**Day-1 & 2**

**Welcome to Day 1** 😎

We'll get to the fun stuff soon.

But before that, the first step is to install the Flutter SDK and get familiar with your IDE so you can make the most of it. If you have already done this, you can skip to the bottom and discover my favourite VSCode extensions.

**Installation**

Full installation instructions are available on the official website:

• Install Flutter

If you want to build Flutter apps on Android and iOS, you'll need to install the corresponding SDKs, as well as the Android emulator and iOS simulator. The guide above explains how to do this.

Note: You can only develop Flutter apps for iOS on a macOS system. If you're on Windows, you can only build for Android. With that said, you could explore services like macincloud and Remote Mac to build iOS apps remotely.

**Configuring your IDE**

You'll also need a code editor such as Android Studio or Visual Studio Code, along with the Dart and Flutter plugins:

• Set up an editor

To make the most of your IDE, check out the documentation for tips on running and debugging your apps, using code snippets, keyword shortcuts, and more:

• Android Studio and IntelliJ

• Visual Studio Code

**Flutter doctor**

Once everything is installed (don't forget to configure your system PATH on Windows or macOS!), you should be able to type flutter doctor on your terminal. And your output should look something like this:

Doctor summary (to see all details, run flutter doctor -v):

[✓] Flutter (Channel stable, 2.8.0, on macOS 11.6 20G165 darwin-x64, locale en-GB)

[✓] Android toolchain - develop for Android devices (Android SDK version 31.0.0)

[✓] Xcode - develop for iOS and macOS (Xcode 13.0)

[✓] Chrome - develop for the web

[✓] Android Studio (version 4.2)

[✓] VS Code (version 1.62.3)

[✓] Connected device (4 available)

• No issues found!

If flutter doctor doesn't report any errors, you already have everything you need to get started building apps.

Time for a quick test drive. This page shows you how to create a new Flutter project and run it:

• Test drive

**Bonus: VS Code extensions to develop like a Flutter Pro**

If you use VSCode, I highly recommend adding some useful extensions. These are my favourites:

• Dracula official theme (who doesn't like Dark Mode? )

• Error Lens (get any feedback about errors as you type: super useful)

• Pubspec assist (*note: similar functionality is also available with the new "Dart: Add Dependency" and "Dart: Add Dev Dependency" commands*)

• Todo Tree

• Better Comments

• Remove Comments

There are many others, but these are the ones I use all the time.

There are also some VSCode settings you can enable to improve your development experience. These can be set by opening **Preferences: Open Settings (JSON)** in the command palette:

• Set "editor.formatOnSave": true to enable **format on save** (works great with dartfmt)

• Set "editor.bracketPairColorization.enabled": true to enable **high performance bracket pair colorization** (explained here)

You can find a copy of my VSCode settings.json file on this gist.

Congratulations, your Flutter development environment is ready to go!

**Daily Challenge 1**

Share the screenshot of your flutter doctor output, and let everyone know you're learning Flutter!!

**Fueling Mind with Dart**

Flutter apps are built using **Dart**, which is an easy language to learn.

The easiest way to try it is to use Dartpad, a free online editor that can run Dart and Flutter apps directly on your browser, without having to install anything on your machine.

**Dart Language Tour:** https://dart.dev/guides/language/language-tour

**Top 16 dart tricks and tips every flutter developer should know:** https://codewithandrea.com/videos/top-dart-tips-and-tricks-for-flutter-devs/?utm\_source=Newsletter&utm\_medium=email&utm\_campaign=02-dart-language

Day-3

Welcome to Day 3 😎

In day 3, you will learn about the Dart type system, the Effective (to make your code more robust and easy to understand), and the Dart Null Safety. 🎯

• Dart Type System: https://dart.dev/guides/language/type-system

• Effective Dart: https://dart.dev/guides/language/effective-dart

• Understanding Null Safety: https://dart.dev/null-safety/understanding-null-safety

You can also cover Dart Null Safety from the following video: https://www.youtube.com/watch?v=llM3tcpaD5k&list=PL6yRaaP0WPkVtoeNIGqILtRAgd3h2CNpT&index=6

Its Daily Challenge Time 🚀

**Day-4 & 5**

**Welcome to Day 4** 😎

Well, after spending 3 extensive days in the dart documentation, it is finally time to get your hands on some coding exercises in dart. The purpose of these coding exercises is the revision of all the basics that you have covered in the last three days. So let’s dive in 🤿

🚀 **Basic Programming** section that will cover the **data types, operators, control statements**, **loops** and **arrays**.

• [ ] Write a program to calculate area of an ellipse having its axes (minor=4cm, major=6cm).

• [ ] Write a program to calculate simple interest for a given P=4000, T=2, R=5.5. (I = P*T*R/100).

• [ ] Write a program that takes the temperature in Fahrenheit and convert it to Celsius And Kelvin:

K = C + 273

C = (F – 32) / 1.8

• [ ] Write a program to swap two variables values.

• [ ] Check whether a number is even or odd using ternary operator?

• [ ] Check whether the entered character is vowel or consonant?

• [ ] Write a program to enter month number between (1-12) and print number of days in month.

• [ ] Write a program to input two integer numbers and display the sum of even numbers between these two input numbers.

• [ ] Write a program to check if a number input by user is PRIME or not, range of input is 1 to 300.

• [ ] Write a function minmax() that takes four integers as input and display the minimum and maximum number.

• [ ] Print 1 to 100 using recursion.

• [ ] Write a program to input two arrays from user of the same size and find the sum of arrays (element by element).

• [ ] Write a program to input array from user and display the elements of array in reverse order (use seperate loops for input and output operation).

• [ ] Write a program that takes a string as input. If the character is upper case convert it into lower case and if the character is lower case then convert it in to upper case.

• [ ] Write a program that initializes a string and finds how many times a substring (entered by user) appears in it. The size of substring must be less than or equal to the size of string.

Note: “is” is a substring of “This is my bigger string and it is c plus plus string”. The substring appears 2 times in the string.

• [ ] Take an array as input from user and calculate the following:

• Mean

• Median

• Mode

• [ ] Write a program to move the given number to the end of a given array.

Expected Output:

The given number is: 0

The given array is: 2 5 7 0 4 0 7 -5 8 0

The new array is:

2 5 7 8 4 -5 7 0 0 0

• [ ] Write a program to move the given number to the end of a given array.

Expected Output:

The given number is: 0

The given array is: 2 5 7 0 4 0 7 -5 8 0

The new array is:

2 5 7 8 4 -5 7 0 0 0

• [ ] Write a program to find the second largest element in an array

🚀 **OOP** section that will cover the basic concepts of OOP, such as **classes, objects, constructors, default constructors,**and many more.

• [ ] Create a class, Heater that contains a single integer field, temperature. Define a constructor that takes no parameters. The temperature field should be set to the value 15 in the constructor. Define the mutators: warmer and cooler, whose effect is to increase or decrease the value of the temperature by 5 respectively. Define an accessor method to return the value of temperature. Demonstrate the use of Heater class.

• [ ] Create a class called Point that has two data members: x‐ and y‐coordinates of the point. Provide a no‐ argument and a 2‐argument constructor. Provide separate get and set functions for the each of the data members i.e. getX, getY, setX, setY. The getter functions should return the corresponding values to the calling function. Provide a display method to display the point in (x, y) format. Make appropriate functions const.

• [ ] Create a class called BankAccount that models a checking account at a bank. The program creates an account with an opening balance, displays the balance, makes a deposit and a withdrawal, and then displays the new balance. Note in withdrawal function, if balance is below Rs. 500 then display message showing insufficient balance otherwise allow withdrawal.

**Week 2**

Week 2 Day 1

**Layouts**

**Welcome to day 1 of week 2, where we will talk about widgets and layouts.**

Flutter really shines when it comes to building beautiful UIs.

With Flutter, you build UIs by using widgets and composing them together in a declarative manner. You can think of widgets as UI components that describe what your application looks like.

Flutter offers an extensive set of widgets, closely matching the Material Design specification. The Flutter widget catalog also includes Cupertino widgets, which are high-fidelity replicas of all the UI components found on iOS.

Because Flutter controls every pixel on the screen, you can even create your own widgets to deliver a completely custom experience that delights your users.

So what are some good resources to learn about widgets & layouts?

**Flutter widgets**

Getting started with widgets is easy, but after a little while some questions may pop up in your head:

• What is a BuildContext?

• Should I create new widget classes or create helper methods?

To answer these questions, the Flutter team has created a series on YouTube called Decoding Flutter and I highly recommend it. And it even has dedicated videos for the two questions above:

• BuildContext?! | Decoding Flutter

• Widgets vs helper methods | Decoding Flutter

**Flutter layouts**

A good place to start is this official guide about Layouts in Flutter, showing you how to build some common layouts with useful illustrations.

• Flutter Widget of the week: This is truly an amazing video series from the Flutter team. Most videos are 1-to-2 minutes long and help you learn about the most useful widgets. You don't need to watch the entire series at once, but I won't stop you if you do . And sometimes knowing that a widget exists means that you don't have to create one yourself.

**Todo**

**Flutter layouts**

Cover the Basic Widgets: https://docs.flutter.dev/development/ui/widgets/basics

Go through the following UI guide: https://docs.flutter.dev/development/ui/layout

**Daily Challenge 1**

**Let’s build layouts** 🚀

Complete the following layout documentation and share the screenshot of your emulator/phone screen:

https://docs.flutter.dev/development/ui/layout/tutorial

Day 2

**Text, and Interactive Widgets**

**Welcome to day 2 of week 2, where we will talk about the Text and Interactive widgets.**

Flutter really shines when it comes to building beautiful UIs.

With Flutter, you build UIs by using widgets and composing them together in a declarative manner. You can think of widgets as UI components that describe what your application looks like. Beautiful text and interactive widgets makes your app more user friendly and attractive to the users.

**Todo**

**Text and Interactive Widgets**

Cover the Text Widgets: https://docs.flutter.dev/development/ui/widgets/text

Interaction with Widgets: https://docs.flutter.dev/development/ui/widgets/interaction

**Daily Challenges**

**Let’s use the magic of text and interaction** 🚀

Complete the following cookbooks for interactions:

https://docs.flutter.dev/cookbook/gestures/handling-taps

https://docs.flutter.dev/cookbook/gestures/ripples

https://docs.flutter.dev/cookbook/gestures/dismissible

**Day 3**

Input and Scrollable Widgets

Welcome to day 3 of week 2, where we will talk about the Input and Scrollable widgets.

Flutter really shines when it comes to building beautiful UIs.

With Flutter, you build UIs by using widgets and composing them together in a declarative manner. You can think of widgets as UI components that describe what your application looks like. Eye catching input fields and smooth scrolls makes your application more user friendly and attractive.

Todo

Input and Scrollable Widget’s Docs

Input Widgets:

https://docs.flutter.dev/development/ui/widgets/input

https://api.flutter.dev/flutter/material/TextField-class.html

https://api.flutter.dev/flutter/material/TextFormField-class.html

Scrollable Widgets

https://docs.flutter.dev/development/ui/widgets/scrolling

Daily Challenges

Let’s use the magic of input and scrolling 🚀

Complete the following cookbooks for interactions:

https://docs.flutter.dev/cookbook/forms/validation

https://docs.flutter.dev/cookbook/forms/text-input

https://docs.flutter.dev/cookbook/forms/focus

**Day 4 & 5**

Building your First mobile app 📱

Welcome to day 5 of week 2, where we will be building your first mobile application by using the widgets which you have covered so far.

What will you be learning?

• How to write a Flutter app that looks natural on iOS, Android, desktop (Windows, for example) and the web

• Basic structure of a Flutter app

• Finding and using packages to extend functionality

• Using hot reload for a quicker development cycle

• How to implement a stateful widget

• How to create an infinite, lazily loaded list

In part two of this section, you'll add interactivity, modify the app's theme, and add the ability to navigate to a new page (called a route in Flutter).

So let’s dive in 🤿

Todo 🚀

Part 1:

https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1#0

Part 2:

https://codelabs.developers.google.com/codelabs/first-flutter-app-pt2/#0

**Week 3**

MVVM & State Management

What about Architecture?

The state management solutions above are tools that you can use to build your apps.

But tools alone are not enough. To build an app, you need to use them as part of a broader application structure or design.

In building Flutter apps over the last 3 years, I've identified three main application layers that are common to most apps:

• UI Layer: this is where we put our widgets

• Domain layer: this is where we define our data models and business logic

• Service Layer: this contains wrappers for networking/backend code (e.g. REST APIs, authentication, Cloud Firestore etc.)

Content for this section is as follow:

• https://medium.com/flutter-kp/state-management-a-simple-explanation-9aea94c99287

• https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro

To get an overview about MVVM, dive into the following blog:

https://medium.com/flutterworld/flutter-mvvm-architecture-f8bed2521958

What about State Management?

Once you're feeling confident with the built-in state management capabilities in Flutter, you can start looking at some of the other solutions.

Here I focus on the most stable, well documented, maintained, and supported packages. The ones I used personally and can recommend without hesitation are:

• provider

Provider

Simply put, Provider = InheritedWidget + Generics. Provider gives you scoped access to things in your widget tree, by type. It is officially endorsed by the Flutter team. It's mature, and you'll find a lot of documentation and Q/A on StackOverflow about it.

For a good conceptual overview of Provider, I recommend Flutter Provider video series on YouTube.

The best place to read about provider is its official documentation, so let’s dive in

https://pub.dev/packages/provider

**Day 2**

Building a Plant Store App

This week, we will be diving to work on real-world application known as the Plant App. The design of this app is fetched from the Behance.com (links will be provided at the bottom of this note).

During the development of this design, you are supposed to follow all the concept that you have learnt so far, specifically the MVVM architecture and the state-management via providers. While development, please note that you are not supposed to enter the static data into the fields which are supposed to be dynamic. For each screen there should be a model, and the data for that screen should be fetched from the services class, which will be having a data class consisting of the required mock data.

In this design you must develop the following screen sections:

• Signup/Login

• Home

• Product Details

• Cart

Note: When the products are added to the cart, you are supposed to add an indicator above the cart logo, which will indicate that the cart is not empty.

If you can’t find the similar images which are mentioned in the design you can use the images of your choice, however, make sure your design looks like the one which is being shared with you.

Wishing you all the best for the week 03, the week of implementation and learning via real projects.

The images for the design are as follows.

• Signup/Login Section

• Home Section

• Product Details Section

• Cart Section

Link to the design to fetch fonts and colors: https://www.behance.net/gallery/130319003/Plant-Store-App/modules/738816611

**Week 4**

Firebase

The main purpose of a database is to efficiently order and manage large amounts of information and make it accessible for users in various ways. In the case of app development, databases make sure that all the data of that app, including the code, user data, and much more, are safely stored in one place.

To initiate the journey of the databases, firebase can one of the best choice for the beginners with little or no knowledge of the databases. Secondly, the documentation of Firebase is written in a way that anyone can learn and gain knowledge related to it in no time.

To Do:

• For those who cannot grasp the concept in a better way from the documentation I am sharing a video for them here: [click on the link below]

Codelab: Get to know Firebase for Flutter

• Firebase overview: https://firebase.flutter.dev/docs/overview

• Firebase CLI: https://firebase.flutter.dev/docs/cli

• Introduction to Cloud Firestore: https://www.youtube.com/watch?v=QcsAb2RR52c

• Cloud Firestore overview: https://firebase.flutter.dev/docs/firestore/overview

• Cloud Firestore usage: https://firebase.flutter.dev/docs/firestore/usage

Note: If you have any queries you can reach me out via slack. Soon you will be assigned a simple application that you will build using the firebase as database.

**Week 5**

**API’s**

API stands for Application Programming Interface. In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses. Their API documentation contains information on how developers are to structure those requests and responses.

Api: https://api.flutter.dev/

https://www.youtube.com/watch?v=s7wmiS2mSXY&ab\_channel=MuleSoftVideos

install Postman:

Mac User : https://dl.pstmn.io/download/latest/osx\_64

Window User: https://www.filehorse.com/download-postman-64/

**Get Api**

The GET method refers to a HyperText Transfer Protocol (HTTP) method that is applied while requesting information from a particular source. It is also used to get a specific variable derived from a group. The HTTP POST asks for input of information from the supplying browser into the server’s message system. Structures of HyperText Markup Language (HTML) come in either POST method or GET method. The method applied, be it the POST method or the GET method, settles on how form data shall be presented before the server.

**https://www.youtube.com/watch?v=81kEKRHMp3s&t=551s&ab\_channel=TheTechBrothers**

Provide Fake API for Practice

https://jsonplaceholder.typicode.com/

**Post Api**

POST is the HTTP method that is designed to send loads of data to a server from a specified resource. Most common HTML forms on the web operate using this request method. It usually transmits relatively small loads of data to a receiver. This method allows data to be sent as a package in a separate communication with the processing script. This means that data sent through the POST method will not be visible in the URL, as parameters are not sent along with the URI.

The format of an HTTP POST should have HTTP headers, followed by a blank line, followed by the request body. POST request can be used to submit a web form or upload a file, but it is critical to ensure that the receiving application resonates with the format being used. The Content-Type header indicates the type of body in the POST request.

Provide Fake API

https://jsonplaceholder.typicode.com/

**Week 6**

**Google Map:** Google Maps, we all are quite aware of the fact of how crucial Maps are in our life, from locating a place to locating nearby shops, in our daily commute, taxi, food deliveries. The first thing to observe is how important maps are? and the second thing to observe is that most of the apps these days have maps in them, so Maps in App development are becoming a serious thing from day to day. Now when we are talking about development in Flutter, its a fact that maybe you come across a situation where you want to **implement maps in Flutter**and the first question after that thought comes is HOW TO IMPLEMENT IT? or WHERE TO START FROM?

**Platform Doc:** https://developers.google.com/maps/documentation

Integrating Google Maps API key

The next step is the last step before you could start coding for the map in Flutter and undoubtedly the important one as this step involves integrating your Google Map API key into your project for both the platforms Android and as well as iOS.

**Adding Google Maps**

https://codelabs.developers.google.com/codelabs/google-maps-in-flutter#0

**Using API**

https://developers.google.com/maps/documentation/android-sdk/get-api-key