- Title Page
- Declaration
- Certificate
- Acknowledgement

TABLE OF CONTENTS

Abstract

We come across various challenges when it comes to daily chores, most important of which is cooking as it is one of the most fundamental parts of your life. However, many people find it challenging to learn it in the right way. From various ingredients to correct timing, cooking is one skill that each of us should know.

As a part of this project, we have developed an app with voice interaction which enables people from different part of world to learn different recipes at home with an easy to use voice controlled interface using Flutter. Flutter, being an SDK (software development kit) helps and simplifies the process as it is easy to use. It promises the ability to build native applications on iOS and Android that achieve native performance.

Using DART language to program flutter and voice interface makes the app different from other cooking applications. The language only requires basics of Object-oriented programming as pre-requisite and thus is easy to learn.

CHAPTER 1
INTRODUCTION

1.1 Introduction

App Development is done using Flutter. The software used for writing the code is Android Studio which has a built-in feature of Emulator for testing the code of the app.

What is Flutter?

- Mobile UI framework for creating native apps for both iOS and Android.
- Single code-base (dart) means we only have to write our app once for multiple devices

Why use Flutter?

- Only 1 code base
- Good layout methodology borrowed from responsive web
- Smooth and quick experience when running apps
- Works well with Firebase as a backend
- Uses Dart, which is a really easy language to pick up
- Uses Material design out-of-the-box
- Great Docs and guides on the Flutter website

1.2 Necessity

In current times, it very essential for each of us to know cooking. All of us know this fact but only few work on it. But while living alone we realise the importance of home-cooked food.

Some people who desire to learn cooking are mostly unable to find someone who could teach them how to cook healthy and edible food.

Though we can access vast recipes using our smartphones, it is not advisable to touch the screen while cooking food as the screen has a lot of bacteria. This can be harmful to us if the bacteria enters our body through the food.

So we have made a mobile application which, using the text-to-speech feature, would dictate the steps of cooking. Once the user selects the recipe, there won't be any need to touch the screen of phone while cooking. The user can thus interact with the app using the voice feature.

Aligning with the Prime Minister's focus on 'Atmanirbhar Bharat', this app could truly help a person to be self-sufficient when it comes to making one's own food.

1.3 Objectives

1. Using a native SDK to develop a mobile application for cooking which contains voice interaction.

2. To be able to make a user friendly interface and to learn use of the flutter platform
3. To learn and implement the DART programming language unlike other native apps which use JAVA script.
4. To Implement the Flutter TTS plug-in as a part of our app development.

1.4 Organization

Flutter is an open-source project, with contributions from Google and the community. Developers inside and outside of Google use Flutter to build beautiful natively-compiled apps for iOS and Android.
Whereas on our end, the project is supported by the School Of Electrical Engineering, MITWPU.

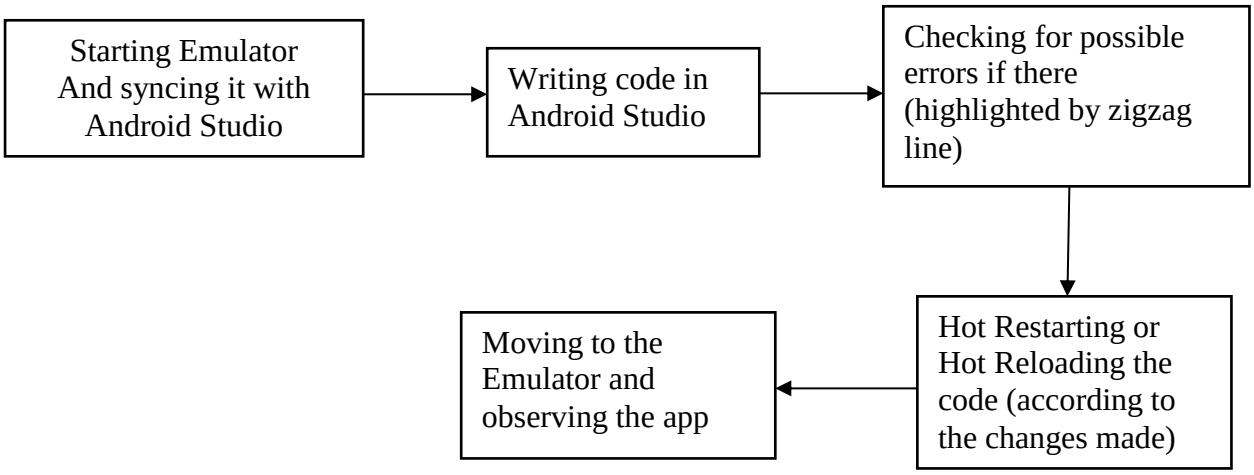
CHAPTER 2
System Model and Literature Survey

2.1 Introduction

Flutter is a cross-platform framework for developing apps that can run on Android, iOS and Google's next-generation operating system Fuschia. It was created by the Google Chrome browser team and publicly released in 2016. The reason for creating Flutter was to evaluate if rendering could be more efficient if traditional layout models were ignored. This resulted in a different way of thinking about layout.

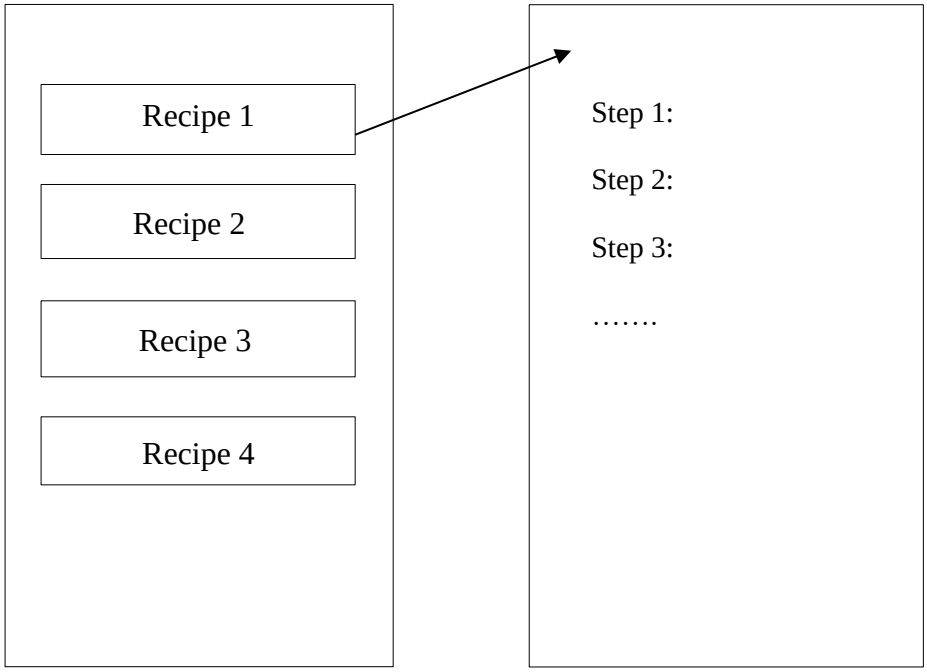
In the times of the Corona Virus outbreak, we all realised how important it is for a person to be self-sufficient. It is now clear that each one of us should know the basic skill of cooking food.
Cooking requires a lot of focus both for our safety as well as regarding the food that we are making. Ingredients, timing, caution, vigilance, cleanliness and taste are some of the important parameters that require some training before-hand.

2.2 Block Diagram



- Go to Tools in Android Studio, then to the AVD manager. From that we can start the AVD (Android Virtual Device) or Emulator. Once the AVD starts and the Android Studio syncs with the device, we can run the code.
- Code has to be written in the main.dart file using the language DART and dependencies can be added according to the requirement.
- As soon as a change is made which might be not compatible with the language (i.e an error), there is a zigzag line visible under the file where the error is present.
- Hot Restart is one way to start the app. If there are small changes made to the code, Hot Reload can be used. Hot reload feature is much faster in executing the changes.
- As soon as the building of app is finished, we can view the changes in the app on the Emulator.

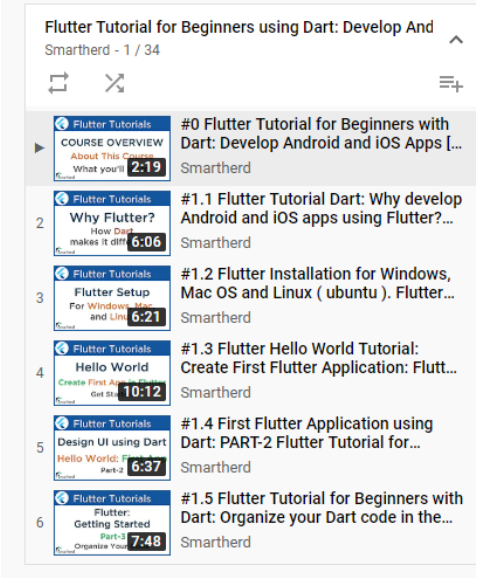
The layout of the app can be simplified as follows:

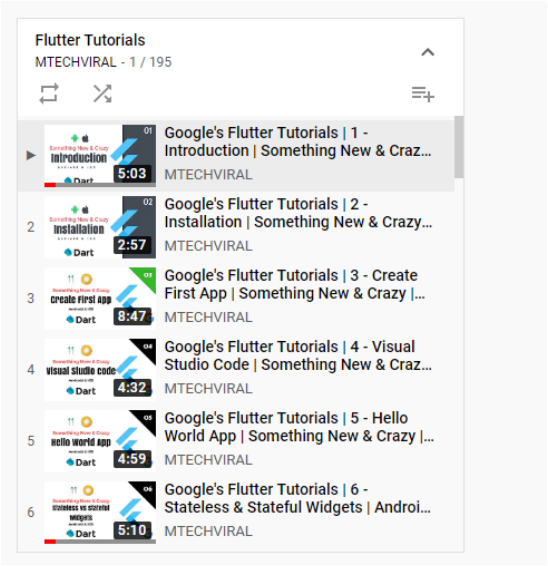


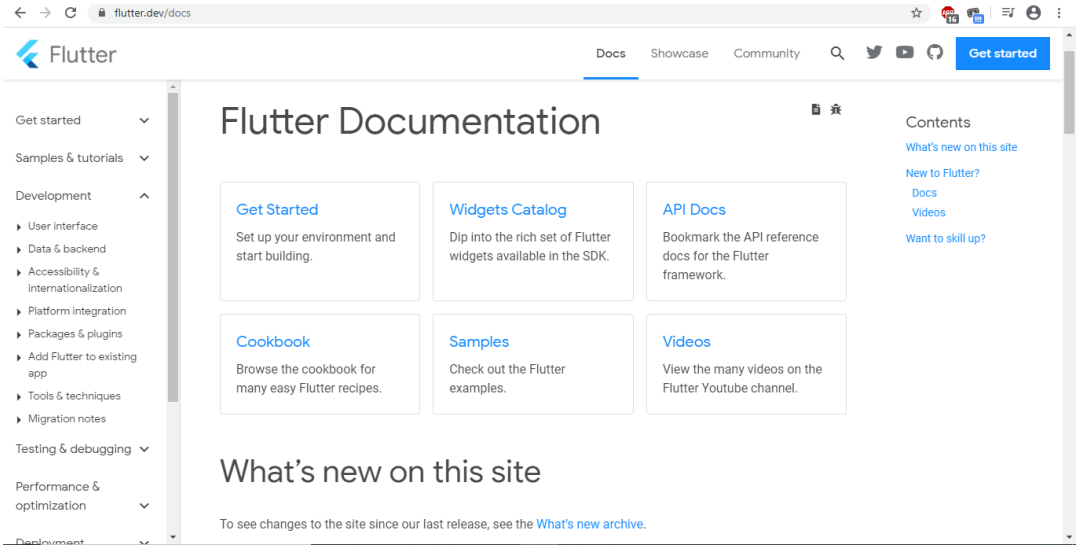
The app would contain the various recipes. By clicking on any one of the recipe, the user would be directed to another screen. This screen would contain the steps for cooking the respective dish. It would also contain a Floating Action Button to allow the user to enable the Text-to-speech plug-in of Flutter. This plug-in would convert the steps to speech and convey them through the speaker of the phone.

2.3 Literature Survey

As a part of Literature Survey, we saw many videos on Internet to gather as much information possible for our project. Since Flutter is quite new and still under development, the main source of our literature were the Youtube videos and documents available on the official site of Flutter.







How experienced a programmer/developer need to be to use Flutter?
Flutter is approachable to programmers familiar with object-oriented concepts (classes, methods, variables, etc) and imperative programming concepts (loops, conditionals, etc).
No prior experience is required in order to learn and use Flutter.

What kinds of apps can be built with Flutter?
Flutter is optimized for 2D mobile apps that want to run on both Android and iOS. Flutter is also great for interactive apps that we want to run on our web pages or on the desktop.
Apps that need to deliver brand-first designs are particularly well suited for Flutter. However, apps that need to look like stock platform apps can also be built with Flutter.
We can build full-featured apps with Flutter, including camera, geolocation, network, storage, 3rd-party SDKs, and more.

What makes Flutter unique?
Flutter is different than most other options for building mobile apps because Flutter uses neither WebView nor the OEM widgets that shipped with the device. Instead, Flutter uses its own high-performance rendering engine to draw widgets.
In addition, Flutter is different because it only has a thin layer of C/C++ code. Flutter implements most of its system (compositing, gestures, animation, framework, widgets, etc) in Dart (a modern, concise, object-oriented language) that developers can easily approach read, change, replace, or remove. This gives developers tremendous control over the system, as well as significantly lowers the bar to approachability for the majority of the system.

How does Flutter run the code on Android?
The engine's C and C++ code are compiled with Android's NDK. The Dart code (both the SDK's and ours) are ahead-of-time (AOT) compiled into native, ARM, and x86 libraries. Those libraries are included in a "runner" Android project, and the whole thing is built into an APK. When launched, the app loads the Flutter library. Any rendering, input, or event handling, and so on, is delegated to the compiled Flutter and app code. This is similar to the way many game engines work.

Debug mode builds use a virtual machine (VM) to run Dart code (hence the “debug” banner they show to remind people that they’re slightly slower) in order to enable Stateful Hot Reload.

What operating systems can I use to build a Flutter app?
Flutter supports development on Linux, Mac, and, Windows.

What language is Flutter written in?
We looked at a lot of languages and runtimes, and ultimately adopted Dart for the framework and widgets. The underlying graphics framework and the Dart virtual machine are implemented in C/C++.

Not only the tutorials for app development, we also looked for some important information to speed up the Android Studio and the testing of code on AVD. This includes increasing the allocation of Virtual RAM, removing the unnecessary plug-ins and disabling Internet while we write code and debug it.

CHAPTER 3
SOFTWARE DESCRIPTION

3.1 Introduction

We first downloaded the Flutter SDK and installed it in C/: drive.
The software that we have used is Android Studio. This software proved out to be the right choice as it has the feature of an in-built Emulator (Android Virtual Device). The path of Flutter SDK need to be specified while building a new project in Android Studio. Then we can create a Flutter project which consequently opens up a .dart file. This is the main.dart file where we write our code.

3.2 Components

- Flutter SDK(Software Development Kit)
- Android Studio
- Android Virtual Device(AVD)
- Required Dependencies & plug-ins (including the DART & Flutter plugin)
- Assets and Images

3.3 Description of Components

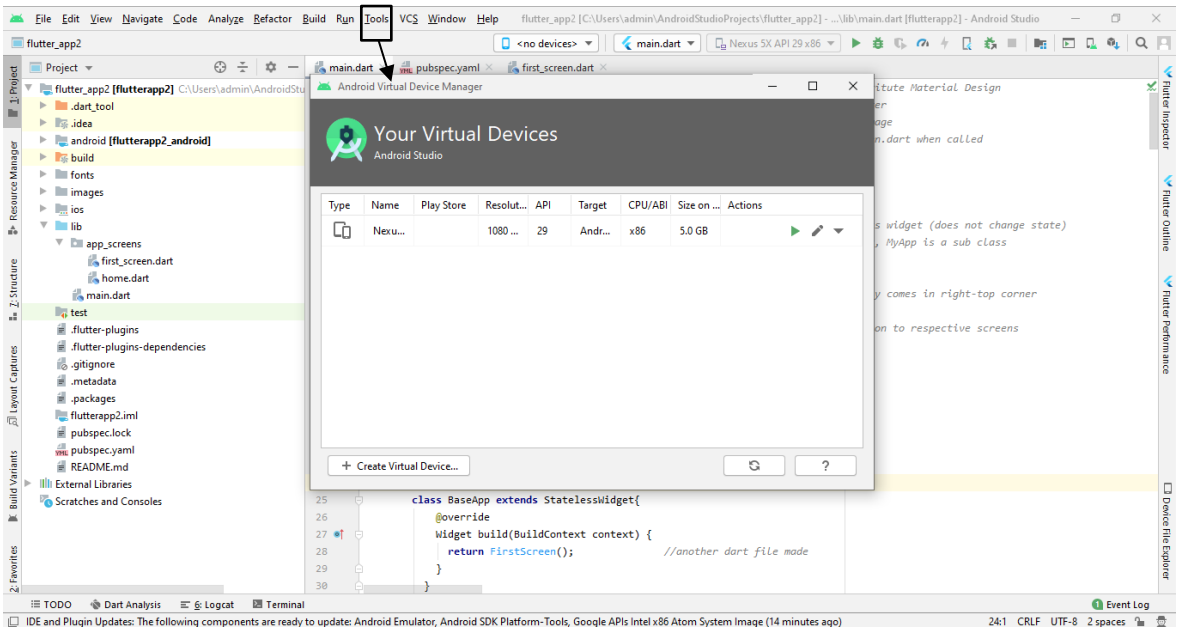
Android Studio: Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

The following features are provided in the current stable version:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations
- Support for building Android Wear apps
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine
- Android Virtual Device (Emulator) to run and debug apps in the Android studio

Emulator or AVD: The Android SDK includes an Android device emulator — a virtual device that runs on your computer. The Android Emulator lets you develop and test Android apps without using a physical device.

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.



Dart: Flutter apps use the Dart programming language. Dart is developed and maintained by Google. It was developed as a successor to Javascript and incorporates many of the features in the ES7 Javascript standard.
As mentioned before, Dart is compiled ahead of time. This allows for faster execution and avoids the Javascript bridge. The code still needs an interface to communicate with the platform code. This is however much faster than the Javascript bridge. Flutter also does not utilize the platform OEM widgets, reducing the number of times communication with the platform occurs.
Dart can also be compiled using a just in time compiler. This is done during development and allows for rapid development cycles.

Flutter takes advantage of the just in time compiler using a feature called hot reload. Hot reload allows code changes to be reloaded and injected on the device very fast. This allows for very efficient development of Flutter apps. The downside is that the app will be less efficient during development.

Why did Flutter choose to use Dart?
Dart scores highly for us on the following primary criteria:

- Developer productivity. One of Flutter's main value propositions is that it saves engineering resources by letting developers create apps for both iOS and Android with the same codebase. Using a highly productive language accelerates developers further and makes Flutter more attractive. This was very important to both our framework team as well as our developers. The majority of Flutter is built in the same language we give to our users, so we need to stay productive at 100k's lines of code, without sacrificing

approachability or readability of the framework and widgets for our developers.

- Object-orientation. For Flutter, we want a language that's suited to Flutter's problem domain: creating visual user experiences. The industry has multiple decades of experience building user interface frameworks in object-oriented languages. While we could use a non-object-oriented language, this would mean reinventing the wheel to solve several hard problems. Plus, the vast majority of developers have experience with object-oriented development, making it easier to learn how to develop with Flutter.

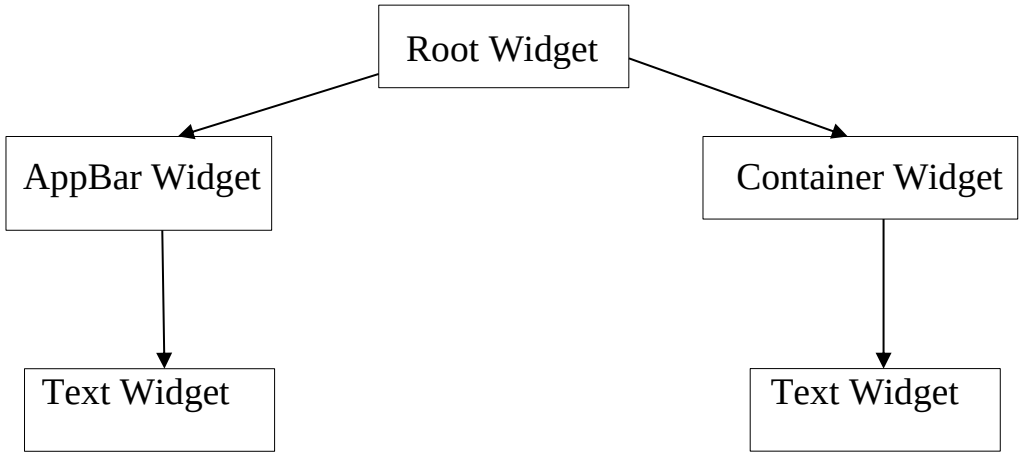
- Predictable, high performance. With Flutter, we want to empower developers to create fast, fluid user experiences. In order to achieve that, we need to be able to run a significant amount of end-developer code during every animation frame. That means we need a language that both delivers high performance and delivers predictable performance, without periodic pauses that would cause dropped frames.

- Fast allocation. The Flutter framework uses a functional-style flow that depends heavily on the underlying memory allocator efficiently handling small, short-lived allocations. This style was developed in languages with this property and doesn't work efficiently in languages that lack this facility.

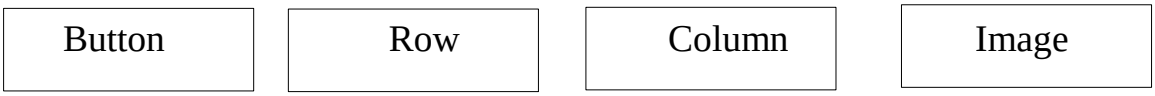
3.4 Integral Components of Code

***Widgets*:** Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state.

Everything inside a Flutter app is essentially a widget. A sample widget tree in Flutter is as follows:



Various other widgets are-



What is Scaffold?

A Scaffold Widget provides a framework which implements the basic material design visual layout structure of the flutter app. It provides APIs for showing drawers, snack bars and bottom sheets.

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.

Stack: Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web's absolute positioning layout model.

***Container*:** The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three dimensional space using a matrix.

CHAPTER 4
APP DEVELOPMENT

The app development began with the installation of Android Studio which had the capability of a built-in Emulator. It was a tedious task as it involved alteration of various settings of Windows which was required for proper functioning of Emulator along with Android Studio.

The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device.

The emulator provides almost all of the capabilities of a real Android device. Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

In order to work with the emulator we need to enable the Virtual Technology setting in the BIOS settings of PC.

As the Emulator syncs up with the Android Studio, we are all set to write the code in main.dart file and observe its output in the emulator.

We also need to download Flutter SDK (Software development kit) and notify the path of SDK in Android Studio.

- **pubspec.yaml** file: Flutter uses the *pubspec.yaml* file, located at the root of your project, to identify assets required by an app. It contains information about project dependencies and metadata.

- **MaterialApp()**: In any event, the MaterialApp widget is the means to follow the Material Design guidelines in Flutter. The idea is, if you want to make a mobile app in Flutter, you'd first insert the MaterialApp widget at the root of the widget tree to get started. Consequently, your app now uses Material Design. It builds a number of useful widgets at the root of your app.

- **Routing in flutter**: Most apps contain several screens for displaying different types of information. For example, an app might have a screen that displays products. When the user taps the image of a product, a new screen displays details about the product.
  Terminology: In Flutter, screens and pages are called routes.
  In Flutter, a route is just a widget. Navigate to a new route using the Navigator.

- **Floating Action Button**: A floating action button is a circular icon button that hovers over content to promote a primary action in the application. Floating action buttons are most commonly used in the Scaffold.floatingActionButton field. A FloatingActionButton in material

design is a button on a screen that is tied to an obvious action which a user would usually do on that specific screen. This button floats above the content of the screen and usually resides on one corner of the screen.

- **Flutter TTS plug-in**: This plug-in is used for the conversion of text-to-speech (TTS).

To use this plugin :

- add the dependency to your pubspec.yaml file.
  dependencies:
  flutter:
  sdk: flutter
  flutter_tts: `^1.2.0`
- Importing the package in the dart file using the following line:
  import *'package:flutter_tts/flutter_tts.dart';*
- instantiate FlutterTts
  FlutterTts flutterTts = FlutterTts();

- **Assets & Images**: Flutter apps can include both code and assets (sometimes called resources). An asset is a file that is bundled and deployed with your app, and is accessible at runtime. Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBM.

Code Snippets:

`import 'package:flutter/material.dart';`	`// Contains the packages which constitute Material Design`
`import'package:flutter_tts/flutter_tts.dart';`	`// To enable the TTS plugin of flutter`
`import 'dart:ui';`	`//to enable display of image`
`import './app_screens/first_screen.dart';`	`// To direct to the file first_screen.dart when called`

`void main() => runApp(new MyApp());`	`//entry point of app (whatever written here is executed)`

`class MyApp extends StatelessWidget{`
`@override`
`Widget build(BuildContext context){`
`  return new MaterialApp()`
	`//defining a class for the stateless widget (does not change state)`
	`// Stateless widget is a superclass, MyApp is a sub class`

`routes: <String, WidgetBuilder>{`
`'/Curry' : (context) => Curry(),`
`'/DryAloo' : (context) => DryAloo(),`
`'/Voice1' : (context) => Voice1(),`
`'/Gallery' : (context) => Gallery()`
	`// defining the routes for navigation to respective screens`

Scaffold(	// allows various important properties to be implemented
appBar: new AppBar(title: new Text("Curry Recipe",	// appears on top of the displayed screen
style: TextStyle(	//for implementing various styles to the text
decoration: TextDecoration.none,	//removing the default underline
fontSize: 30.0,	//setting font size
fontFamily: 'Raleway',	// defining font style
fontWeight: FontWeight.w300,	// defining font weight/ text thickness
fontStyle: FontStyle.italic,	//styling the text to Italic
color: Colors.white),),	// making the font color to white
backgroundColor: Colors.amber,),	// Background Color for the appBar

class Voice1 extends StatelessWidget{	//Stateless Widget for the TTS Widget implementation screen
final FlutterTts flutterTts = FlutterTts();	//instantiate Flutter TTS plug-in

@override
Widget build(BuildContext context) {
Future _speak() async {
	//defining the speak function which is to be called later

await flutterTts.setLanguage("en-US");	//feature of TTS plugin to set language
await flutterTts.setSpeechRate(0.85);	//feature of TTS plugin to set speech rate
await flutterTts.setPitch(1.2);	//feature of TTS plugin to set pitch
await flutterTts.speak("1. Clean all your utensils as well as the uncut. . . . . 8.Your meal is now ready to eat....Enjoy "); }	//text which is to be converted to speech is placed under flutterTts.speak()

return Scaffold(body: Column(	//adding a column widget to insert the list tiles
children: <Widget>[ListTile(	//adding a list tile to column so as to insert text
title: new Text("1. Clean all your utensils as well the uncut ladyfinger(भिंडी) thoroughly.",	//text as a step 1 (Similarly for other steps)
style: TextStyle(fontSize: 20.0)),),	//styling the text

floatingActionButton: FloatingActionButton(	//defining a FAB to make TTS feature user-friendly
onPressed:() => _speak(),	//calling previously defined function when FAB is pressed
child: Icon(Icons.play_arrow),	//assigning a play icon to FAB

backgroundColor: Colors.deepPurple,),	//assigning a color to FAB

body: Column(	// Column widget to align widgets vertically
children: <Widget>[	//initialising children widgets of column
Row(	// Row widget to align widgets horizontally
children: <Widget>[Container(	//initialising children widgets of row
width:100.0, height: 150.0, margin: EdgeInsets.all(15.0),	//setting width, height & margin attributes of Container
decoration: new BoxDecoration(	//to add an image
image: new DecorationImage(image: new AssetImage('images/Chilli.jpg'),	//defining path of image

first_screen.dart

drawer: Drawer(elevation: 10.0,	//appears as three lines on top left of the app screen
child: Center(child: Container(	//to align the child Container widget in the centre
margin: EdgeInsets.only(top: 40.0),	//to assign margin parameters to the container widget
child: Column(	//to implement the various flat buttons of recipes in form of column
children: <Widget>[FlatButton(	//a title bar of recipe with navigation to another screen
child: Text("Curry", style: TextStyle(decoration: TextDecoration.none, fontSize: 30.0, fontFamily: 'Raleway', fontWeight: FontWeight.w300, fontStyle: FontStyle.normal, color: Colors.white)),	//styling text
color: Colors.amber,	//assigning color to the flat button
onPressed: (){ Navigator.pushNamed(context, '/Curry'); },),	//pushing to another screen as the flat button is pressed

body: new Stack(	//used to implement one widget over another(required for background image)
children: <Widget>[	//specifying children widgets of Stack
new Container(	//first widget as Container which would contain the background image
decoration: new BoxDecoration(	//to add an image
image: new DecorationImage(image: new AssetImage('images/MITWPU.jpg'),),),	//defining the image asset path

child: BackdropFilter(	//to implement a filter to the image so as to blur it
filter: new ImageFilter.blur(sigmaX: 3.0,sigmaY: 3.0),	//specifying blur
child: new Container(decoration: new BoxDecoration(Colors.black.withOpacity(0.2),),),),),	//assigning opacity to image
Container(	//for the text that is displayed on the base page
margin: EdgeInsets.all(45.0),	//assigning margins to the container
child: new Text("Welcome to this cooking app. It has been developed as a part of our Mini-Project.",	//text for the container
style: TextStyle(decoration: TextDecoration.none, fontSize: 30.0, fontFamily: 'Roboto', fontWeight: FontWeight.w300, fontStyle: FontStyle.italic, color: Colors.black)),),	//styling text

Container(margin: EdgeInsets.only(top:400.0, left: 100.0, right: 100.0),	//for the flat button on the base page
padding: EdgeInsets.all(15.0),	//assigning spacing from adjacent widgets
child: FlatButton(child: new Text("Gallery",	//text for the flat button
style: TextStyle(decoration: TextDecoration.none, fontSize: 45.0, fontFamily: 'Raleway', fontWeight: FontWeight.w300, fontStyle: FontStyle.italic, color: Colors.white)),	//styling the text for the flat button
color: Colors.redAccent,	//assigning color to flat button
onPressed: () { Navigator.pushNamed(context, '/Gallery'); },),	//Navigator to push to Images screen

CHAPTER 5
IMPLEMENTATION AND DEBUGGING

The following are the two ways to implement the code and observe the output on the Emulator. The difference in the two ways is as follows:

1. **Hot Reload feature**: Flutter's hot reload feature helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM). After the VM updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing you to quickly view the effects of your changes.

2. **Hot Restart feature:** Hot restart takes more time than hot reload, and it destroys or rebuilds the state value and rebuilds it to the default. The app widget tree is completely rebuilt with a newly typed code. This takes about 23-30 seconds compared with the default app restart time.

- The Hot Reload feature proves out to be quite useful while testing apps as all it takes is few seconds to implement the change that we do in the code.

- The Emulator plays a very important role in helping us test the app and presents an exact view how the app will look when installed and run on a smartphone.

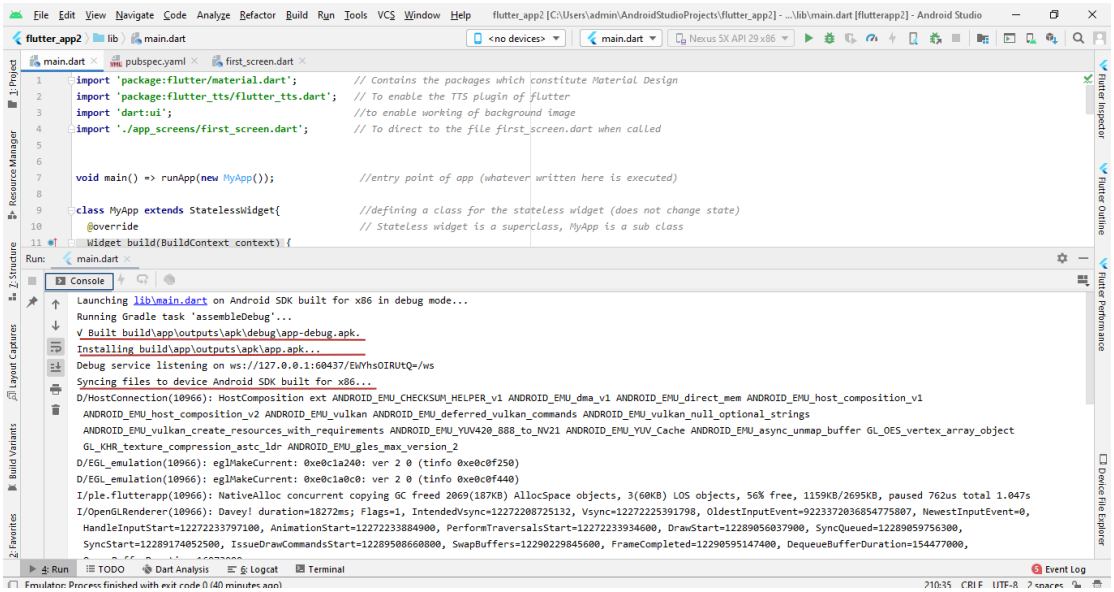
- The emulator when started with the Android Studio utilises a lot of RAM which usual processes don't consume. This can cause the PC to hang if another process is started or the Internet is enabled.

- Sometimes, debugging requires a lot of patience as the Emulator takes some time initially to catch up with the changes made in the code in Android Studio.

- Debug mode builds use a virtual machine (VM) to run Dart code (hence the "debug" banner they show to remind people that they're slightly slower) in order to enable Stateful Hot Reload.

Each time we Hot restart the code, a number of changes take place which require a longer time as compare to Hot Reload.

A screenshot of how the coding workspace looks is as follows:



The Console where the debugging processes appear is highlighted.

During app development:

- JIT (Just-in time compilation)
- Faster compilation during development
- Faster app reload

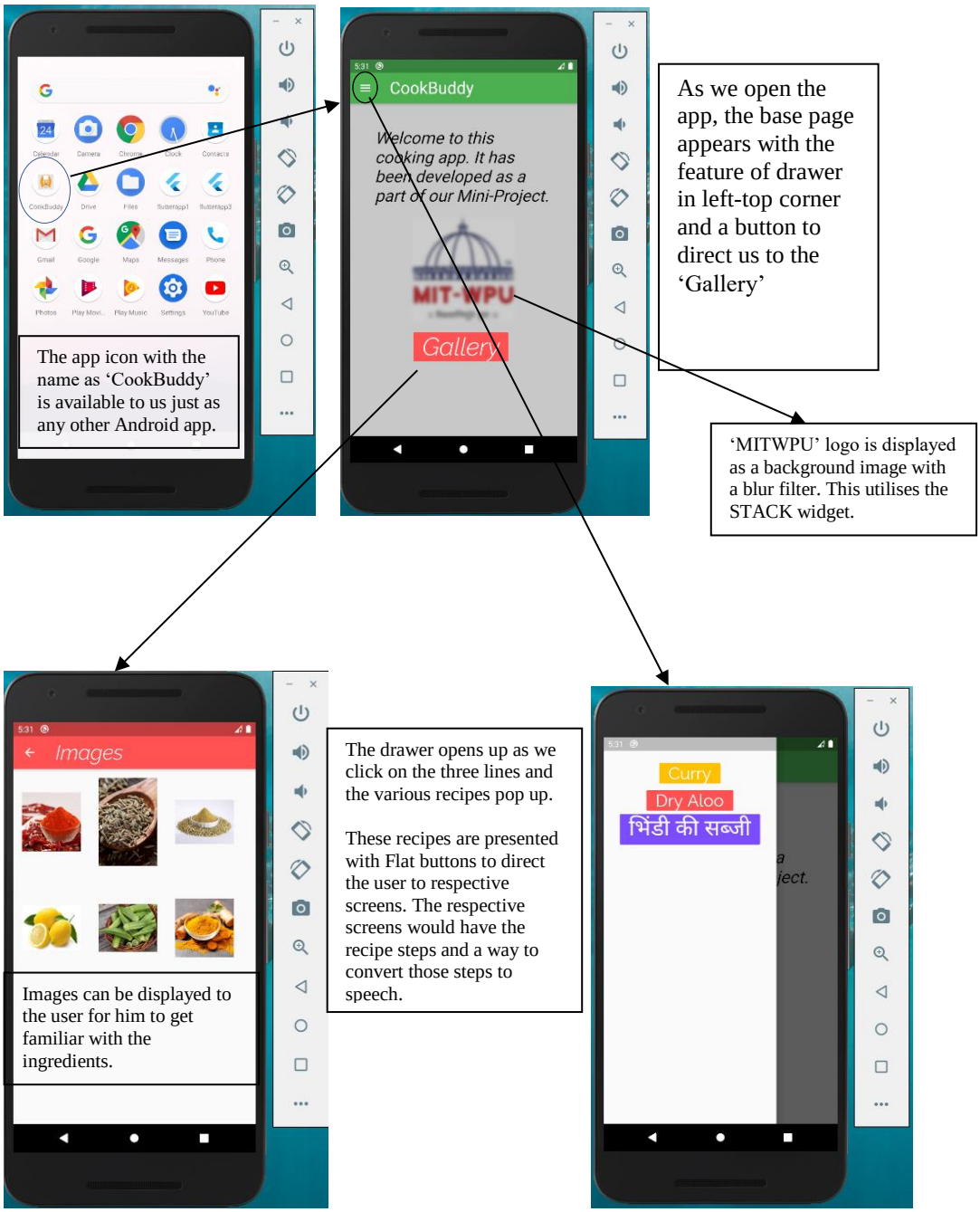
When app is released:

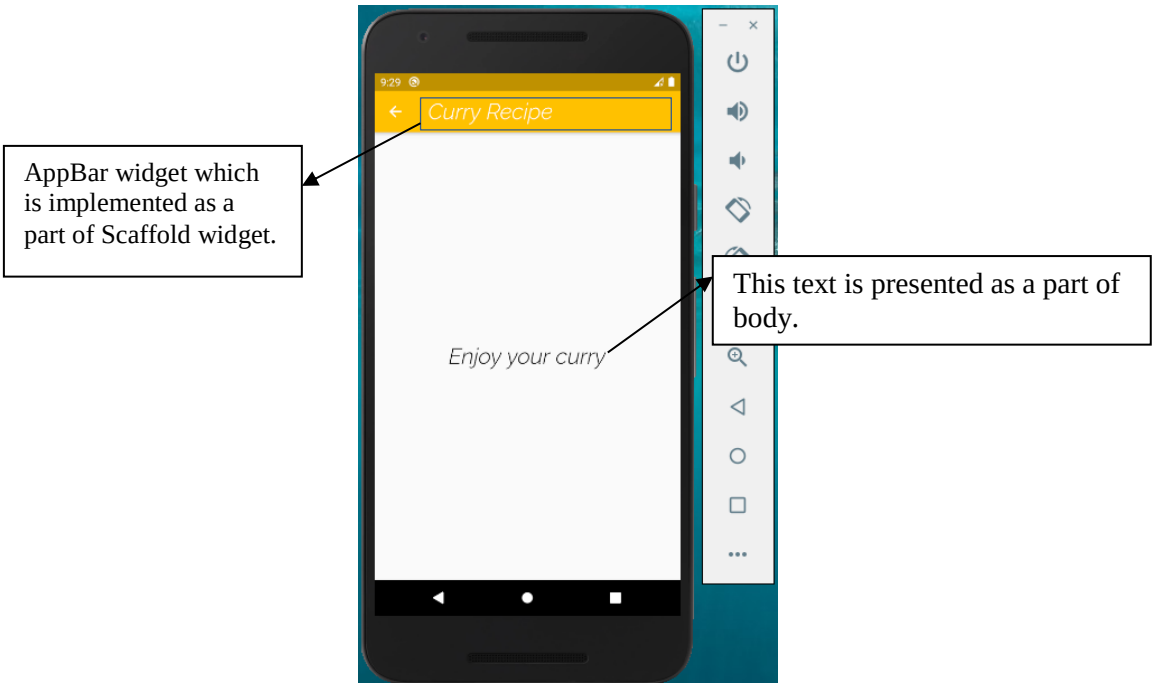
- AOT (ahead-of-time): while running apps on device it is much faster

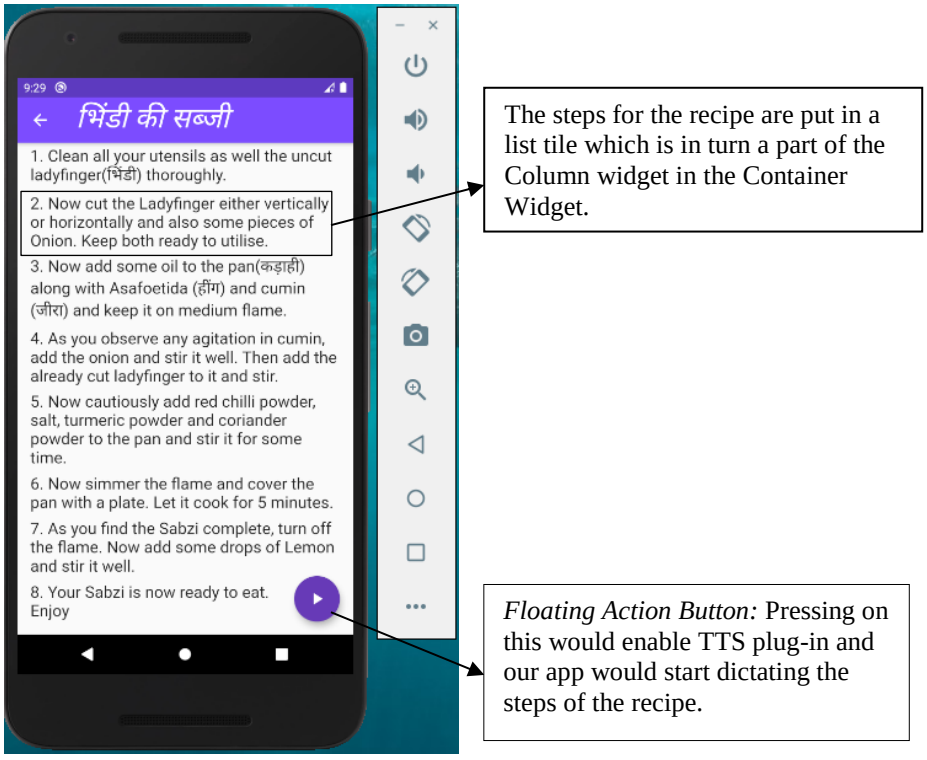
The major advantages are faster development and execution due to JIT & AOT. Moreover, the user experience is same as compared to Native apps for both Android and iOS.

CHAPTER 6
RESULT

6.1 App Snapshots







6.2 Derived Results

- Flutter SDK lets you build an android application quickly with a comparatively low end machine as well.

- Thanks to its reactive framework, Flutter automatically updates the contents of the interface when required. A mere updating of the state variables does the trick and this greatly assists in security checks and QA tasks.
- Similar to Android Studio's Instant Run Feature, Flutter too has a Hot Reload feature that instantly lets you see the result of your code change. This lets developers code quicker and experiment freely thereby creating innovative results.
- Flutter lets you define the looks and functionality of your application using dart alone. This way you don't have to deal with XML files for creating layouts. On top of that the modern layout widgets can be made to work as templates.
- Dart libraries are updated constantly and that greatly assists the quality of code one can develop in flutter. Also it makes your code precise and less bulky.
- Thanks to its recent intervention into the mobile app development ecosystem, its creators kept in mind the compatibility concerns for both platforms. Flutters widgets comply with the design guidelines of both platforms.
- Keeping all these factors in mind, Flutter is definitely a very anticipating topic and is bound to create an impact in some form or the other. It is capable of delivering some super smooth and responsive animations to UI elements and appeals to the interest of several industry verticals. More importantly it is for sure that cross-platform development will not be only what react native preaches.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

Overall it was a great experience building an app for the first time on this newly developed SDK, Flutter.
Flutter is very new, but a promising platform that has attracted the attention of large companies who've released their app already using flutter. It is interesting because of its simplicity compared to developing web applications, and because of its speed as compared with native applications.
The output and the designs have a lot of scope for improvement which can be implemented as we learn more about app development as a whole.

7.2 Future Scope

As a part of future scope, we can release this Android app and publish it on Google Play Store.
Before publishing, we can put some finishing touches on our app. The steps to be followed to publish the app can be listed as-

- Adding a launcher icon
- Signing the app
  To publish on the Play Store, you need to give your app a digital signature.
- Shrinking your code with R8
- Reviewing the app manifest
- Reviewing the build configuration
- Building the app for release
- Publishing to the Google Play Store

APPLICATIONS

- This app can find application for the users who wish to learn cooking and make their own food. The app can prove out to be a great start for the beginners.
- Only in the times of crisis is when we realise the importance of things and skills which we often ignore to recognise. Self-sufficiency and its importance prove out to be really important in today's times (Corona outbreak). This app would truly help an individual in becoming self-sufficient.
- The scope of this app can further be expanded to various other skills which are integral to each and every human. Just as steps are presented in this app, steps to make any other things can also be implemented.
  Moreover, notekeeper (which utilises the property of a Stateful Widget) can also be implemented as a part of this app. This would allow the user to enter various recipes for various other creations and later save them for further use.

REFERENCES

- https://developer.android.com/studio/run/managing-avds
- https://flutter.dev/docs/resources/faq
- https://www.youtube.com/watch?v=qWL1lGchpRA&list=PLR2qQy0Zxs_UdqAcaipPR3CG1Ly57UlhV (Flutter Tutorials - MTECHVIRAL)
- https://www.youtube.com/watch?v=fmPmrJGbb6w&list=PLlxmoA0rQ-Lw6tAs2fGFuXGP13-dWdKsB (Flutter Tutorial for Beginners using Dart: Develop Android and iOS mobile app - SMARTHERD)
- https://www.youtube.com/watch?v=1ukSR1GRtMU&list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ (Flutter Tutorial for Beginners- The Net Ninja)
- https://medium.com/flutter
- https://pub.dev/packages/flutter_tts#-example-tab-