Experiment 1

Aim: To install and configure Flutter environment

Theory:

**Flutter**

Flutter is an open-source framework developed by Google for building cross-platform mobile, web, and desktop applications from a single codebase. It uses the Dart programming language and allows developers to create high-performance, visually appealing applications for iOS, Android, and even web platforms, while maintaining a native-like user experience.

Its key features are

 **Single Codebase**: Write one codebase that can run on multiple platforms (iOS, Android, web, desktop).

 **Hot Reload**: Enables developers to see code changes immediately without restarting the app, speeding up the development process.

 **Widgets**: Flutter's UI is built entirely using customizable widgets, making it flexible and easy to create highly interactive and dynamic interfaces.

 **Performance**: Flutter compiles to native ARM code, providing better performance compared to other cross-platform frameworks like React Native.

 **Access to Native Features**: Flutter offers plugins and platform channels to access native features and APIs when needed.

**Dart**

Dart is a programming language developed by Google, primarily used for building web, mobile, and desktop applications. It is the language behind Flutter, making it a key technology for Flutter developers. Dart was designed to be easy to learn, efficient, and highly scalable, with features that support both client-side and server-side development.

Key features of Dart:

1. **Object-Oriented**: Uses classes and objects for structure.
2. **Strongly Typed**: Types are checked at compile-time, ensuring safer code.
3. **JIT and AOT Compilation**: Supports both Just-In-Time and Ahead-Of-Time compilation for fast development and optimized performance.
4. **Concurrency with Isolates**: Uses isolates for safe concurrent programming without shared memory.
5. **Rich Standard Library**: Comes with many built-in libraries for common tasks.
6. **Cross-Platform**: Runs on mobile (via Flutter), web (via JavaScript), and desktop.
7. **Async-Await**: Simplifies asynchronous programming with async/await syntax.
8. **Null Safety**: Prevents null errors by ensuring variables must be initialized or explicitly nullable.

**Usage in Flutter**

Since Flutter is built using Dart, the language is the foundation for creating Flutter applications. Flutter's rich widget system, user interface components, and state management are all defined and manipulated through Dart code.

### ****Widgets in Flutter****

In Flutter, **everything is a widget**. A widget is a fundamental building block of a Flutter app's user interface (UI). Widgets describe the structure, appearance, and behavior of the UI.

#### Key Types of Widgets:

1. **Stateless Widget**:

* A widget that doesn’t change during the lifetime of the widget.
* It is immutable, meaning once created, it cannot change.
* Example: A button or text label that doesn’t need to change after its creation.

*class MyWidget extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return Text('Hello, World!');*

*}*

*}*

1. **Stateful Widget**:

* A widget that can change over time based on user interaction or other factors.
* Stateful widgets have an associated **State** object, which holds the mutable data and can trigger the widget to rebuild when the state changes.

*class CounterWidget extends StatefulWidget {*

*@override*

*\_CounterWidgetState createState() => \_CounterWidgetState();*

*}*

*class \_CounterWidgetState extends State<CounterWidget> {*

*int \_counter = 0;*

*void \_incrementCounter() {*

*setState(() {*

*\_counter++;*

*});*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Column(*

*children: [*

*Text('Counter: $\_counter'),*

*ElevatedButton(*

*onPressed: \_incrementCounter,*

*child: Text('Increment'),*

*),*

*],*

*);*

*}*

*}*

### ****State Management in Flutter****

State management refers to how you manage and control the state (data) of your application, ensuring that when data changes, the relevant UI components are updated. Flutter provides various approaches to manage state, from simple to more complex solutions.

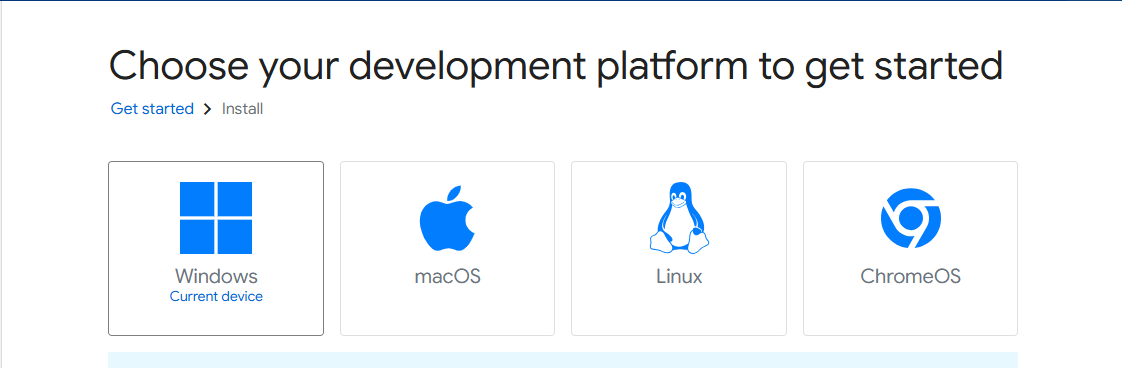
#### Common Approaches to State Management:

1. **setState()**:
   * Simple and most commonly used method for managing state locally within a widget.
   * Calls setState() to notify the framework to rebuild the widget with new data.
   * Best for small apps or components with minimal state requirements.
2. **InheritedWidget**:
   * A more advanced way to share state across multiple widgets in the widget tree.
   * Can be used for data that needs to be accessed by multiple child widgets.
   * More complex than setState(), but can be useful for state propagation across a large widget tree.
3. **Provider**:
   * A popular and powerful package for state management in Flutter.
   * Provider makes it easy to pass and manage data throughout the app by using a single source of truth.
   * It allows you to create shared state and updates UI efficiently when data changes

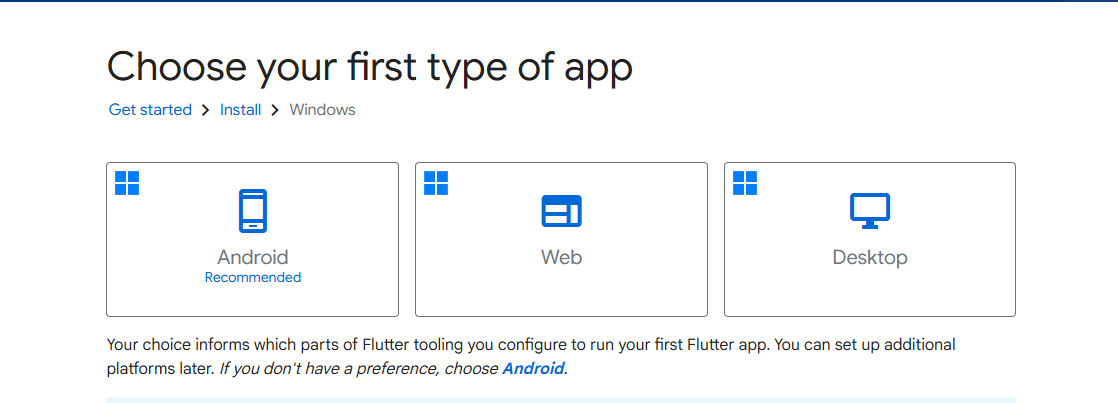
Installation of Flutter

Step 1: Go and visit website <https://docs.flutter.dev/get-started/install>

Step 2: Select your platform



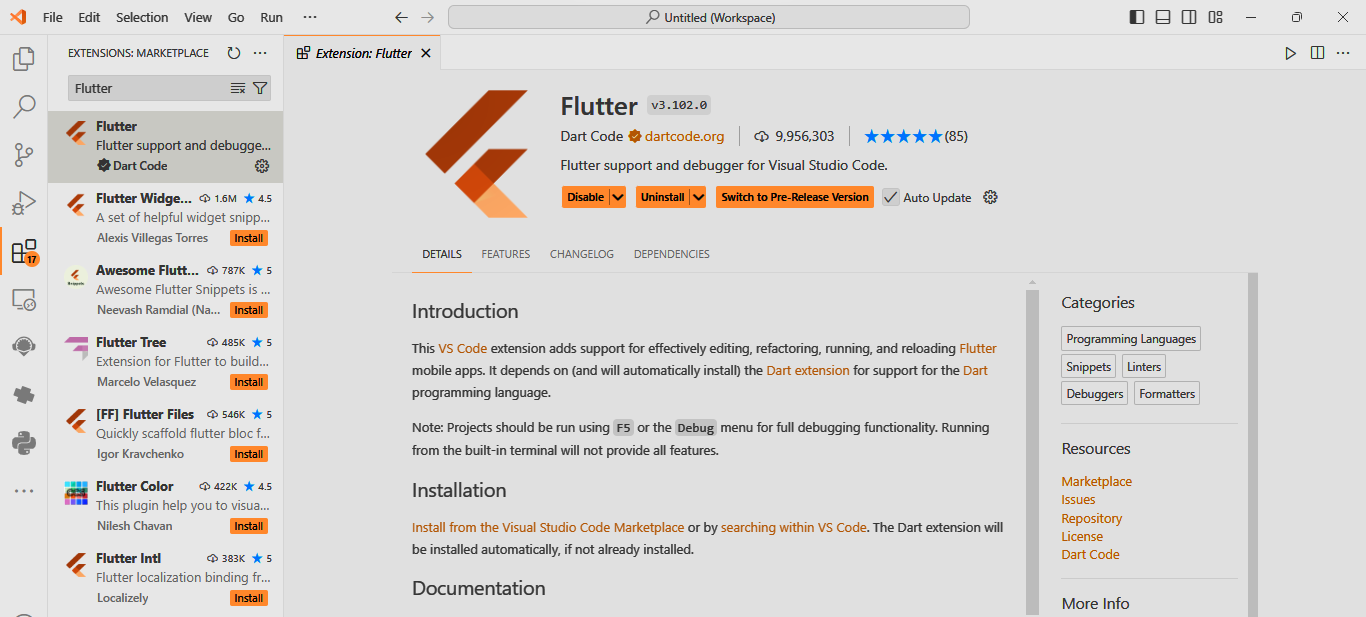
Step 3: Choose platform for which you are developing your application



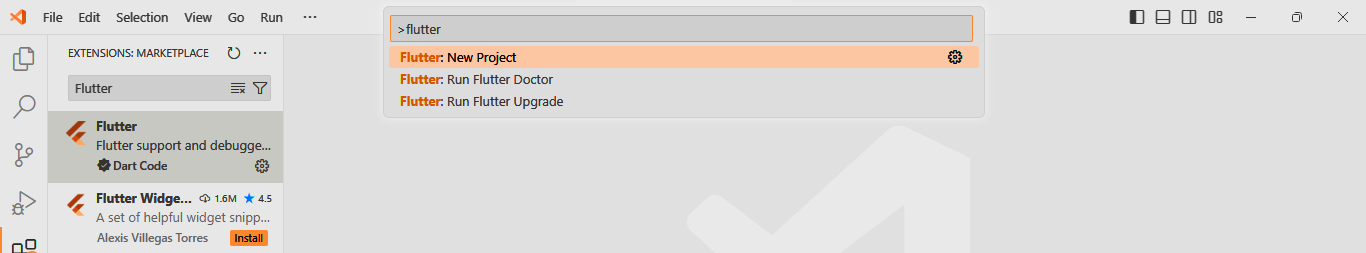
Step 4: Verify that your PC meets the software & hardware requirements

Step 5: Download Git and Android Studio if you don’t have it installed, it is a perquisite for Flutter

Step 6: Choose your IDE for installation, we have chosen Visual Studio Code here



Step 7: Download Flutter SDK using Visual Studio Code



Step 8: Click Download SDK, and select location for download

