offenso
Tech School

FLUTTER HANDBOOK

40/445 Automobile Road,
Mamangalam Kara, Palarivattom
P.O Kochi-682025

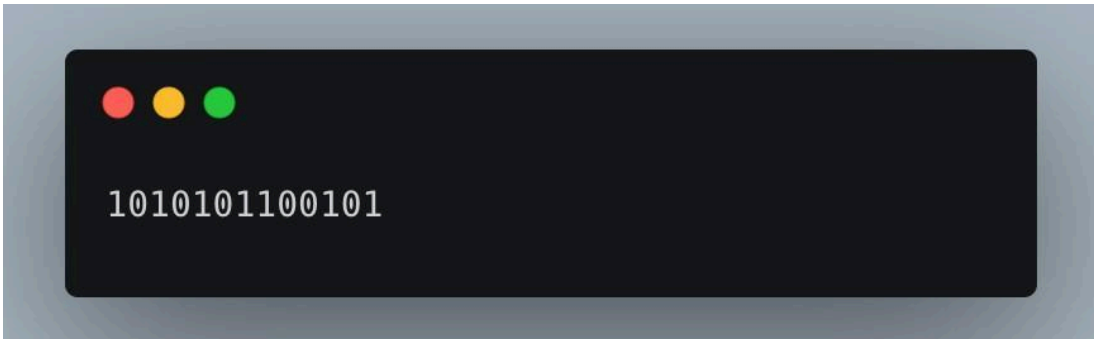
Index

1. What is a programming language?

A **programming language** is a way for humans to **give instructions to a computer** so it can do a task.

Just like we use English or Malayalam to talk to people, we use programming languages (like **Dart, Python, Java, C**) to talk to computers.

Computers don't understand human languages.They only understand **machine language (0s and 1s)**:



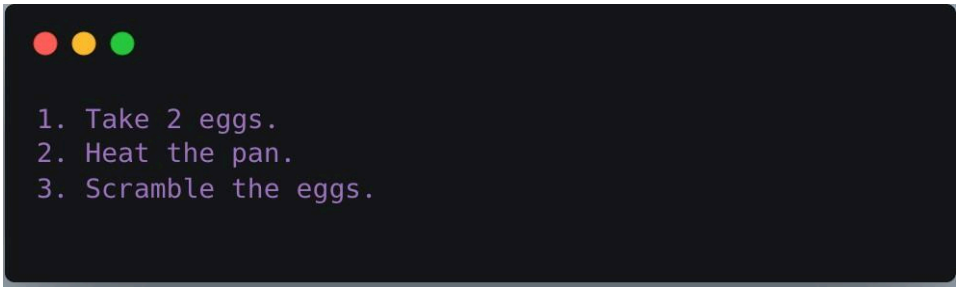
But humans cannot write this easily. So programming languages act as a **bridge** between humans and computers.

1.2 Simple Analogy: Programming = Writing a Recipe

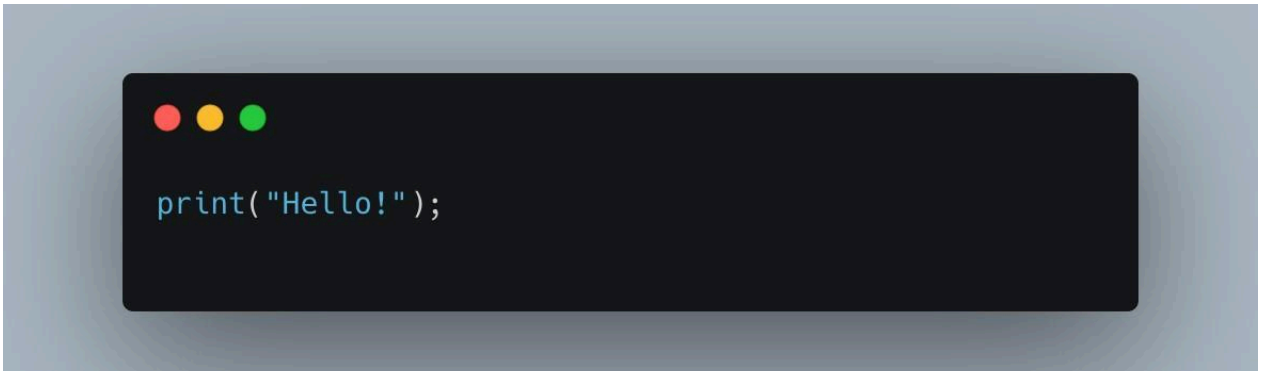
A programming language is like **writing a recipe for a dish**.

- A recipe tells a cook what to do → A program tells the computer what to do.

Recipe example (human language):



💻 **Programming example (Dart language):**



Both are sets of instructions.
The difference is: one is for a cook, the other is for a computer.

1.3 Why do we need programming languages?

Because computers:

- cannot think
- cannot guess
- do exactly what we tell them so instructions must be clear, structured, and precise.

Programming languages help us:

- build mobile apps
- design websites
- make games
- control robots
- handle data
- run machines

Everything from Instagram to your phone's calculator is created using programming languages.

1.4 Types of Programming Languages (Simple View)

High-level languages (easy to read)

- Dart
- Python
- JavaScript
- Java

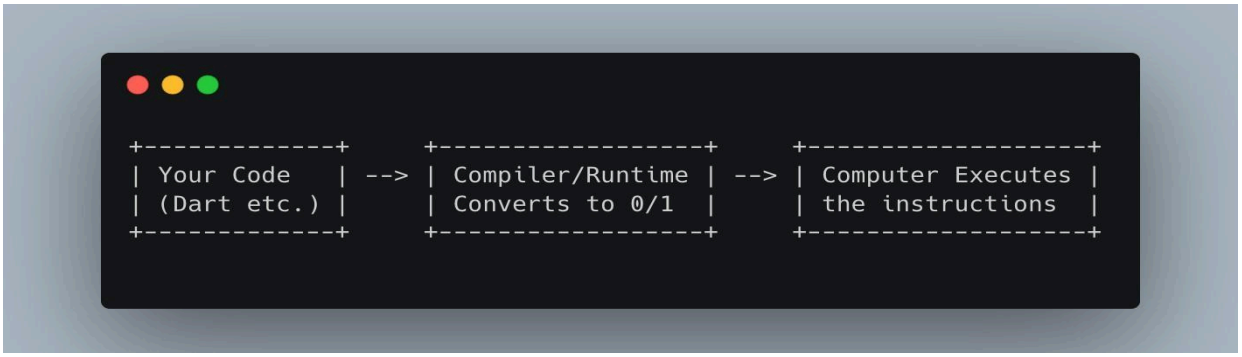
These look similar to English:

Low-level languages

Closer to machine code. Harder to read.

1.5 How Does Code Turn Into Something the Computer Understands?

Your Code → Compiler/Interpreter → Machine Code (0s & 1s) → Computer Executes It



1.6 Simple Example: Program to Add Two Numbers

This tells the computer:

- store 5 in **a**
- store 10 in **b**
- add them
- show the result

Just like telling a friend: “Take 5 apples + 10 apples = tell me the total.”

2. What is Dart?

Dart is a **programming language created by Google**, and it is the **main language used to build Flutter apps**.

Think of Dart as the **language you use to talk to Flutter**.

Flutter = the framework
Dart = the language you write code in

Without Dart, Flutter cannot run.

2.1 Why Did Google Choose Dart for Flutter?

Google picked Dart because it gives **three big benefits**:

⭐ 1. Fast performance

Dart compiles to **native machine code**, which means Flutter apps run very fast on:

- Android
- iOS
- Web
- Desktop

⭐ 2. Hot Reload support

Flutter's famous *Hot Reload* works smoothly because of Dart. You can update the UI instantly without restarting the app.

⭐ 3. Easy to learn

Dart looks similar to:

- Java
- JavaScript
- C / C++
- Kotlin

So anyone who knows basic programming can understand it quickly.

2.2 What Type of Language Is Dart?

Dart is:

- **Object-Oriented** (uses classes & objects)
- **Type-Safe** (prevents errors by checking data types)
- **Flexible** (supports both JIT & AOT compilation)
- **Portable** (runs on mobile, web, desktop)

2.3 Where Is Dart Used?

Dart is used in:

✔ Flutter mobile apps - (Android + iOS)

✔ Web apps - (Dart → JavaScript)

✔ Desktop apps - (Windows, macOS, Linux)

✔ Backend projects - (with Dart servers like *Dart Frog*, *Shelf*) But

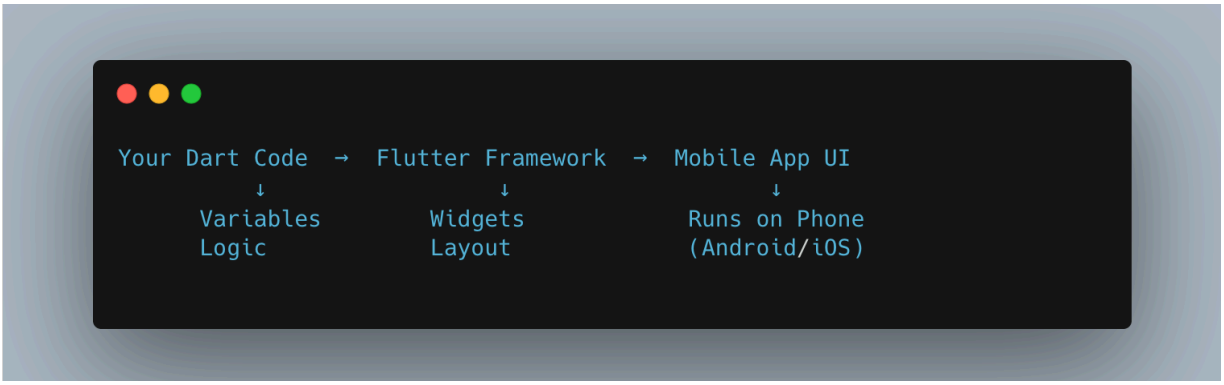
Flutter is the most popular use-case.

Simple dart code example:

```dart
void main() {
  print("Hello from Dart!");
}
```

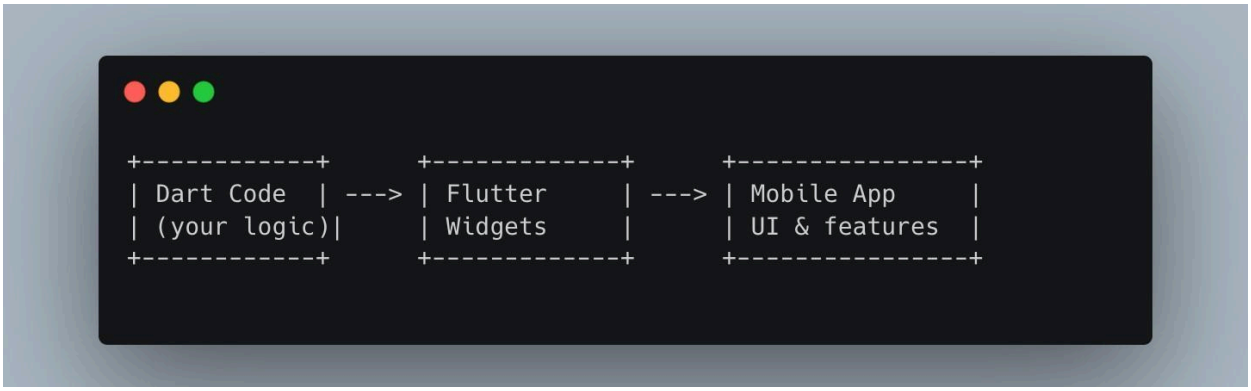
This program prints text to the screen.

2.4 How Dart Works With Flutter (Simple Diagram)



```
Your Dart Code  →  Flutter Framework  →  Mobile App UI
        ↓                  ↓                   ↓
   Variables          Widgets            Runs on Phone
   Logic              Layout             (Android/iOS)
```

Or visually:



2.5 Why Should Beginners Learn Dart?

- It is **simple to learn**
- It is **clean and readable**
- It works perfectly with Flutter
- It gives you access to **mobile, web, and desktop development**
- There is **no complicated setup** like Java/Kotlin or Swift

3. What Is Flutter?

Flutter is an open-source UI toolkit created by Google that allows you to build:

- Android apps
- iOS apps
- Web apps
- Windows apps
- macOS apps
- Linux apps
- Embedded/IoT apps

—all using **one single codebase**.

✔ Flutter uses Dart as its programming language.

✔ It provides ready-made UI components called Widgets.

In simple words: Flutter = Build apps for any device using one code.

3.1 Flutter Advantages

1. One Codebase for All Platforms

You write one app → it runs on:

- Android
- iOS
- Web
- Windows
- Mac
- Linux

This saves months of work and reduces cost.

2. Fast Development with Hot Reload

Flutter has **Hot Reload**:

- Change UI or logic
- Instantly see updates
- No restart
- No waiting

This makes development **fun and fast**.

3. Beautiful UI with Widgets

Flutter's widget system makes creating UI easy:

- Buttons
- Text
- Forms
- Animations
- Custom designs

4. Near-Native Performance

Flutter does **not** use web views or OEM UI. It draws UI directly using the **Skia engine**, making it super fast.

Animations, scroll, interactions → all feel smooth.

5. Strong Community & Google Support

Flutter is backed by Google and used internally in Google products. The

ecosystem is huge:

- Thousands of packages
- Global community
- Constant updates
- Lots of tutorials and learning resources

6. Cost-Effective for Companies

Companies can hire **one Flutter team instead of separate teams** for Android + iOS + Web + Desktop.

This reduces cost by **50–70%**.

7. Perfect for Startups, MVPs, and Production Apps

Flutter is widely used because:

- Fast time to market
- Beautiful apps
- Single codebase
- Easy maintenance

3.2 Companies Using Flutter (Real Examples)

Many global companies use Flutter for production apps:

✔ Google

- Google Ads app
- Google Assistant modules

✔ BMW

- MyBMW mobile app

✔ Alibaba Group

- Xianyu app (huge user base)

✔ eBay

- eBay Motors

✔ Nubank

- Digital bank serving millions

✔ Toyota

- In-car infotainment using Flutter

✔ ByteDance (TikTok parent)

- Multiple internal apps

✔ Philips

- Healthcare applications

✔ Reflectly

- AI journaling

app Flutter is adopted

by:

- Startups
- Enterprises
- Banks
- Automotive companies

- Government apps

4. What Is an IDE?

An **IDE (Integrated Development Environment)** is a **software application** that gives you everything you need to **write code, fix errors, and build apps**—all in one place.

In short:

An IDE is like a smart workspace for programmers.

It combines:

- a code editor
- tools to run your app
- tools to find and fix errors
- tools to manage your files
- tools to install packages

Think of an IDE like a "school bag" where all your study items are in one place.

4.1 Why Do We Need an IDE?

Without an IDE, you would need:

- one app to write code
- another to run it
- another to debug errors
- another for tools (This would be very confusing!)

An IDE gives you **everything in a single window**, making development easier and faster.

4.2 IDE Example: Visual Studio Code (VS Code)

What is VS Code?

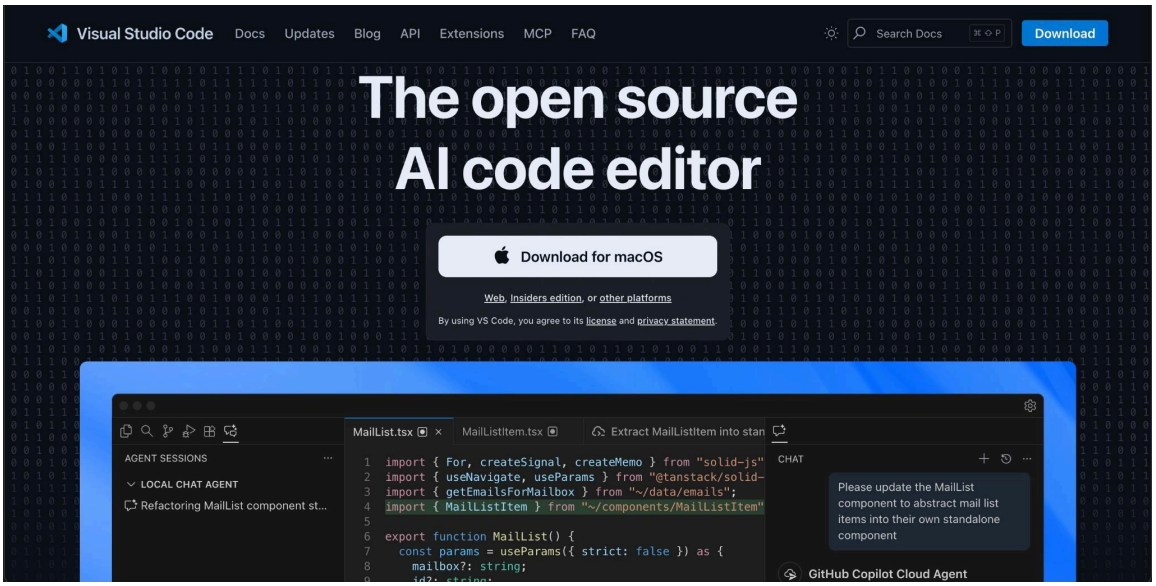
VS Code is a **lightweight, fast, and beginner-friendly IDE** created by Microsoft.

✔ Why beginners love VS Code:

- Very easy to install
- Simple and clean interface
- Supports Flutter with extensions
- Uses fewer system resources
- Fast performance
- Highly customizable

✔ What VS Code offers for Flutter:

- Code editor with syntax colors
- Flutter & Dart extensions
- Device/emulator selection
- Run/Debug buttons
- Integrated terminal
- Auto-suggestions
- Error warnings



You can find vs code in this URL: [vs code link]

IDE Example: Android Studio

What is Android Studio?

Android Studio is a full-featured, heavy-power IDE designed by Google for Android development.

It supports Flutter too, but it is more advanced and uses more RAM/CPU.

✔ Why some developers prefer Android Studio:

- Built-in Android tools (Logcat, Emulator, Profiler)
- Strong debugging tools
- Better for complex or large apps
- Built-in emulator (fast and reliable)

✔ Downsides for absolute beginners:

- Harder to learn
- Heavier on system performance
- Slower on low-end laptops

****Recommendation for beginners:**

👉 Start with ***VS Code*** (use Android Studio only if you need its advanced tools later)

5. Install Flutter In Windows 11

Youtube Link: https://youtu.be/jftxz4SOEqI?si=0djlTxs26pJgNlaI

6. Install VS Code in Windows 11

Youtube link: https://youtu.be/wU7IQLIOwoo?si=Ksx_gK1Xife1gWrL

7. What Is a Widget?

In Flutter, **everything you see on the screen is a widget**.

A **widget** is like a **building block** used to create your app's user interface (UI). Think of

widgets as **LEGO pieces**:

- You combine many small pieces
- To build a big

structure Similarly, in Flutter:

- You combine many widgets
- To build your screen (UI)

7.1 Two Types of Widgets

1. StatelessWidget

A widget that **does not change** once built. Example: Text, Icon, AppBar. Use when:

- UI is fixed
- No data updates

2. StatefulWidget

A widget that **can change** when something happens. Example: counter app, forms, toggles, animations.

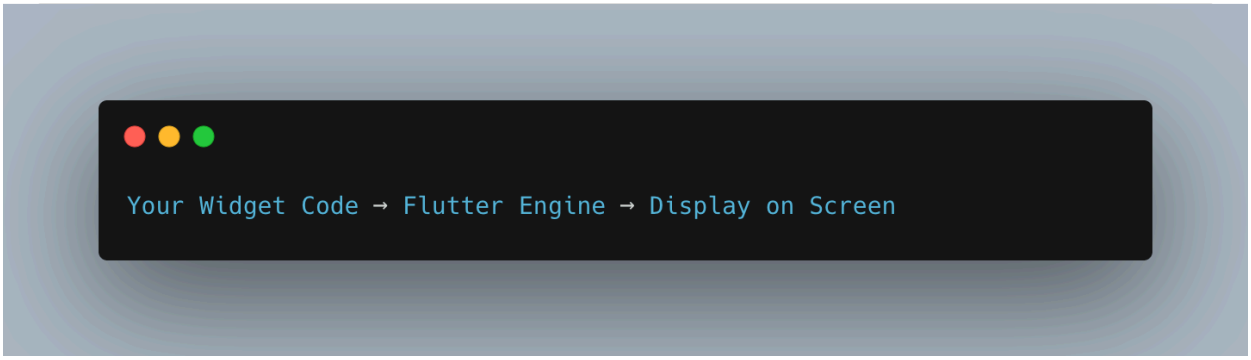
Use when:

- Data changes
- User interaction needed
- UI updates

7.2 How Widgets Work Internally?

Widgets don't draw themselves. Instead, they **describe** how the UI should look. Flutter takes this description and paints it on screen.

Simple flow:



7.3 What Is a Widget Tree?

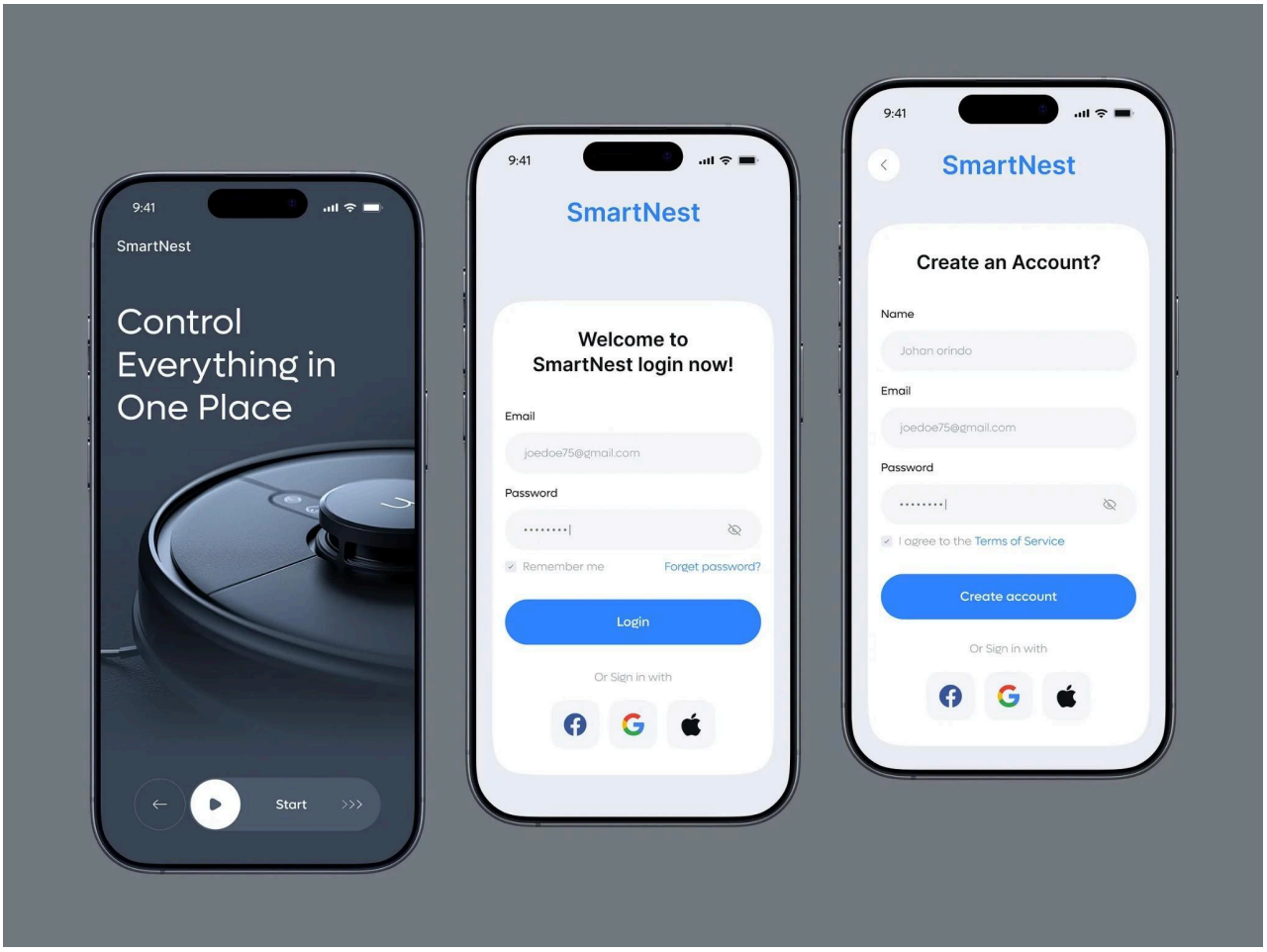
Widgets are arranged inside each other in a tree structure. Example:

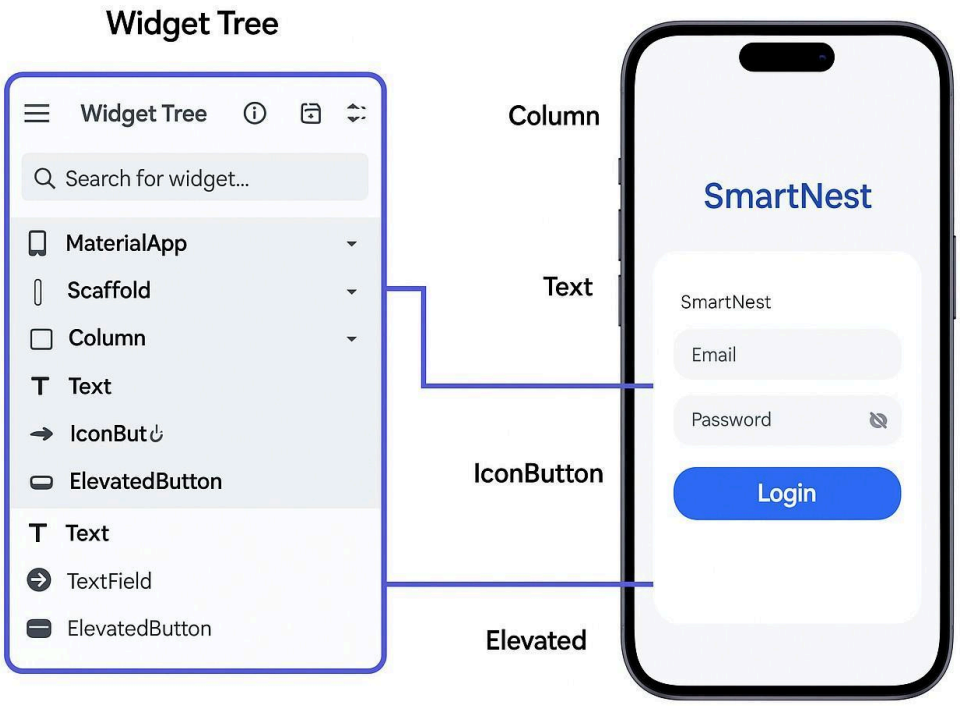
7.4 Why Widgets Are Important?

Because Flutter uses a **widget-based architecture**, meaning:

- Every UI element is a widget
- Every layout is a widget
- Every page is a widget
- Your entire app is a tree of

widgets Learning widgets = learning Flutter.





8. What Is State Management?

8.1 What Is State?

State = any data that can change in your app. Examples:

- The value of a counter
- Whether a password field is visible or hidden
- The text inside a TextField
- Whether a user is logged in or logged out
- A list of items in a cart

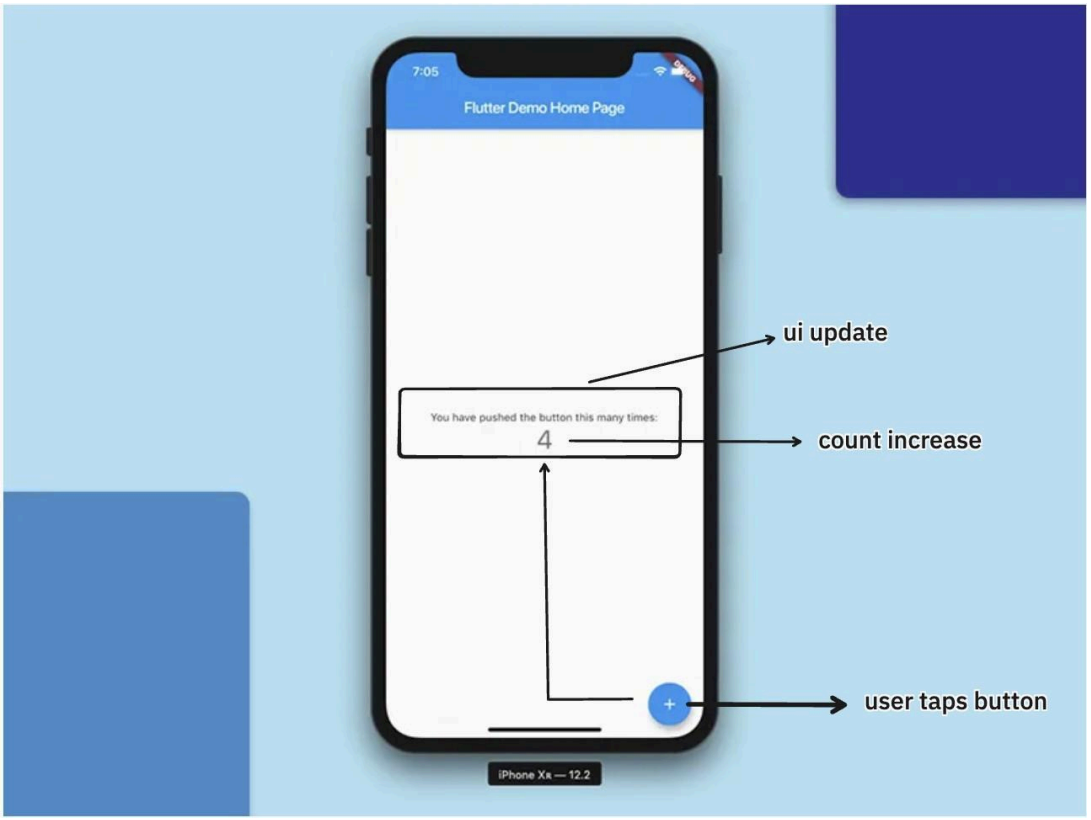
If the data changes, and the UI must update, that is the state.

8.2 What Is State Management?

State Management = the way you control and update changing data in your app.

Flutter rebuilds UI based on the latest state.

The visual representation is added below.



8.3 Why Do We Need State Management?

Because apps are interactive. When a

user:

- types text
- taps buttons
- opens screens
- logs in
- selects items

…the UI must update.

State management helps Flutter understand:

- **WHEN** something changed
- **WHERE** it changed
- **HOW** to rebuild the UI

8.4 Types of State Management in Flutter

There are **two categories**:

Local State (Simple State)

Used inside a single widget or screen.

Example: counter app → changes inside one page. Tools

for local state:

- `setState()` (simplest)
- `ValueNotifier`
- `InheritedWidget`

❑ **App-wide State (Global State)** Used

when multiple screens share data. Example:

- user login info
- cart items
- theme mode
- language

settings Tools:

- Provider (Beginner-friendly)
- Riverpod
- BLoC
- GetX
- MobX

9. Pubspec.yaml & packages

9.1 What is pubspec.yaml?

pubspec.yaml is a small YAML file at the root of every Flutter project. It's the project manifest — it tells Flutter:

- the project name and version
- what external packages your app needs (dependencies)
- what assets (images, fonts) to include
- build and environment settings

Think of it like the project's instruction card: "what to install" and "what to include".

9.2 Adding a package (step-by-step)

1. Find a package on **pub.dev** (the official Dart/Flutter package site).
2. Copy the package name and recommended version (e.g., `provider: ^6.0.5`).

3. Paste it under `dependencies:` in `pubspec.yaml`.
4. Save the file.

Run in terminal (project root): `flutter pub get`

5. This downloads and links the package to your project.

VS Code normally runs `pub get` automatically when you save `pubspec.yaml`.

9.3 How packages appear in code

After adding `provider`:

9.4 Assets (images & fonts)

Put images in a folder like **`assets/images/`**

List them in `pubspec.yaml` under `flutter: assets:` (exact indentation is important).
Example (referencing your uploaded file as an asset):

To use it:

```
Image.asset('assets/images/login_screen.png')
```

9.5 Basic structure (annotated example)

```
name: my_first_app            # project name
description: A simple app
version: 1.0.0+1              # semantic version + build number

environment:
  sdk: ">=2.18.0 <4.0.0"      # Dart SDK version range

dependencies:                 # runtime packages used by your app
  flutter:
    sdk: flutter
  provider: ^6.0.5            # example package (state mgmt)
  shared_preferences: ^2.0.15 # local storage example

dev_dependencies:             # packages used during development or testing
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0

flutter:                      # flutter-specific settings
  uses-material-design: true

  assets:                     # images and other files to bundle
    - assets/images/logo.png
    - assets/images/login_screen.png

  fonts:                      # custom fonts
    - family: Poppins
      fonts:
        - asset: assets/fonts/Poppins-Regular.ttf
```

9.6 Version syntax & compatibility

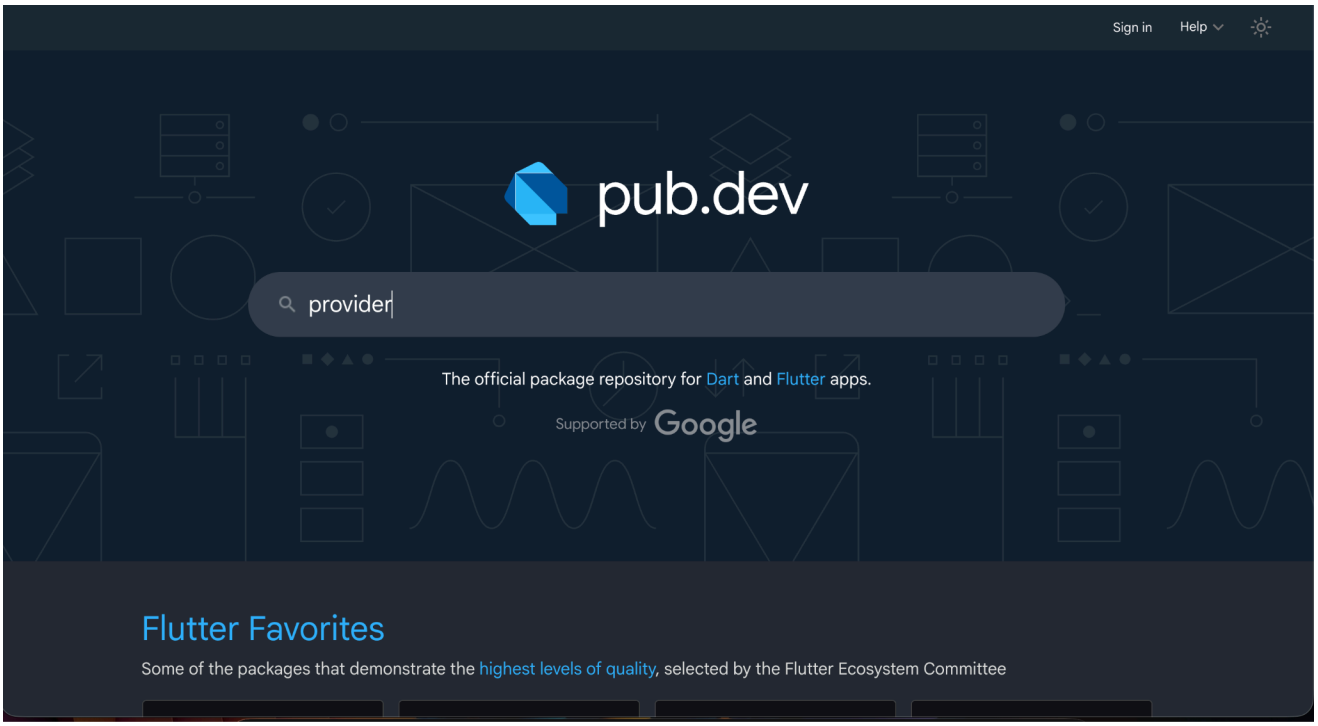
- `^1.2.3` means “compatible with 1.2.3”, i.e., allow non-breaking updates.
- Use semver awareness — avoid forcing `any` unless you understand the consequences.
- If a package requires a newer Dart SDK, update your `environment` section.

9.7 Picking packages — quick rules

Popularity: check likes, pub points, and number of pub.dev scores.

- Maintenance: last published date — avoid abandoned packages.
- Null-safety: prefer null-safe packages (most are now).
- License: ensure license is compatible with your project.
- Read the docs: good README + example code is a sign of quality.

This is the official Flutter repository called [pub.dev.](pub.dev)



9.8 Common pitfalls & fixes

- Indentation errors: YAML is indentation-sensitive. Always use 2 spaces (no tabs).
- Asset not found: check the path, ensure file is in the project, run **`flutter pub get`**, and restart the app if needed.
- Version conflicts: if two packages require different versions, use **`dependency_overrides`** carefully or select compatible versions.
- pub get failed: read the terminal output — it will tell you what is wrong (network, version mismatch, missing asset).
- Package imports fail: ensure you run **`flutter pub get`** and IDE recognizes packages (restart IDE if necessary).

9.9 Useful commands

- `flutter pub get` — download dependencies
- `flutter pub upgrade` — update to latest compatible versions
- `flutter pub outdated` — shows what can be upgraded
- `flutter pub add <package>` — add package from terminal (convenient)

9.10 Good practices

- Keep `pubspec.yaml` tidy and documented.
- Pin only where necessary; use caret `^` for safe updates.
- Store assets in `assets/images/` and fonts in `assets/fonts/`.
- Commit `pubspec.yaml` and `pubspec.lock` to version control (lock ensures repeatable builds).
- Test package upgrades in a branch before merging.

10. Project Structure in Flutter

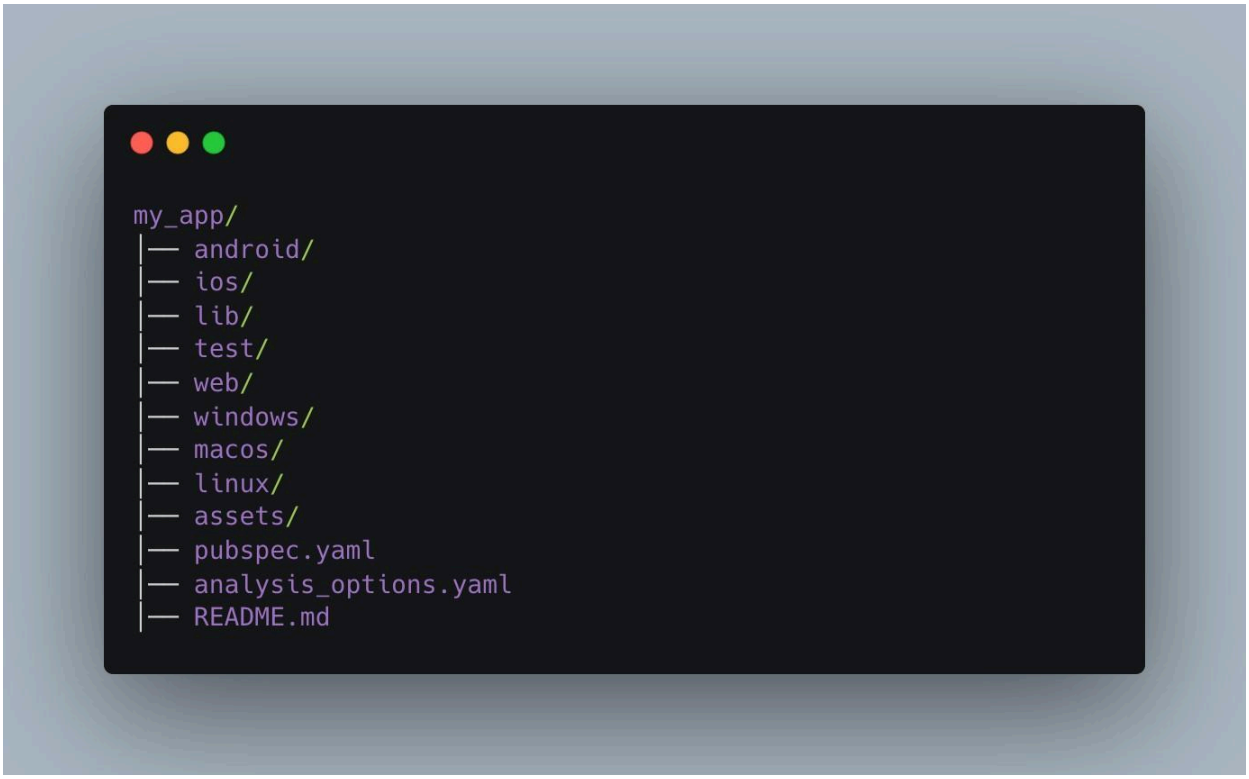
When you create a Flutter project, Flutter automatically generates a set of folders and files. Each folder has a purpose. Understanding these will help you stay organized as your app grows.

Think of your project as a **school bag with separate compartments**:

- one for books
- one for notebooks
- one for pens
- one for lunch

A Flutter project is the same: everything has a place.

10.1 Default Flutter Project Structure



10.2 Project Structure Explained Visually

10.3 Beginner Tips for Clean Project Structure

✔ Don't put all your code in `main.dart`
✔ Separate screens
✔ Create reusable widgets in `/widgets`
✔ Use `/models` for clean data handling
✔ Keep images inside `/assets/images`
✔ Keep API calls inside `/services`
✔ Keep constants inside `/utils`

11. What is frontend, backend, API

The **Frontend** is the **visible part** of an app — everything the user can see and interact with.

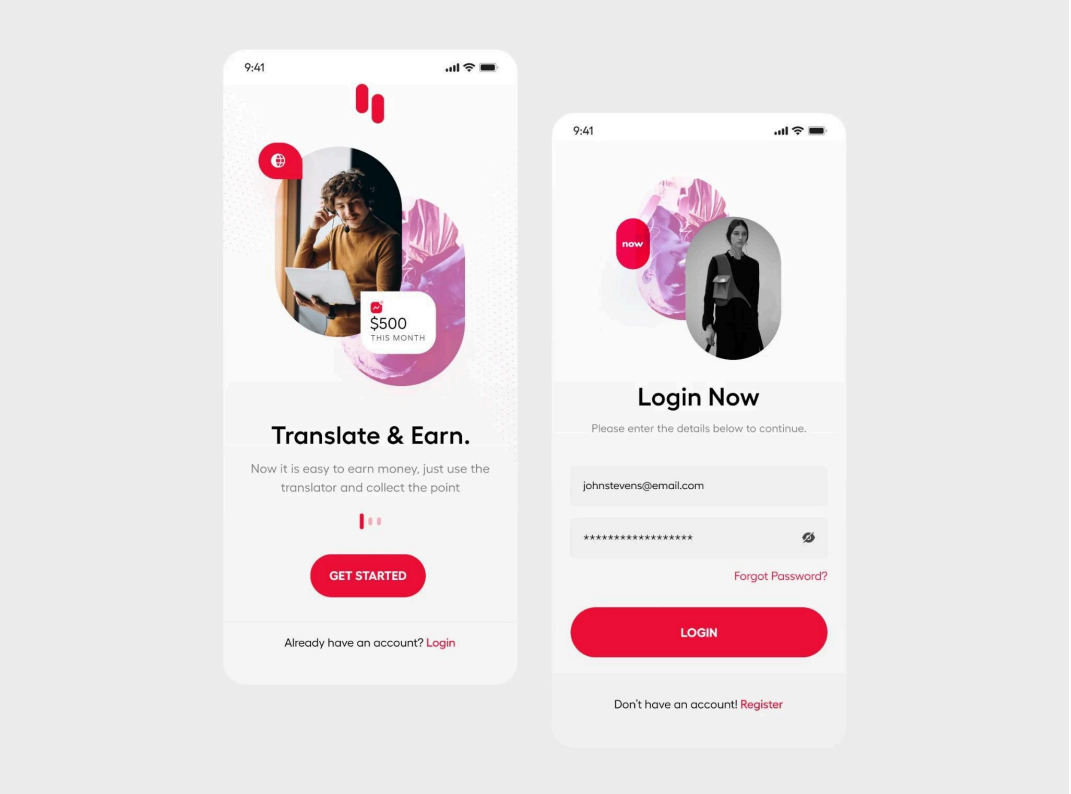
This includes:

- Buttons
- Text
- Images
- Screens
- Forms
- Animations

In Flutter:

Frontend = Widgets, UI, Screens, Layout, Animations.

Simple example:



11.1 What Is Backend? (What happens behind the scenes)

The **Backend** is the **hidden part** of an app — the engine that handles data, logic, and storage.

Users cannot see it, but it does all the heavy work.

Backend does:

- User authentication (login/signup)
- Saving user details
- Fetching products, messages, notifications
- Database operations
- Payment processing
- Sending OTP

Backend examples:

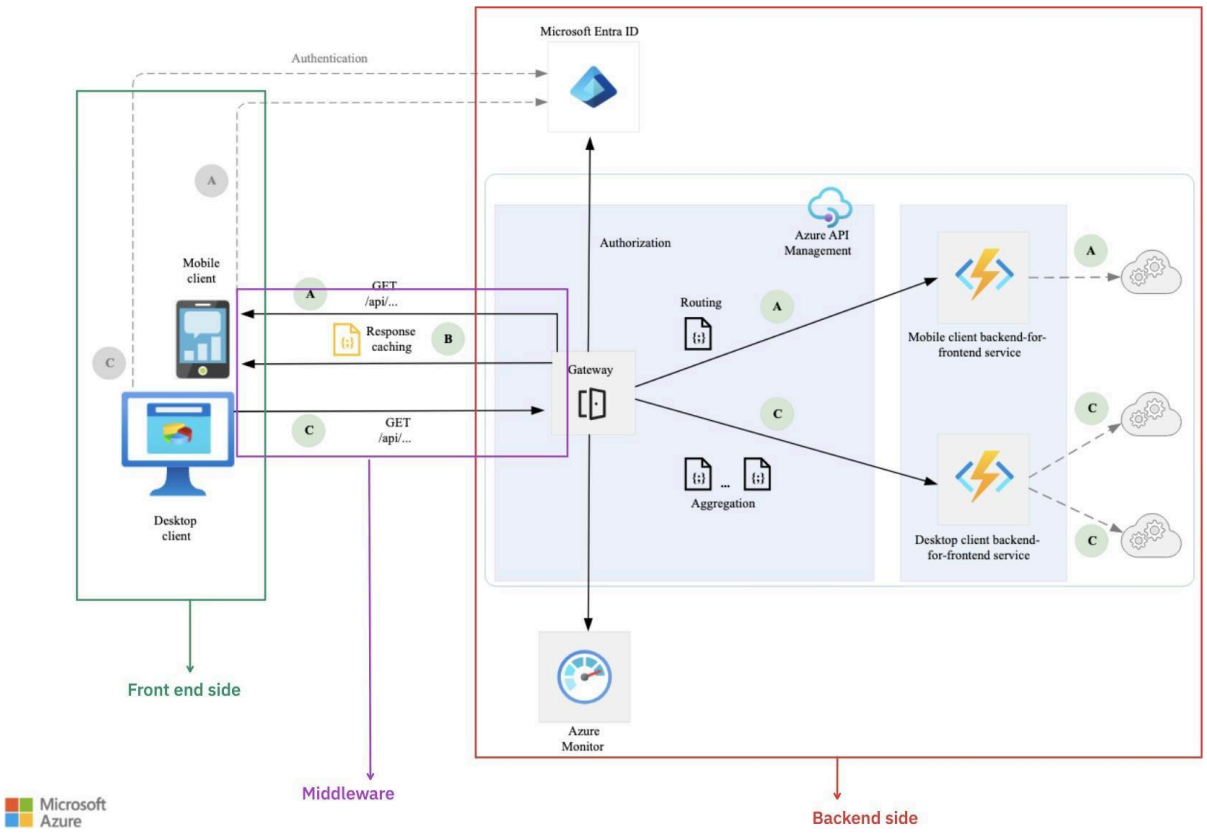
- Firebase
- Node.js
- Python (Django)
- PHP
- Java Spring
- Go

Note: Flutter does NOT have a backend.

Flutter = Frontend only. You connect Flutter to a backend.

11.2 What Is an API? (The connection between Frontend & Backend)

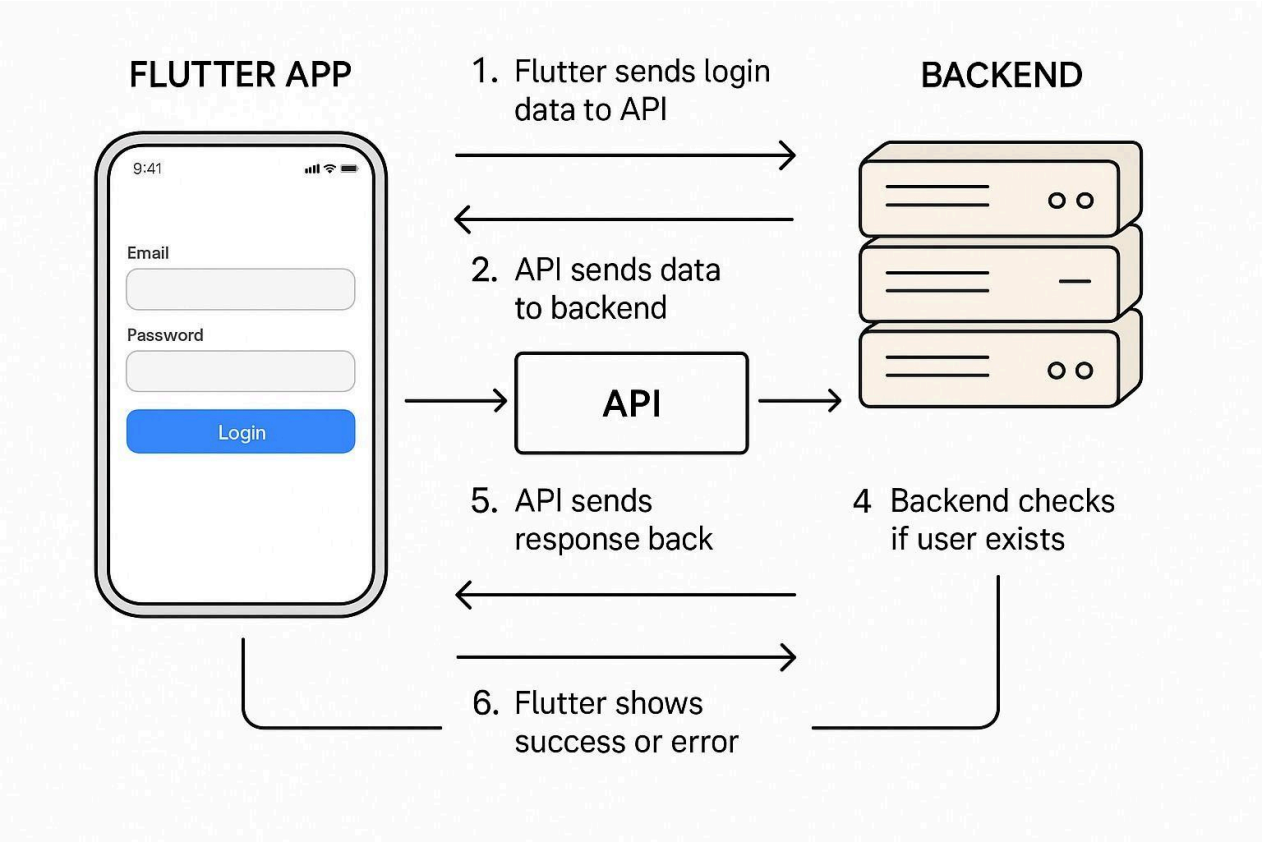
1. API = Messenger between Frontend and Backend.
2. Backend stores data → API sends it to frontend.
3. Frontend sends user input → API gives it to backend.



Real-world example:

User enters email + password → taps *Login*.

1. Flutter sends login data to API
2. API sends data to backend
3. Backend checks if user exists
4. API sends response back
5. Flutter shows success or error

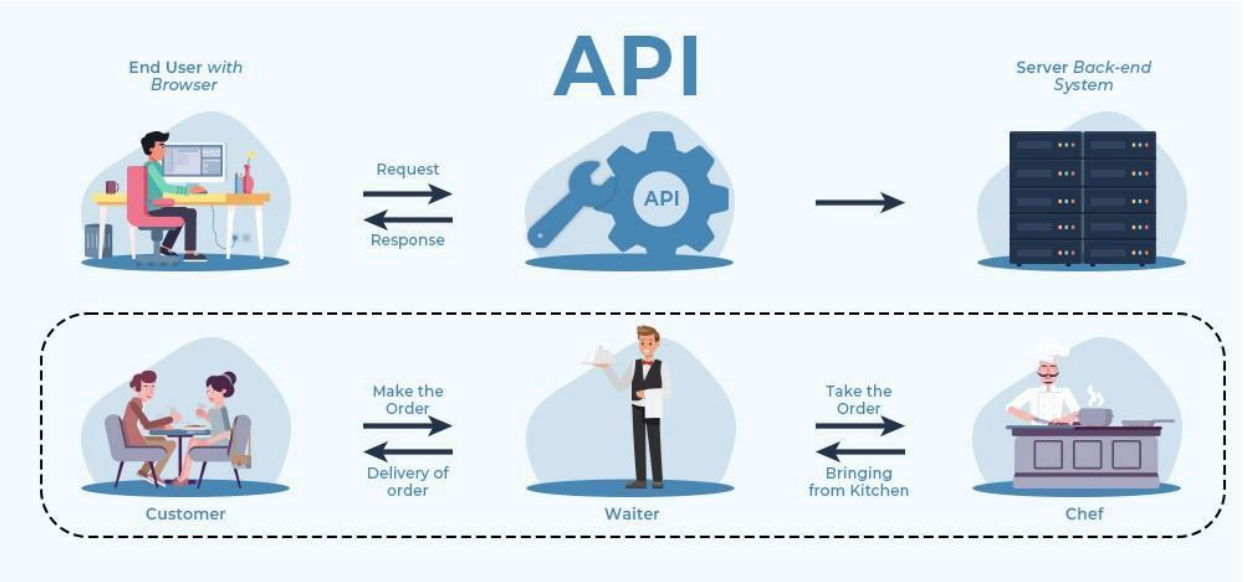


11.3 Simple Analogy

Restaurant Analogy:

- Frontend = The part you see
  Menu, tables, lights, waiters, plates.
- Backend = The kitchen
  Where food is prepared.
- API = The waiter
  Take your order → take it to the kitchen → bring back the food.

The visual representation is added below.



12. Device Emulator & Physical Device

12.1 What Is a Device Emulator?

A Device Emulator (or Simulator on iOS) is a virtual mobile device that runs on your computer.

It behaves like a real phone but exists only inside your laptop/PC.

⭐ **Emulators are used for:**

- Testing your app without a real phone
- Checking UI layouts
- Testing different screen sizes
- Quickly running & debugging code

📌 **Android = Emulator**

📌 **iOS = Simulator**

12.2 Advantages of Emulator

- ✔ No need for a mobile device
- ✔ Multiple screen sizes & models
- ✔ Fast for UI testing
- ✔ Can take screenshots easily
- ✔ Works directly with VS Code/Android Studio

12.3 Disadvantages of Emulator

- ✘ Slower on low-end laptops
- ✘ Drains CPU & RAM
- ✘ May not support all sensors
- ✘ iOS Simulator only works on macOS

12.4 What Is a Physical Device?

A Physical Device = a real Android phone or iPhone.

You connect it to your computer via USB and run your app on it.

⭐ Used for:

- Real-world testing
- Checking performance
- Testing camera, GPS, Bluetooth
- Checking animations & gestures
- Accurate debugging

12.5 Advantages of Physical Device

✔ Real performance
✔ Access to real sensors
✔ Faster & smoother than emulator
✔ More accurate debugging

12.6 Disadvantages of Physical Device

✘ Needs USB cable or WiFi pairing
✘ Must enable Developer Mode
✘ Need device drivers (Android)
✘ Harder to test multiple screen sizes

12.7 When to Use What? (Simple Rule)

✔ Use Emulator for:

- UI design
- Layout testing
- Basic functionality

✔ **Use Physical Device for:**

- Real performance testing
- Animations
- Camera/GPS/Bluetooth
- Production-level testing

13. Where to Deploy Your Flutter App

Once you finish building your app, the final step is **deploying** it — making it available for users to download and use.

Flutter apps can be deployed to **multiple platforms**:

Android (Google Play Store)

✔ **What you need:**

- Google Play Developer Account ($25 one-time fee)
- App Bundle (.AAB file)
- App Icon
- Screenshots
- App description
- Privacy policy

✔ **Steps to deploy:**

1. Create an **AAB** file: `flutter build appbundle`
2. Upload to Google Play Console
3. Add screenshots, description, category
4. Submit for review
5. Google approves & publishes your app

🌍 **Users download your app from: Google Play Store**

2️⃣ iOS (Apple App Store)

✔ **What you need:**

- Apple Developer Account ($99/year)
- A Mac or cloud Mac
- iOS certificate & provisioning
- IPA build
- App screenshots
- Privacy details

✔ **Steps:**

1. Build an iOS release: `flutter build ios`
2. Upload via **Xcode** or **Transporter**
3. Add screenshots and metadata
4. Submit for review

🍎 **Users download from: Apple App Store**

3️⃣ Web Deployment

Flutter can compile into a **website**.

✔ **Build web version:** `flutter build web`

This generates a folder: `/build/web`

You can deploy this to:

- Firebase Hosting
- Netlify
- Vercel
- GitHub Pages
- Any static hosting service

Uses:

- Admin dashboards
- Portfolio apps
- Landing pages
- Simple tools

14. What Is the Future of Flutter?

Flutter has a strong and promising future because it is one of the few technologies that allows you to build mobile, web, desktop, and embedded apps using a single codebase.

Google Actively Supports Flutter

Flutter is developed and maintained by **Google**. Big companies rarely drop projects that have:

- huge community
- millions of users
- active updates
- real businesses depending on it

Flutter receives **frequent releases**, **new features**, and **long-term stability**.

Flutter Is Expanding Beyond Mobile

Earlier Flutter was only for **Android & iOS**. Now

it supports:

- **Web apps**
- **Windows apps**
- **macOS apps**
- **Linux apps**
- **Embedded devices** (cars, TVs, appliances)

This makes Flutter a **unified development platform**.

Companies Using Flutter Are Increasing

Millions of developers + thousands of companies rely on Flutter.

Well-known companies using Flutter:

- Google Ads app
- BMW MyBMW app
- eBay Motors
- Nubank
- Toyota
- Alibaba
- ByteDance

This shows Flutter is **trusted globally**.

4. Fast Performance & Native Feel

Flutter compiles to **native machine code**, making it:

- extremely fast
- smooth for animations
- optimized for modern hardware

In the future, even more performance improvements are expected.

5. Strong Community + Massive Package Ecosystem

Flutter has:

- a huge community
- thousands of packages
- active open-source contributors
- long-term educational resources

Flutter's ecosystem is growing, not shrinking.

6. AI + Flutter = Strong Future

Flutter is increasingly being used with:

- AI models
- ChatGPT integrations
- Google Gemini
- ML Kit
- On-device ML

Flutter + AI = powerful modern apps.

7. Job Opportunities Are Strong

Because Flutter supports multiple platforms, companies reduce development cost by using one team.

This means:

- more companies adopting Flutter
- more Flutter job openings
- higher demand for cross-platform developers

Flutter Is Moving Toward "Write Once, Run Everywhere"

In the future, one Flutter app could run on:

- phones
- tablets
- laptops
- smart TVs
- IoT devices
- cars
- the web

This makes Flutter a **long-term framework** for multi-platform app