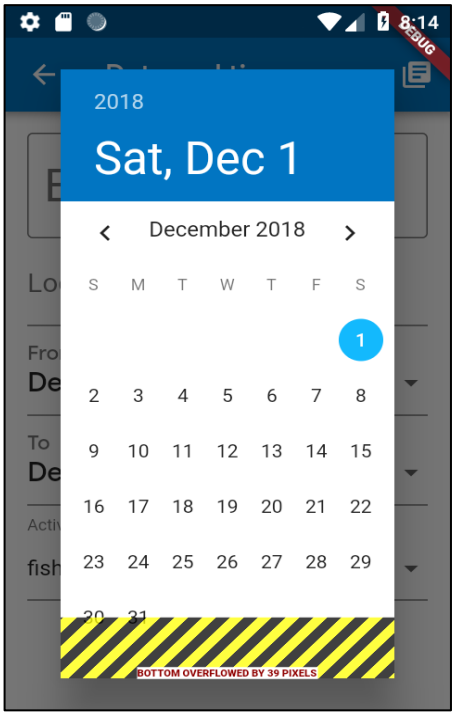

Fig 7.1 Render-Flex overflowed error

2) Render-Box was not laid out:

While this error is pretty common, it's often a side effect of a primary error occurring earlier in the rendering pipeline. the issue is related to violation of box constraints, and

it needs to be solved by providing more information to Flutter about how you'd like to constrain the widgets in question. The Render-Box was not laid out error is often caused by one of two other errors:

- 'Vertical viewport was given unbounded height'
- 'An Input-Decorator…cannot have an unbounded width'

3) Vertical viewport was given unbounded height:

This is another common layout error you could run into while creating a UI in Flutter app. The error is often caused when a ListView (or other kinds of scrollable widgets such as GridView) is placed inside a Column. A ListView takes all the vertical space available to it, unless it's constrained by its parent widget. However, a Column doesn't impose any constraint on its children's height by default. The combination of the two behaviors leads to the failure of determining the size of the ListView.

How to fix it?

To fix this error, specify how tall the ListView should be. To make it as tall as the remaining space in the Column, wrap it using an Expanded widget. Otherwise, specify an absolute height using a SizedBox widget or a relative height using a Flexible widget.

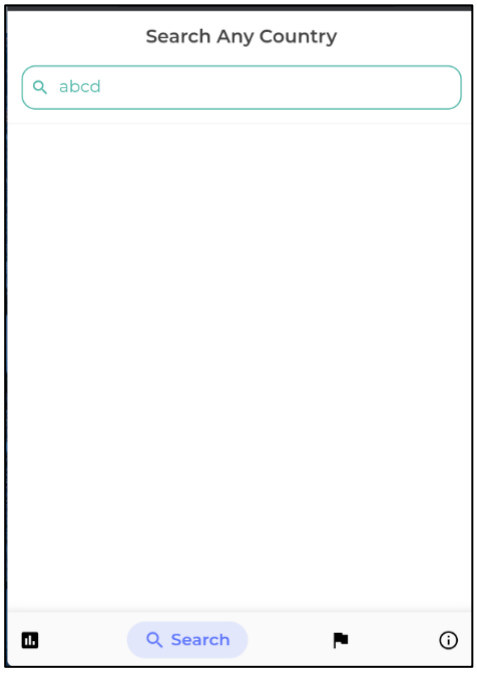

Fig 7.2 Incorrect Country Input

Fig 7.2 displays the scenario when the user enters a non-existing country name. As we expect, no country is displayed until the correct input is provided.

Fig 7.3 Correct Country Input

Fig 7.3 represents the scenario when the user has entered few letters in the search bar and the countries with name matching to the entered value is displayed to the user to be selected.

7.2 Handling Errors in REST API

The simplest way we handle errors is to respond with an appropriate status code. Here are some common response codes:

i. **400 Bad Request** – client sent an invalid request, such as lacking required request body or parameter
ii. **401 Unauthorized** – client failed to authenticate with the server
iii. **403 Forbidden** – client authenticated but does not have permission to access the requested resource
iv. **404 Not Found** – the requested resource does not exist
v. **412 Precondition Failed** – one or more conditions in the request header fields evaluated to false
vi. **500 Internal Server Error** – a generic error occurred on the server
vii. **503 Service Unavailable** – the requested service is not available

To minimize these kinds of responses to the client, we should diligently attempt to handle or catch internal errors and respond with other appropriate status codes wherever possible.

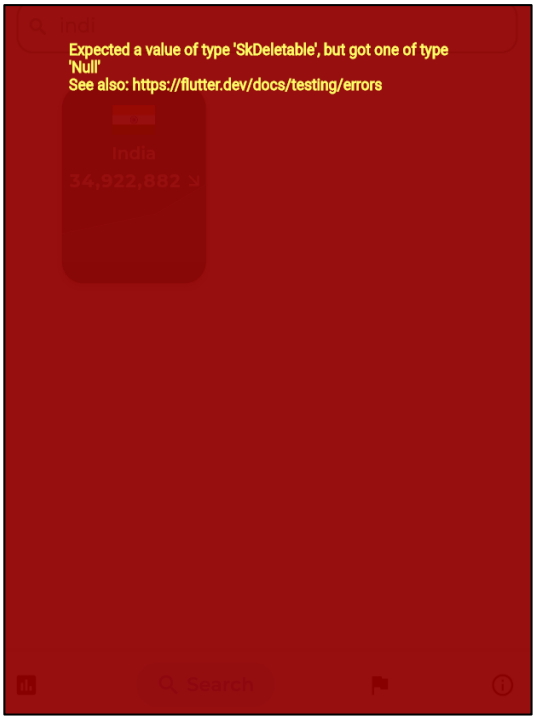

Fig 7.4 Wrong expected value

Fig 7.4 displays a case when the value expected is different from the one that is received from the API (generally null value). Best practice is to check if the variable is null before generating JSON.

```
    ApiService.apiKey;
  try {
    var response = await http.get(url);
    var json = jsonDecode(response.body);
    if (json['status'] == "ok") {
      return json;
    } else if (json['status'] == "error") {
      throw FetchDataException(json['code'] + json['message']);
    }
  } on SocketException {
    throw FetchDataException('No Internet connection');
  }
}
```

The snippet checks if the status code is “OK”, if so then the json file is returned, else if the status code is “error” then the Fetch-Data Exception is thrown.

```
getStatsResponse(StateLocation stateLocation,
    {String code = "", bool yesterday = false}) async {
  String endpoint = _getStatsEndpoint(
      location: stateLocation, code: code, yesterday: yesterday);
  String url = _apiService.statsUrl + endpoint;
  try {
    var response = await http.get(url);
    if (response.statusCode == 200) {
      // ignore: non_constant_identifier_names
      var Json = json.decode(response.body);
      if (stateLocation == StateLocation.TOP_FIVE) {
        return Json.sublist(0, 6);
      }
      return Json;
    } else {
      throw FetchDataException("Failed to load stats");
    }
  } on SocketException {
    throw FetchDataException("No internet connection");
```

The snippet is trying to find the specific information from the file received using endpoints. It checks if the status code is 200, if so then it decodes the json file, else it will throw FetchDataException.

```
class AppException implements Exception {
  final _message;
  final _status;

  AppException([
    this._message,
    this._status,
  ]);

  String toString() {
    return "$_message";
  }
}

class FetchDataException extends AppException {
  FetchDataException([String message]) : super(message);
}
```

The snippet shows how the exceptions that are thrown is handled.

8. RESULTS

The following are the screens of the App along with its features.

Fig 8.1(a) Main Screen (Mobile View)

Fig 8.1(b) Main Screen (Mobile View)

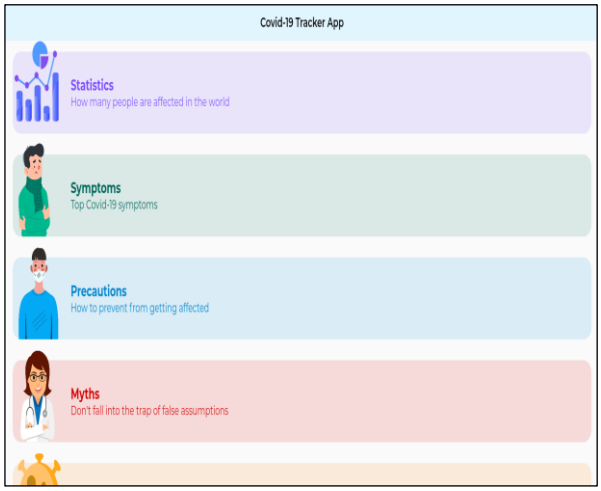

Fig 8.1(c) Main Screen (Web View)

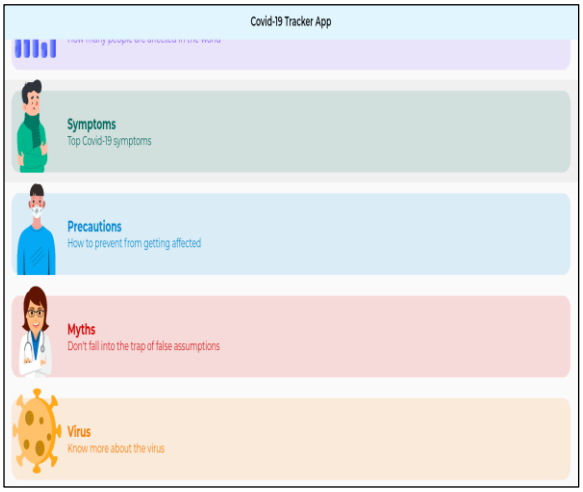

Fig 8.1(d) Main Screen (Web View)

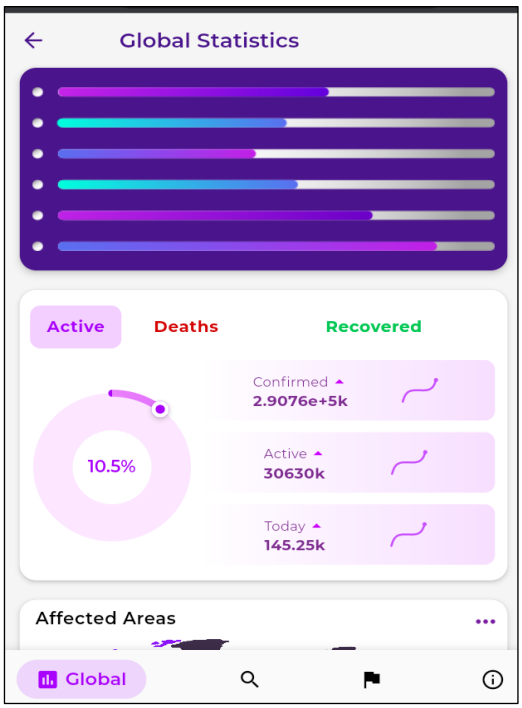

Fig 8.2 Statistics-Active

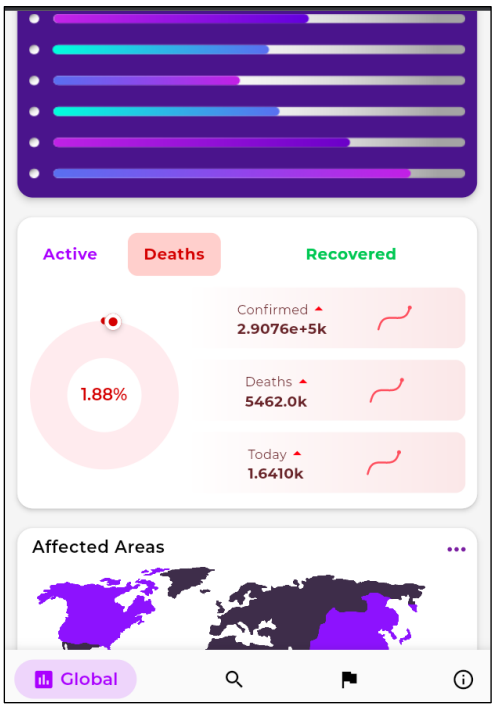

Fig 8.3 Statistics-Deaths

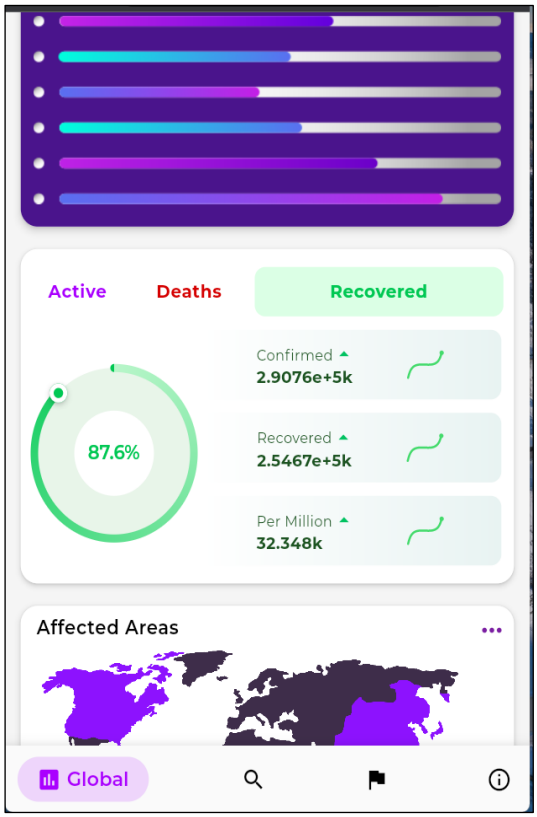

Fig 8.4 Statistics-Recovered

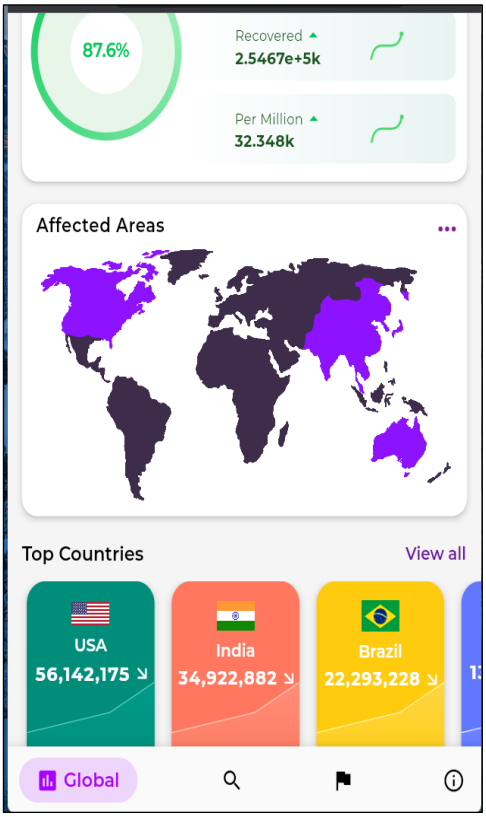

Fig 8.5 Statistics of World

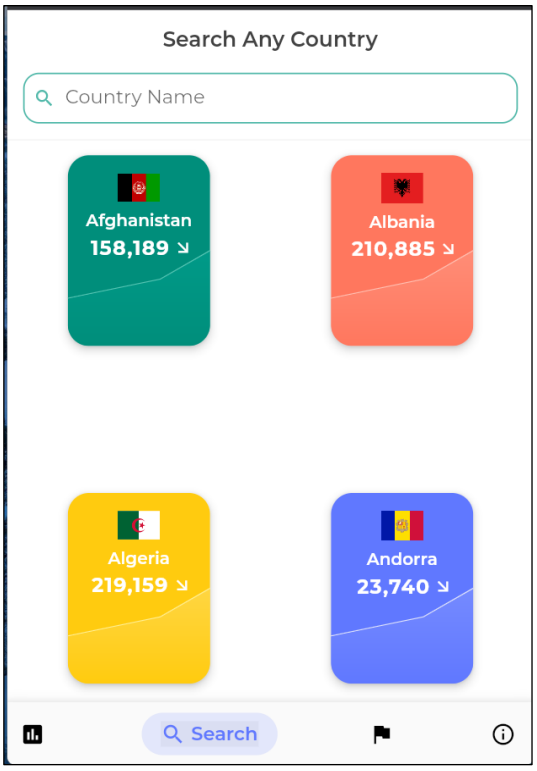

Fig 8.6 Statistics-Search Country

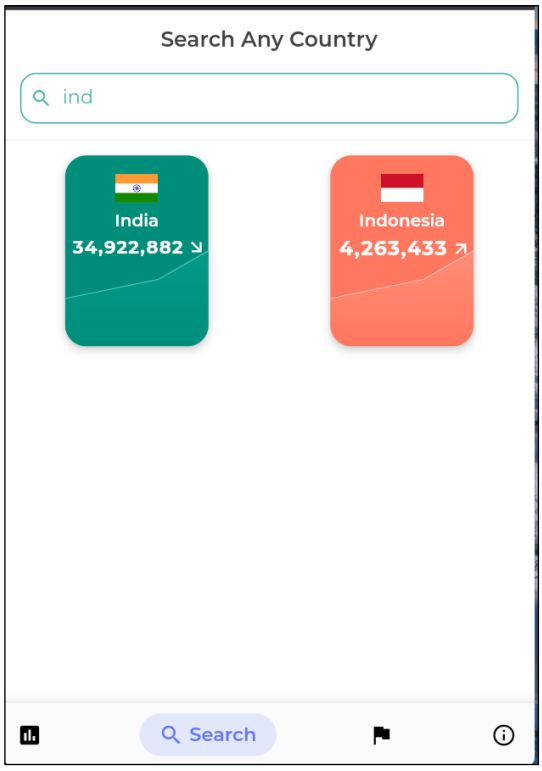

Fig 8.7 Result of search

Fig 8.8 Choose Default Country

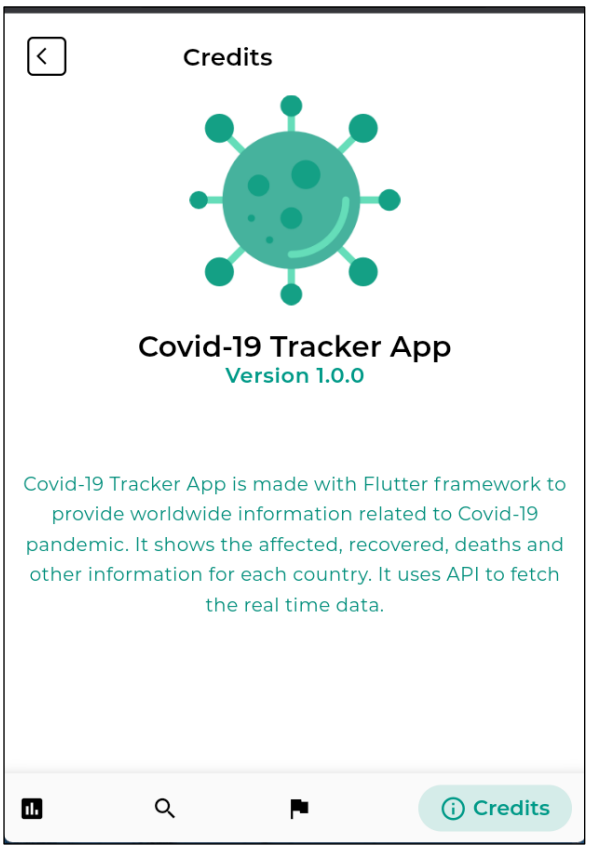

Fig 8.9 About the App

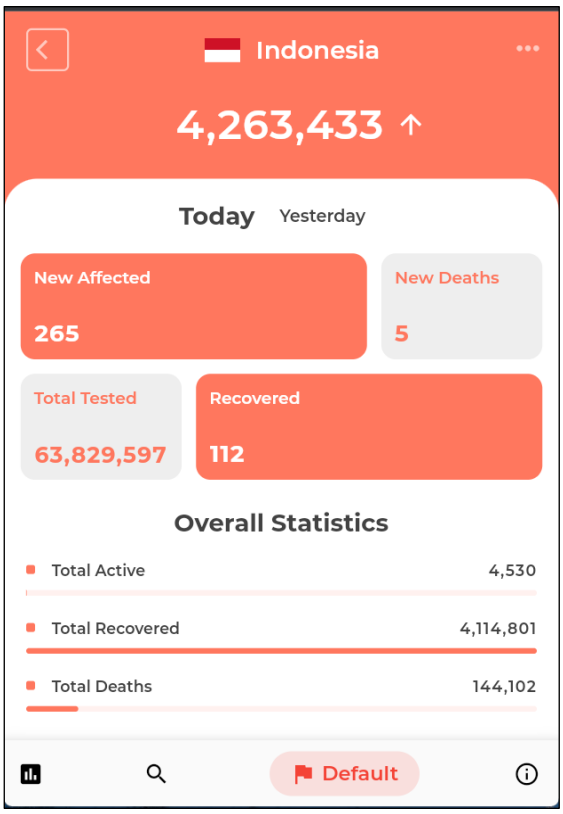

Fig 8.10 Country Statistics

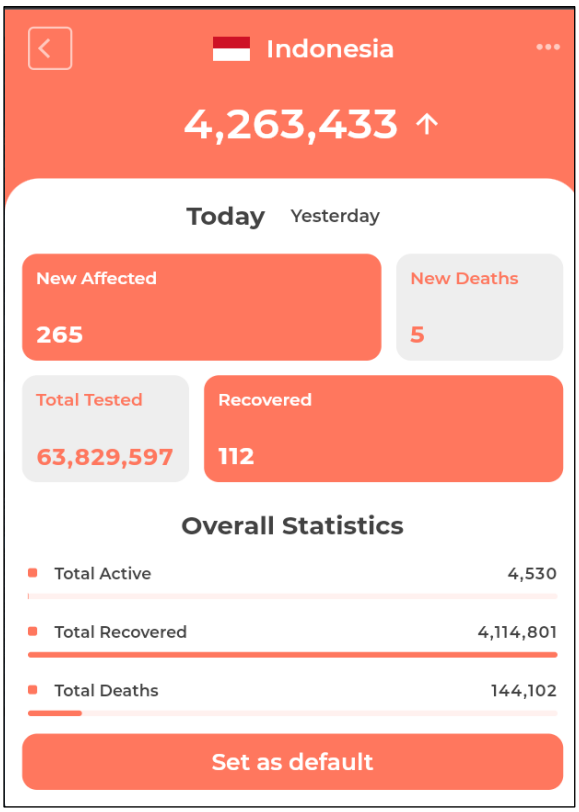

Fig 8.11 Set Default Country

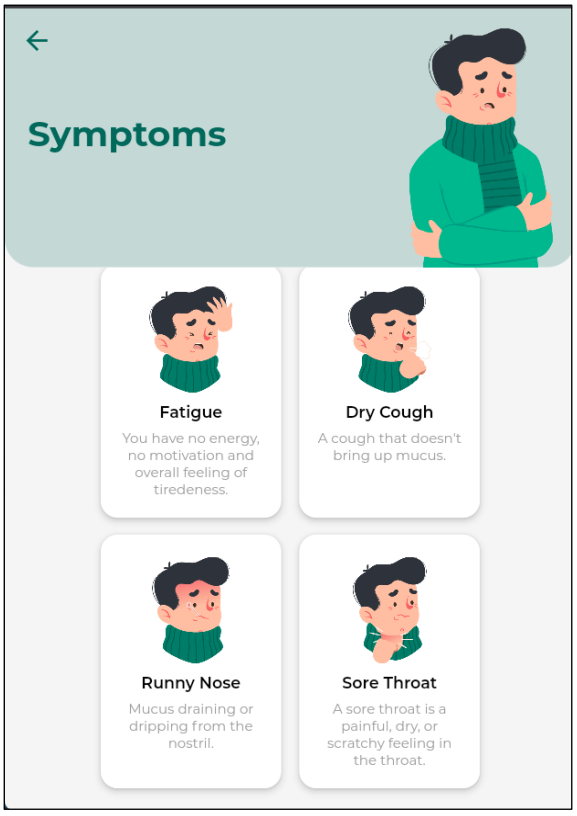

Fig 8.12 (a) Symptoms

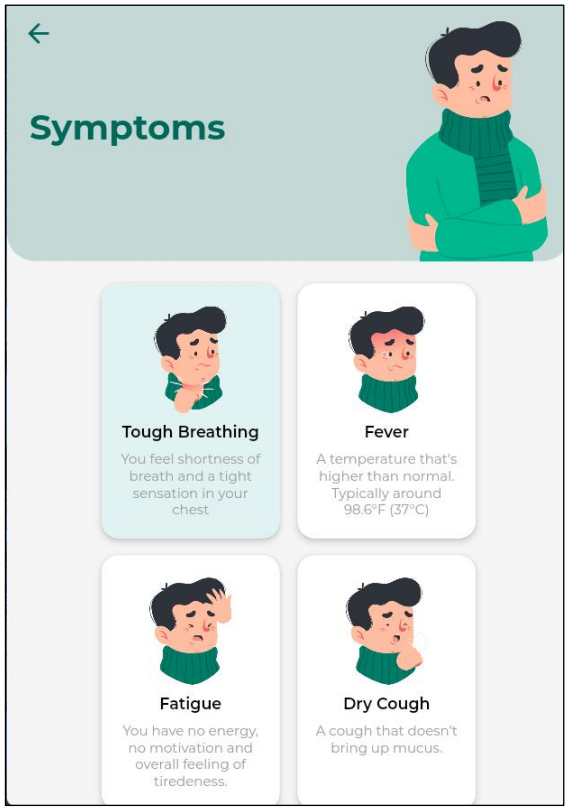

Fig 8.12 (b) Symptoms

Fig 8.13 (a) Precautions

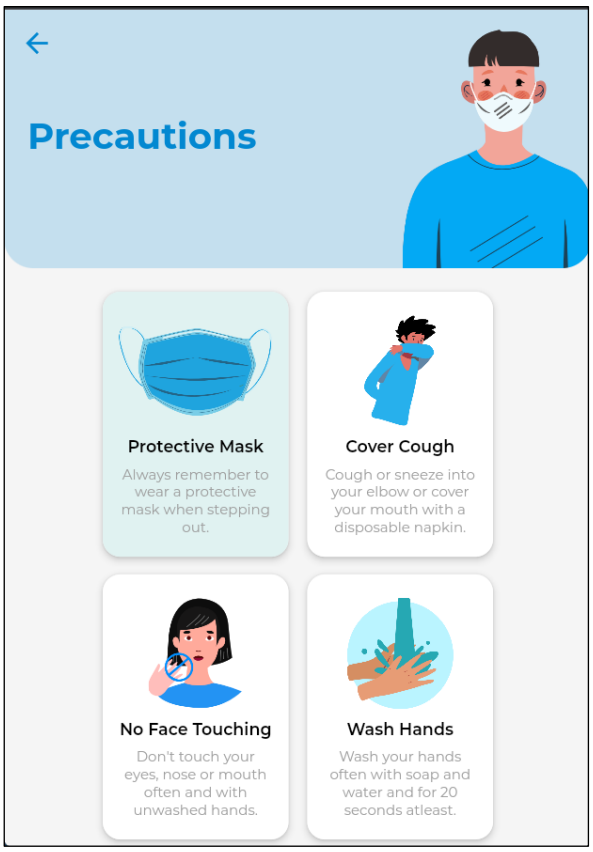

Fig 8.13 (b) Precautions

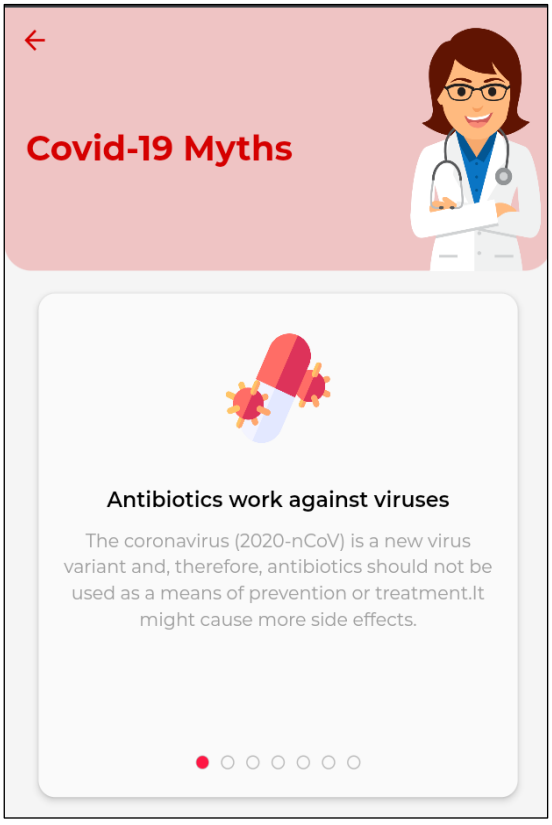

Fig 8.14 (a) Myths

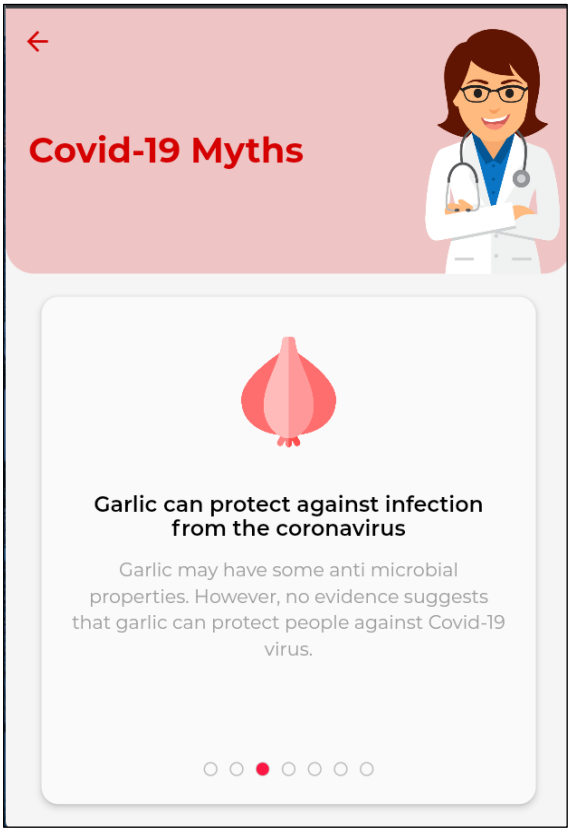

Fig 8.14 (b) Myths

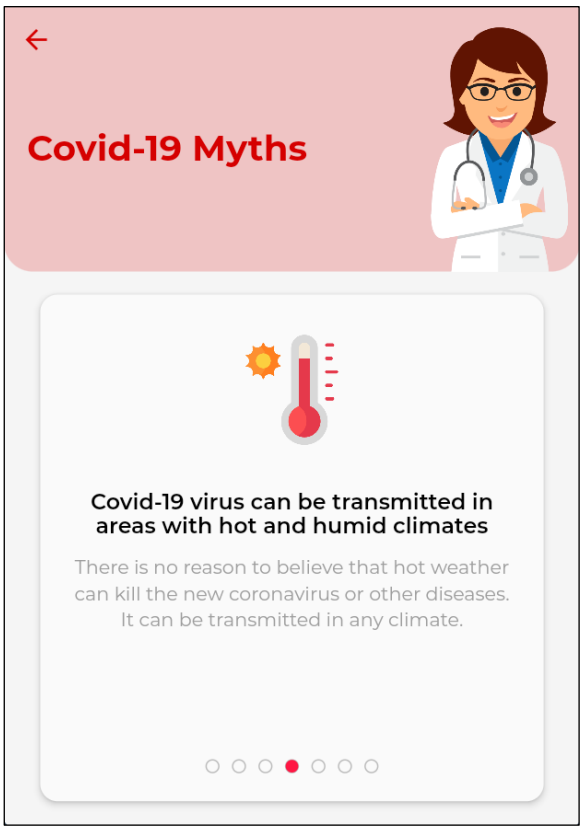

Fig 8.14 (c) Myths

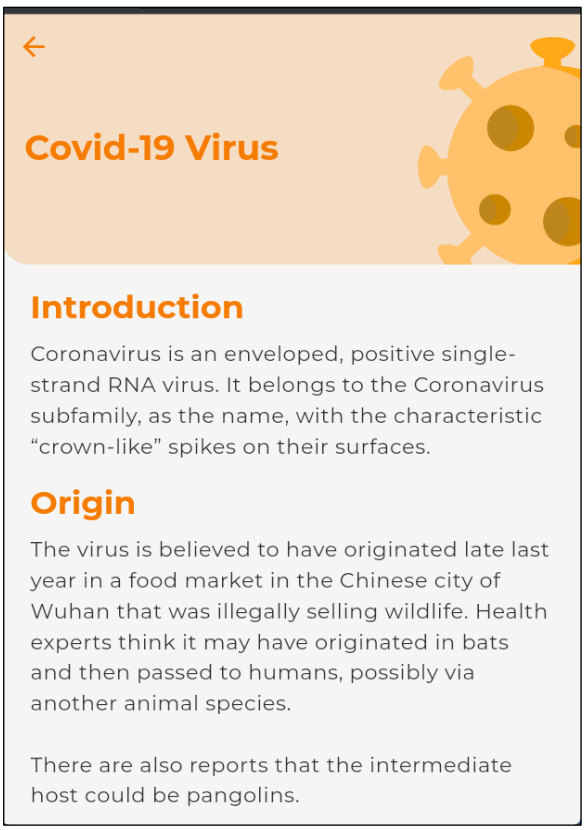

Fig 8.15 (a) About the Virus

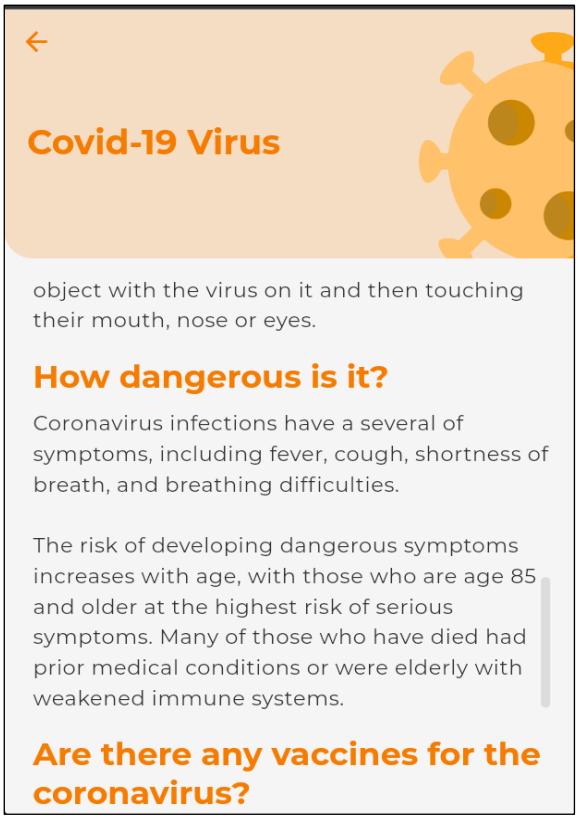

Fig 8.15 (b) About the Virus

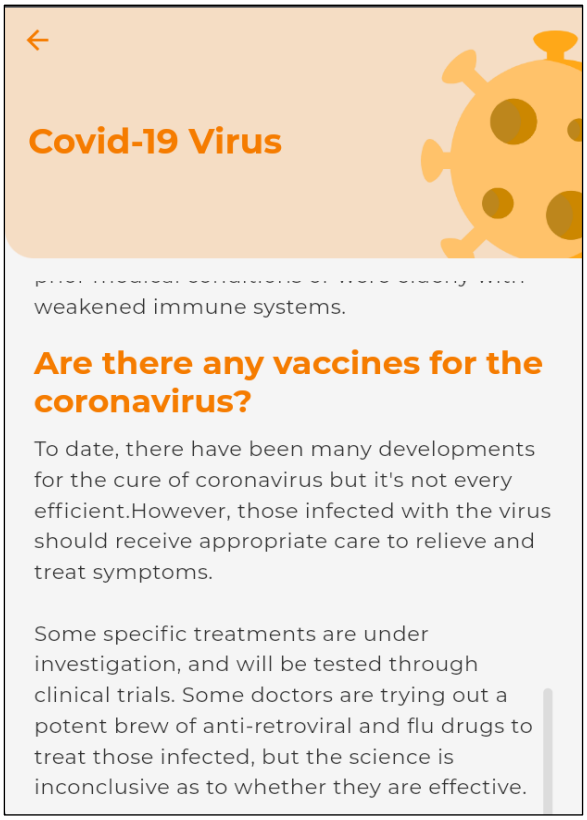

Fig 8.15 (c) About the Virus

9. CONCLUSION AND FUTURE WORK

9.1 Conclusion

The app provides efficient means to find information related to COVID-19 by displaying most necessary details to the user and is beginner friendly. It successfully provides the digital protection of the society, creates public awareness, and helps users to be up to date with essential details.

9.2 Future Work

The app can be extended to have many new features such as:

- Use APIs to provide COVID-19 news of different countries and default country
- Make the app accurate.
- Provide geo-locations.
- Make it contact COVID-19 centers
- Allow the users to book vaccination slots.
- Take basic self-assessment tests by users.
- Provide information on nearby containment zones.

10. REFERENCES

- https://flutter.dev/learn
- https://docs.flutter.dev/reference/tutorials
- https://www.tutorialspoint.com/flutter/index.htm
- https://www.geeksforgeeks.org/flutter-material-design/
- https://www.ibm.com/cloud/learn/rest-apis
- https://www.raywenderlich.com/24499516-getting-started-with-flutter
- https://www.w3adda.com/flutter-tutorial
- https://fluttertutorial.in/
- https://www.irjmets.com/
- https://www.ijcrt.org/papers/
- https://ijesc.org/
- https://ieeexplore.ieee.org/document/9071367
- https://ieeexplore.ieee.org/document/9429595

REFERENCE PAPERS

1) Ragda Mamoun, Mohamed Nasor, Sahar H. Abulikailik. **Design and Development of Mobile Healthcare Application Prototype Using Flutter**. 2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)
2) Shady Boukhary, Eduardo Colmenares. **A Clean Approach to Flutter Development through the Flutter Clean Architecture Package**. 2019 International Conference on Computational Science and Computational Intelligence (CSCI)
3) Madhuram. M, Ashu Kumar, Pandyamanian. **Cross Platform Development using Flutter**. International Journal of Engineering Science and Computing, April 2019
4) Aakanksha Tashildar, Nisha Shah, Rushabh Gala, Trishul Giri, Pranali Chavhan **Application Development using Flutter**. International Research Journal of Modernization in Engineering Technology and Science 2020
5) Achal Agrawal, Amit Agrawal, Rahul Arya, Hardik Jain, Jyoti Manoorkar. **Comparison of Flutter with Other Development Platforms**. International Journal of Creative Research Thoughts 2021