SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Unit-name : Introduction to Flutter

- Topics : Overview of Flutter,
  History and evolution of Flutter,
  Apps that are build using Flutter
  Feature of Flutter
  Advantages of Flutter
  Disadvantages of Flutter
  Flutter Installation in Windows
  Architecture of Flutter Applications

Overview of Flutter

- Flutter is framework created by **Google.**
- A cross-platform framework used to develop application for
  - Android
  - iOS
  - Web
  - Desktop

  Dart Programming language is used for that.

Why Flutter for App Development?

- Flutter is Open-Source framework Therefore; anyone can use it for any given purpose.
- Flutter is so fast that it takes less than 30 sec for first full compilation.
- Comes with Hot-Reload & Hot-Restart.
- Due to Stateful widget hot-reload feature Flutter is very fast iterative coding style.
- Ready to use widget, Flutter has many widgets that you can use to build flutter application.
- Such as: http, get, share plus, toggle switch etc.
- Flutter Community is bit small if we compare with other framework like React.
- But Flutter is growing very fast then other framework.
- Anyone who have basic knowledge of OOPS & UI Designing can easily learn Flutter.

Apps that are build using Flutter

- Dream 11
- eBay
- Stadia gaming platform
- Google Pay

Feature of Flutter

- Open-Source: Flutter is a free and open-source framework for developing mobile applications.
- Cross-platform: This feature allows Flutter to write the code once, maintain, and can run on different platforms. It saves the time, effort, and money of the developers. Minimal Coding.
- Hot Reload: Whenever the developer makes changes in the code, then these changes can be seen instantaneously with Hot Reload. It means the changes immediately visible in the app itself. It is a very handy feature, which allows the developer to fix the bugs instantly. Hot Reload & Hot Restart
- Accessible Native Features and SDKs: This feature allows the app development process easy and delightful through Flutter's native code, third-party integration, and platform APIs. Thus, we can easily access the SDKs on both platforms.
- Minimal code: Flutter app is developed by Dart programming language, which uses JIT and AOT compilation to improve the overall start-up time, functioning and accelerates the performance. JIT enhances the development system and refreshes the UI without putting extra effort into building a new one.
- Widgets: The Flutter framework offers widgets, which are capable of developing customizable specific designs. Most importantly, Flutter has two sets of widgets: Material Design and Cupertino widgets that help to provide a glitch-free experience on all platforms.

Advantage of Flutter

- It makes the app development process extremely fast because of the hot-reload feature. This feature allows us to change or update the code are reflected as soon as the alterations are made.
- It provides the smoother and seamless scrolling experiences of using the app without much hangs or cuts, which makes running applications faster in comparison to other mobile app development frameworks.
- Flutter reduces the time and efforts of testing. As we know, flutter apps are cross-platform so that testers do not always need to run the same set of tests on different platforms for the same app.
- It has an excellent user interface because it uses a design-centric widget, high-development tools, advanced APIs, and many more features.
- It is similar to a reactive framework where the developers do not need to update the UI content manually.
- It is suitable for MVP (Minimum Viable Product) apps because of its speedy development process and cross-platform nature.

Disadvantages of Flutter

- The Flutter is a comparatively new language that needs continuous integration support through the maintenance of scripts.
- It provides very limited access to SDK libraries. It means a developer does not have a lot of functionalities to create a mobile application. Such types of functionalities need to be developed by the Flutter developer themselves.
- The Flutter apps do not support the browser. It only supports Android and iOS platforms.
- It uses Dart programming for coding, so a developer needs to learn new technologies. However, it is easy to learn for developers.

Flutter Installation in Windows

System requirements for Windows

- **Operating System** Windows 7 or Later (You can also use Mac or Linux OS.).
- **Disk Space**400 MB (It does not include disk space for IDE/tools).
- **Tools**

  1. Windows PowerShell
  2. Git for Windows 2.x (Here, Use Git from Windows Command Prompt option).

- **SDK** Flutter SDK for Windows **IDE** Android Studio (Official).

- Install Git
  Step 1: To download Git,
  Step 2: Run the .exe file to complete the installation. During installation, make sure that you have selected the recommended option.
- Install the Flutter SDK
  Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website, click on Get started button; you will get the following screen.
  Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.
  Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, D: /Flutter.
  Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:
  Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.
  Step 4.2: Now, select path -> click on edit. The following screen appears.
  Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.
  Step 5: Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.
  $ flutter doctor
  Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required running Flutter as well as the development tools that are available but not connected with the device.
  Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.
  Step 7.1: Download the latest Android Studio executable or zip file from the official site.
  Step 7.2: When the download is complete, open the **.exe** file and run it. You will get the following dialog box.
  Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.
  Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.
  Note:> Meanwhile, the installation wizard also includes downloading Android SDK components that are required by Flutter for development.
  Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.
  Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and

select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.
Step 8.2: Choose your device definition and click on Next
Step 8.3: Select the system image for the latest Android version and click on Next.
Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.
Step 8.5: Last, click on the icon pointed into the red color rectangle.
The Android emulator displayed as below screen.
Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.
Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.
Step 9.3: Restart the Android Studio.

System Requirements for macOS

2.8 GB (It does not include disk space for IDE/tools).

Operating System: macOS (64-bit).
Disk Space: 3.5 Gb
Tools: Git, IDE (Xcode includes git)
• Get the Flutter SDK

Step 1: Download the installation bundle of the Flutter Software Development Kit for macOS. To download Flutter SDK, Go to its official website.
Step 2: When your download is complete, extract the zip file and place it in the desired installation folder or location.
Step 3: To run the Flutter command, you need to update the system path to include the flutter bin directory.

```
• $ export PATH="$PATH:`pwd`/flutter/bin"
```

Step 4: Next, enable the updated path in the current terminal window using the below command and then verify it also.

```
• ~/.bashrc source $HOME/.bash_profile echo $PATH
```

Step 5: Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.
`

```
$ flutter doctor
```

Step 6: When you run the above command, it will analyze the system and the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.
Step 7: Install the latest Xcode tools if reported by the Flutter doctor tool.
Step 8: Install the latest Android Studio and SDK, if reported by the Flutter doctor tool.
Step 9: Next, you need to set up an iOS simulator or connect an iPhone device to the system for developing an iOS application.
Step 10: Again, set up an android emulator or connect an android device to the system for developing an android application.
Step 11: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself.

Architecture of Flutter Applications:

• The Flutter architecture mainly comprises of four components.

1. Flutter Engine
2. Foundation Library
3. Widget
4. Design Specific Widget

1.Flutter Engine :
It is a portable runtime for high-quality mobile apps and primarily based on the C++ language. It implements Flutter core libraries that include animation and graphics, file and network I/O, plugin architecture, accessibility support, and a dart runtime for developing, compiling, and running Flutter applications. It takes Google's open-source graphics library, **Skia()**, to render low-level graphics.

2.Foundation Library :

It contains all the required packages for the basic building blocks of writing a Flutter application. These libraries are written in Dart language.

3.Widgets :

In Flutter, everything is a widget, which is the core concept of this framework. Widget in the Flutter is basically a user interface component that affects and controls the view and interface of the app. It represents an immutable description of part of the user interface and includes graphics, text, shapes, and animations that are created using widgets. The widgets are similar to the React components.

In Flutter, the application is itself a widget that contains many sub widgets. It means the app is the top-level widget, and its UI is build using one or more children widgets, which again includes sub child widgets. This feature helps you to create a complex user interface very easily.

We can understand it from the hello world example created in the previous section. Here, we are going to explain the example with the following diagram.

4. Design Specific Widgets:
The Flutter framework has two sets of widgets that conform to specific design languages. These are Material Design for Android application and Cupertino Style for IOS application.

• Gestures:
It is a widget that provides interaction (how to listen for and respond to) in Flutter using GestureDetector. GestureDector is an invisible widget, which includes tapping, dragging, and scaling interaction of its child widget. We can also use other interactive features into the existing widgets by composing with the GestureDetector widget.

• State Management:
Flutter widget maintains its state by using a special widget, StatefulWidget. It is always auto re-

rendered whenever its internal state is changed. The re-rendering is optimized by calculating the distance between old and new widget UI and render only necessary things that are changes.

• Layers
Layers are an important concept of the Flutter framework, which are grouped into multiple categories in terms of complexity and arranged in the top-down approach. The topmost layer is the UI of the application, which is specific to the Android and iOS platforms. The second topmost layer contains all the Flutter native widgets. The next layer is the rendering layer, which renders everything in the Flutter app. Then, the layers go down to Gestures, foundation library, engine, and finally, core platform-specific code. The following diagram specifies the layers in Flutter app development.

Flutter First Application

• To create Flutter application, do
the following steps:

Step 1: Open the Android Studio.

Step 2: Create the Flutter project. To create a project, go to File-> New->New Flutter Project. The following screen helps to understand it more clearly.
Step 3: In the next wizard, you need to choose the Flutter Application.
For this, select Flutter Application-> click Next, as shown in the below screen.
Step 4: Next, configure the application details as shown in the below screen and click on the Next button.

• **Project Name:** Write your Application Name.
• **Flutter SDK Path:** <path_to_flutter_sdk>
• **Project Location:** <path_to_project_folder>
• **Descriptions:** <A new Flutter hello world application>.

Step 5: In the next wizard, you need to set the company domain name and click the Finish button. After clicking the Finish button, it will take some time to create a project. When the project is created, you will get a fully working Flutter application with minimal functionality.
Step 6: Now, let us check the structure of the Flutter project application and its purpose. In the below image, you can see the various folders and components of the Flutter application structure, which are going to discuss here.

• **.idea**: This folder is at the very top of the project structure, which holds the configuration for Android Studio. It doesn't matter because we are not going to work with Android Studio so that the content of this folder can be ignored.

• **.android**: This folder holds a complete Android project and used when you build the Flutter application for Android. When the Flutter code is compiled into the native code, it will get injected into this Android project, so that the result is a native Android application. For Example: When you are using the Android emulator, this Android project is used to build the Android app, which further deployed to the Android Virtual Device.

• **.ios**: This folder holds a complete Mac project and used when you build the Flutter application for iOS. It is similar to the android folder that is used when developing an app for

Android. When the Flutter code is compiled into the native code, it will get injected into this iOS project, so that the result is a native iOS application. Building a Flutter application for iOS is only possible when you are working on macOS.

• **.lib**: It is an essential folder, which stands for the library. It is a folder where we will do our 99 percent of project work. Inside the lib folder, we will find the Dart files which contain the code of our Flutter application. By default, this folder contains the file main.dart, which is the entry file of the Flutter application.

• .**test**: This folder contains a Dart code, which is written for the Flutter application to perform the automated test when building the app. It won't be too important for us here.
We can also have some default files in the Flutter application. In 99.99 percent of cases, we don't touch these files manually. These files are:

• .**gitignore**: It is a text file containing a list of files, file extensions, and folders that tells Git which files should be ignored in a project. Git is a version-control file for tracking changes in source code during software development Git.

• .**metadata**: It is an auto-generated file by the flutter tools, which is used to track the propertiesof the Flutter project. This file performs the internal tasks, so you do not need to edit the content manually at any time.

• .**packages**: It is an auto-generated file by the Flutter SDK, which is used to contain a list of dependencies for your Flutter project.

• **flutter_demoapp.iml**: It is always named according to the Flutter project's name that contains additional settings of the project. This file performs the internal tasks, which is managed by the Flutter SDK, so you do not need to edit the content manually at any time.

• **pubspec.yaml**: It is the project's configuration file that will use a lot during working with the Flutter project. It allows you how your application works. This file contains:
• Project general settings such as name, description, and version of the project. • Project dependencies.
• Project assets (e.g., images).

• **pubspec.lock**: It is an auto-generated file based on the .yaml file. It holds more detail setup about all dependencies.

• **README.md**: It is an auto-generated file that holds information about the project. We can edit this file if we want to share information with the developers.

Step 7: Open the main.dart file and replace the code with the following code snippets.

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyHomePage());
}

class MyHomePage extends StatelessWidget {
  final String title;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
             home: Scaffold(
             body: Center(
                 child:
             Text(
              'Hello World',
            ),
      ),
    );
  }
}
```

Step 8: Let us understand the above code snippet line by line.

• To start Flutter programming, you need first to import the Flutter package. Here, we have imported a Material package. This package allows you to create user interface according to the Material design guidelines specified by Android.

• The second line is an entry point of the Flutter applications similar to the main method in other programming languages. It calls the runApp function and pass it an object of MyApp The primary purpose of this function is to attach the given widget to the screen.

• Line 5 to 18 is a widget used for creating UI in the Flutter framework. Here, the StatelessWidget does not maintain any state of the widget. MyApp extends StatelessWidget that overrides its build The build method is used for creating a part of the UI of the application. In this block, the build method uses MaterialApp, a widget to create the root level UI of the application and contains three properties - title, theme, and home.

• **Title:** It is the title of the Flutter application.

• **Theme:** It is the theme of the widget. By default, it set the blue as the overall color of the application.

• **Home:** It is the inner UI of the application, which sets another widget (MyHomePage) for the application.

- the **MyHomePage** is similar to **MyApp**, except it will return

- the **Scaffold** widget is a top-level widget after the MaterialApp widget for creating the user interface. This widget contains two

  properties **appBar** and **body**. The appBar shows the header of the app, and body property shows the actual content of the application.
- Here, **AppBar** render the header of the application, **Center** widget is used to center the child widget, and **Text** is the final widget used to show the text content and displays in the center of the screen.

Step 9: Now, run the application. To do this, go to Run->Run main.dart, as shown in the below screen.
Step 10: Finally, you will get the output as below screen.