Unit - 1

Introduction To Mobile Applications

1. Define Flutter?

Flutter is an open source framework or tool chain developed and managed by Google itself. So because of support provided by Google Developer teams.

2. List Features of Flutter ?

1. Single Codebase: With Flutter, developers can write code once and deploy it across multiple platforms, including iOS, Android, web, and desktop.

2. Hot Reload: Flutter's hot reload feature allows developers to instantly view changes made to the code without restarting the application, making the development process faster and more efficient.

3. Rich Widget Library: Flutter provides a rich set of customizable widgets that enable developers to create beautiful and highly responsive user interfaces.

4. High Performance: Flutter apps are compiled directly to native machine code, resulting in high performance and smooth animations.

5. Native Look and Feel: Flutter allows developers to build apps with a native look and feel on each platform, ensuring a consistent user experience across different devices.

6. Access to Native Features: Flutter provides plugins that allow developers to access native features and APIs, enabling integration with device-specific functionalities.

7. Strong Community Support: Flutter has a large and active community of developers who contribute to its growth by sharing resources, tutorials, and plugins.

3. List advantages and disadvantages of Flutter?

Advantages -

- Faster Development
- Cross-Platform Compatibility
- Beautiful UIs
- High Performance
- Access to Native Features

Disadvantages -

- Large app size
- Limit Libraries
- Learning curve
- Platform specific limitations

4. What is flutter SDK ?

The Flutter SDK (Software Development Kit) is a framework developed by Google for building cross-platform mobile applications.

5. What is Android Studio ?

Android Studio is the official Integrated Development Environment (IDE) for android application development. Android Studio provides more features that enhance our productivity while building Android apps.

6. List features of Android Studio ?

- It has a fast and feature-rich emulator for app testing.
- Apply changes to the resource code of our running app without restarting the app.
- Android Studio provides extensive testing tools and frameworks.

- It supports C++ and NDK.
- It provides build-in supports for Google Cloud Platform.
- It makes it easy to integrate Google Cloud Messaging and App Engine

7. Write simple Hello World Program in Flutter and explain each line of code ?

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

void main()

{

        runApp(MyApp());

}

class MyApp extends StatelessWidget

{

        @override Widget

        build(BuildContext context)

        {

                return MaterialApp(home: Center( child: Text('Hello World!'),),

        );

}
```

import 'package:flutter/material.dart';

This line imports the core Flutter libraries containing essential widgets like MaterialApp, Center, and Text.

void main() { ... }

This is the entry point of your application. The runApp function is called here, which takes a Widget as an argument.

class MyApp extends StatelessWidget { ... }

This defines a class named MyApp that extends the StatelessWidget class. Flutter uses widgets to build

the user interface. A StatelessWidget doesn't hold any state that can change.

@override

This annotation indicates that we're overriding a method inherited from the parent class.

Widget build(BuildContext context) { ... }

This method is responsible for building the UI of your app. It takes a BuildContext object as input, which provides information about the widget tree.

return MaterialApp(...);

This line returns a MaterialApp widget, which is the foundation for most Flutter apps using Material Design.

home: Center(...);

The home property specifies the widget that will be displayed as the app's home screen. Here, we use a Center widget to center the content on the screen.

child: Text('Hello World!');

The Center widget has a child property that defines the widget displayed in the center. In this case, we use a Text widget to display the message "Hello World!" on the screen.

8. Explain Hot reload and its benefits ?

A hot reload is a great functionality present in a flutter. It is the easiest and the fastest function which helps you to apply changes, fix bugs, creating UIs, and add features.

It takes approximately one second to perform its functionality. In hot reload it does not destroy the preserved state. But you cannot use a hot reload once the app gets killed.

Benefits -

1. Fast Iteration: Allows for rapid testing of UI changes, bug fixes, and new features.

2. State Preservation: Keeps the current state of the app intact, so there's no need to navigate back to the point you were testing.

3. Efficient Debugging: Provides an immediate feedback loop, which helps in quickly identifying and resolving issues.

How it works?

- When changes are made to a Flutter app, the Dart compiler rebuilds only the modified parts of the widget tree.
- The updated code is then injected into the running Dart Virtual Machine (VM). The Flutter framework re-renders the modified widgets while preserving the app's current state.

9. Differences between Hot Reload and Hot Restart ?

Hot Reloat :

- It performs very fast as compared to hot restart or default restart of flutter.
- If we are using the state in our app then hot reload will not change the state of the app.
- We perform hot reload by using key ctrl+\.

Hot Restart :

- It is slower than hot reload but faster than the default restart.
- It doesn't preserve the state of our it starts from the initial state of our app.
- We perform hot restart using ctrl+shift+\.

10. Explain Basic Flutter Application Structure ?

>hello_app

>android **[hello_app_android]**

>ios

>lib

main.dart

↓ test

widget_test.dart

.gitignore

.metadata

.packages

hello_app.iml

pubspec.lock

.pubspec.yaml

README.md

↓ External libraries

> Dart Packages

> Dart SDK

>Flutter for Android

Scratches and Consoles

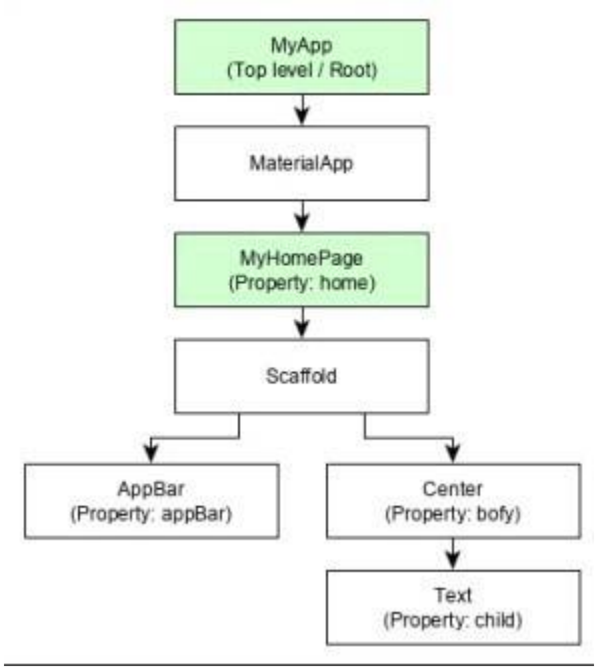
Explaination -

- **android –** Auto generated source code to create android application
- **ios –** Auto generated source code to create ios application
- **lib –** Main folder containing Dart code written using flutter framework

  ib/main.dart – Entry point of the Flutter application
- **test –** Folder containing Dart code to test the flutter application

  test/widget_test.dart – Sample code

- **.gitignore –** Git version control file
- .**metadata –** auto generated by the flutter tools
- .**packages –** auto generated to track the flutter packages
- .**iml –** project file used by Android studio
- **pubspec.yaml –** Used by Pub, Flutter package manager
- **pubspec**.**lock** – Auto generated by the Flutter package manager, Pub
- **README.md –** Project description file written in Markdown format



11. Define MaterialApp?

MaterialApp is a class in the Flutter framework that provides a default implementation of Material Design

12. Define Scaffold ?

Scaffold is a class in flutter which provides many widgets or we can say APIs like Drawer, Snack-

Bar, Bottom-Navigation-Bar, Floating-Action-Button, AppBar, etc.

13. Define Online Sandboxes? List benefits

Online sandboxes are web-based development environments that allow you to write, run, and share code directly from your browser without needing to install any software on your local machine.

Benefits -

1. **Accessibility:** Accessible from any device with an internet connection.
2. **No Setup Required:** Eliminates the need for local environment setup.
3. **Ease of Use:** Simplifies the process of starting new projects.
4. **Collaboration:** Enables easy sharing and collaborative coding.
5. **Cost-Effective:** Often free or lower cost than traditional development environments.

Examples -

- FlutLab.io
- DartPad
- Zapp.run
- Repl.it (with Dart/Flutter support)

14. Define Flutlab.io ? Explain how it is different from other sandboxes?

FlutLab.io is an online IDE and sandbox specifically designed for Flutter development.

It allows developers to create, test, and debug Flutter applications directly in the browser.

How FlutLab.io Differs from Other Sandboxes -

1. Flutter-Specific:

Optimized specifically for Flutter, providing a more tailored experience.

2. Rich Features:

Includes a full-featured code editor, real-time web emulator, and hot reload support.

3. Integration:

Seamless integration with other tools and services tailored for Flutter development.

Features of Flutlab.io -

- Real-time collaboration
- Web Emulator
- Hot Reload
- Project Sharing

Limitation of Flutlab.io -

- Performance
- Internet Dependancy
- Limited Resources

Unit - 2

Dart Programming Language Basics

1. Define Dart?

Dart is an open-source, general-purpose, object-oriented programming language with C-style syntax developed by Google.

2. Explain variables in Dart ?

Variable is used to store the value and refer the memory location in computer memory. In Dart, variables can be declared using 'var', 'final', or 'const'.

Using 'var' :

The `var` keyword is used when you want the compiler to infer the type of the variable based on the assigned value.

Example -

```
void main() {

        var name = 'abc xy';

        var age = 30;

        var height  = 8.4;

}
```

Using 'const' :

The const keyword is used to declare constants.

Syntax -

```
const variable_name;

OR

const data_type variable_name;
```

Using 'final' :

The final keyword is also used to define constants. Here the value is set at run time and can be different each time the program is run.

Syntax -

```
final variable_name;
```

OR

```
final data_type variable_name;
```

3. What is the difference between final and const ?

- **`final`:** The value is set at runtime and can be different each time the program is run.
- **`const`:** The value is set at compile-time and remains constant.

4. List and Explain Data types in Dart Language ?

1. Number

The Darts Number is used to store the numeric values.

The number can be two types - integer and double.

- **Integer -** Integer values represent the whole number or non-fractional values

  ```
  int marks = 80;
  ```

- **Double -** Double value represents the information (double-precision) for floating number

  ```
  double pi = 3.14;
  ```

2. Strings -

Strings are a sequence of characters enclosed in single or double quotes.

String interpolation allows you to include variables in a string.

Example -

```
void man() {
        String hello = "Hello world";
```

```
        print('This is $hello');
}
```

3. Boolean -

The Boolean data type represents Boolean values true and false.

Dart uses the bool keyword to represent a Boolean value.

Example -

```
void main() {
        bool isVisible = true;
        print(isVisible);
}
```

5. Explain operators in Dart ?

In Dart programming, operators are specialized symbols that perform operations on values (operands).

They provide a concise and readable way to manipulate data and control program flow, forming the backbone of any Dart program.

Types of operators in dart:

1. Arithmetic Operators -

Arithmetic operators are used to perform basic operations such as multiplication, division, substraction, addition, etc.

Operators - +, -, *, /, %

//Write basic example

2. Relational Operators -

This class of operators contain those operators which are used to perform relational operation on the operands.

Operators - ==, !=, >, <, >=, <=

//Write basic example

3. Logical Operators -

This class of operators contain those operators which are used to logically combine two or more conditions of the operands.

Operators - &&, || , !

//Write example

4. Assignment Operators -

Assignment operators are used to assigning value to the variables and compound assignment operator where we combine an operator with an assignment operators to shorten the steps and make code more effective

Operators - =, +=, -=, *=, /=, %=

//Write example

5. Increment and Decrement Operators -

Dart Arithmetic Increment Operator takes a numeric variable as operand and increments its value by 1.

Dart Arithmetic Decrement Operator takes a numeric variable as operand and decrements its value by 1

Operator - ++, --

//Write example

6. Bitwise and Shift Operators -

The Bitwise operators perform operation bit by bit on the value of the two operands.

Operators - &, |, ^, <<, >>, tilt sign

//Write example

7. Type Test Operators -

This class of operators contain those operators which are used to perform comparison on the operands.

Operators -

1. is - Gives boolean value true as output if the object has specific type.

2. is! - Give boolean value false as output if the object has specific type.

3. as - as operator is used for typecasting. It performs a cast at runtime if the cast is valid else, it throws an error.

8. Dart Conditional Operators -

The Conditional Operator is same as if-else statement and provides similar functionality as conditional statement.

It is the second form of if-else statement. It is also identified as **"Ternary Operator".**

Syntax 1 -

condition ? exp1 : exp2;

If condition is true, then the expression evaluates expr1 (and returns its value);

otherwise, it evaluates and returns the value of expr2.

Syntax 2 -

exp1 ?? Expr2

If the exp1 is not-null, returns its value, otherwise returns the exp2's value.

6. Explain flow control constructs ?

Flow control constructs are fundamental elements in programming languages that manage the order in which statements, instructions, or function calls are executed or evaluated.

Common Flow Control Constructs -

1. Conditional Statements:

These include **if, if-else, if-else if ladder, and switch statements**. They allow the execution of specific code blocks based on whether a condition is true or false.

2. Loops:

Constructs like **for, while, and do-while** loops that repeat a block of code multiple times.

3. Branching Statements:

Such as **break, continue, and return**, which control the flow within loops and functions.

7. Explain object oriented programming in Dart?

Object-Oriented Programming (OOP) in Dart is a programming paradigm that structures code around objects, combining data and methods to model real-world entities.

Class :-

In Dart, class is a fundamental concept used for object-oriented programming. They serve as a blueprint for creating objects, which are instances of the class.

Syntax -

```
class Class_name {
  // Class members go here
}
```

Objects :-

In Dart, objects are instances of classes. They represent real-world entities or concepts and encapsulate data (fields) and behavior (methods) related to those entities.

Example -

```
class MyClass {
  // Class members go here
}
void main() {
  MyClass myObject = MyClass(); // Object creation
}
```

Encapsulation :-

Encapsulation refers to the bundling of data and the methods into single unit.

Inheritance :-

Inheritance is a key concept in object-oriented programming, allowing for code reusability, and Dart is no exception.

The new class inherits properties and behavior (data members and methods) from the existing class.

Example -

```
class Animal {
  void eat() {
    print('The animal eats');
  }
}
class Dog extends Animal {
  @override
  void eat() {
    print('The dog eats');
  }
  void bark() {
    print('The dog barks');
  }
}
void main() {
  Dog myDog = Dog();
  myDog.eat(); // Outputs 'The dog eats'
  myDog.bark(); // Outputs 'The dog barks'
}
```

Polymorphism In Dart :-

Polymorphism in Dart refers to the ability of a class or data type to take on multiple forms or behave differently depending on the context.

Types of polymorphism -

1. Compile time polymorphism -

- Operator Overloading

2. Run Time polymorphism

- Method overriding
- Method overloading

Abstraction in Dart :-

Abstraction in Dart is a fundamental concept that allows us to hide complex details and display only necessary information.

8. Explain the concept of arrow functions in Dart ?

Dart has a special syntax for the function body, which is only one line. The arrow function is represented by => symbol. It is a shorthand syntax for any function that has only one expression.

Syntax -

```
returnType functionName(parameters...) => expression;
```

Example -

```
int add(int n1, int n2) => n1 + n2;

int sub(int n1, int n2) => n1 - n2;

int mul(int n1, int n2) => n1 * n2;

double div(int n1, int n2) => n1 / n2;


void main()

{

        int num1 = 100;

        int num2 = 30;

        print("The sum is ${add(num1, num2)}");

        print("The diff is ${sub(num1, num2)}");

        print("The mul is ${mul(num1, num2)}");

        print("The div is ${div(num1, num2)}");

}
```

9. Define Instance variables ?

In Dart, instance variables are variables that are defined within a class and are associated with a specific object of that class.

10. What are named parameters ?

In Dart, named parameters are optional parameters that can be referenced by name when calling a function. They are defined within curly braces in a function's parameter list.

11. What are methods ?

Methods are functions defined within the class, allowing objects to perform actions or behaviors.

Unit - 3

Widgets

1. Define Widgets?

Widgets are building blocks which the developer, will compose together to make a UI. In Flutter, everything is a widget.

Types of Widgets -

1. Visible (Output and Input)
2. Invisible (Layout and Control)

2. Define Gestures?

Gestures are primarily a way for a user to interact with a mobile (or any touch based device) application. GestureDetector handle gestures.

3. List and explain some of the widely used gestures ?

- **Tap:** Touching the surface of the device with fingertip for a short period and then releasing the fingertip.
- **Double Tap:** Tapping twice in a short time.
- **Drag:** Touching the surface of the device with fingertip and then moving the fingertip in a steady manner and then finally releasing the fingertip.
- **Flick:** Similar to dragging, but doing it in a speeder way.
- **Pinch:** Pinching the surface of the device using two fingers.
- **Spread/Zoom:** Opposite of pinching.
- **Panning:** Touching the surface of the device with fingertip and moving it in any direction without releasing the fingertip.

4. Define state?

The state of an app can very simply be defined as anything that exists in the memory of the app while the app is running.

The state of the widget is the data of the objects that its properties (parameters) are sustaining at the time of its creation (when the widget is painted on the screen).

5. Explain different layers in flutter ?

- **Widgets Layer:** Constructs the UI using widgets.
- **Rendering Layer:** Handles layout and painting of the widgets.
- **Painting Layer:** Draws the UI on the screen.
- **Gestures Layer:** Detects and responds to user gestures.
- **Services Layer:** Provides core services such as plugins and platform-specific functionality.

6. Explain the types of widgets in flutter ?

1. Platform specific widgets:

Flutter has widgets specific to a particular platform - Android or iOS.-widgets are designed in accordance with Material design guideline by Android Or ios.

Platform Specific Widgets are like tools that are specially designed for a particular type of device or operating system. Think of them as clothes that fit perfectly for a particular person.

Android: Imagine these widgets are designed to look and feel like Android apps. They follow the Material Design guidelines by Google.

Examples -

- **FloatingActionButton:** A round button that floats above the content, often used for primary actions.
- **SnackBar:** A small banner that appears at the bottom of the screen to show a quick message.
- **BottomNavigationBar:** A bar at the bottom of the screen for navigating between different sections of an app.
- **BottomNavigationBar**
- **DropdownButton**
- **IconButton**

IOS: These widgets look and feel like iOS apps. They follow Apple's Cupertino design guidelines.

Examples -

- **CupertinoButton:** A button that looks like an iOS button.
- **CupertinoSlider:** A slider that looks like an iOS slider.
- **CupertinoNavigationBar:** A navigation bar that looks like an iOS navigation bar.
- **CupertinoDatePicker**
- **CupertinoTimerPicker**
- **CupertinotTextField**

Advantages

- Feels natural and familiar to users of that platform.
- More work to create different versions for each platform.
- Less flexibility for customization.

2. Layout widgets:

Layout Widgets are like the layout plans of a room where you decide where to place furniture. In your app, layout widgets help you arrange other widgets (like text, buttons, images) on the screen.

Types of Layout Widgets

1. Single Child Widget -

Single Child Widgets are designed to contain only one widget inside them. Think of them as boxes that hold one item and maybe do something special to it, like adding padding or centering it.

Examples

- **Container:** A versatile box that can add padding, margins, borders, and background colors to one child widget.
- **Padding:** Adds space around a single child widget.

- **Align:** Aligns one child widget within its boundaries.
- **Center:** Centers one child widget within itself.
- **SizedBox:** Gives a specific size to its single child.

Usage:

These are used when you need to apply specific properties to one widget.

For example, centering a text or adding padding around an image.

2. Multiple Child Widget -

Multiple Child Widgets can contain multiple widgets inside them. Think of them as shelves where you can place multiple items in a particular order.

Examples:

- **Row:** Arranges its children in a horizontal line, like placing items side by side on a shelf.
- **Column:** Arranges its children in a vertical line, like stacking items on top of each other.
- **Stack:** Allows its children to overlap each other, like placing pictures one on top of another.
- **ListView:** A scrollable list of widgets arranged in a single column. Think of a list you can scroll through.
- **GridView:** A scrollable grid of widgets arranged in rows and columns, like a photo gallery.

Usage:

These are used for more complex layouts where you need to arrange several widgets.

For example, creating a row of buttons or a column of text fields.

3. State maintenance widgets:

In Flutter, all widgets are either derived from StatelessWidget or StatefulWidget.

1. Stateless Widget -

Stateless Widget is something that does not have a state. The only area of focus of a stateless widget is the information displayed and the user interface, They deal with situations that are independent of the user's input.

Key Characteristics:

- **Immutability:** Stateless widgets cannot change their state after they are built.
- **Performance:** Because they are immutable, stateless widgets are lightweight and simple, making them more efficient.

2. Stateful Widget -

A stateful widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely.

4. Platform independent / basic widgets:

Platform Independent Widgets are like universal tools that work the same way no matter what type of device you use. Think of them as clothes that fit anyone, regardless of size or shape.

Examples -

- Text: Displays a piece of text.
- Image: Shows an image.
- Container: A versatile box that can contain other widgets and add padding, margins, borders, and background colors.
- Row: Arranges child widgets in a horizontal line.
- Column: Arranges child widgets in a vertical line.

Advantages -

- One set of code works on multiple platforms (Android, iOS, etc.).

- Easier to manage and update since you don’t need to create multiple versions.

Disadvantages -

- Might not look exactly like native apps on each platform.

7. Explain MaterialApp widget ?

MaterialApp is a container for your entire Flutter app. It sets up the app’s main structure and provides essential features and services.

The MaterialApp widget is the starting point for any Flutter app that uses Material Design.

It sets up various settings and configurations for the app, such as the title, theme, locale, and home screen.

Reasons to use MaterialApp -

- A consistent look and feel across al platforms
- Easy theming
- Support for device orietation changes
- A number of pre-built widgets

Key Properties of MaterialApp -

- **home:** The default route of the app, which is usually the first screen the user sees.
- **title:** A one-line description used by the device to identify the app for the user.
- **theme:** The theme data for the app, which allows you to customize the colors and styles.
- **debugShowCheckedModeBanner:** If true, shows a debug banner on the top right corner of the screen (useful for development).
- **routes:** A map of routes to widgets.
- **navigatorObservers:** A list of navigator observers.

8. Explain Scaffold Widget ?

The Scaffold widget provides a structure for the visual interface of a Material Design app.

It implements the basic visual layout structure, including an app bar, body, bottom navigation bar, floating action button, and drawer.

Key Properties -

- **appBar:** A horizontal bar typically shown at the top of the app, containing the app's title and other actions.
- **body:** The primary content of the Scaffold.
- **floatingActionButton:** A button that floats above the content.
- **drawer:** A panel that slides in from the side of the app.

Example -

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

void main() {

runApp(MaterialApp(

home: Scaffold(

        appBar: AppBar(

        title: Text('Scaffold Example'),

        ),

        body: Center(

        child: Text('This is the body of the Scaffold.'),

        ),

),

));

}
```

9. List differences between MatrialApp and Scaffold Widget ?

MaterialApp -

MaterialApp Widget is the starting point of your app, it tells Flutter that you are going to use Material components and follow the material design in your app.

MaterialApp is a widget that introduces a number of widgets Navigator, Theme that are required to build a material design app.

Scaffold -

Scaffold Widget is used under MaterialApp, it gives you many basic functionalities, like AppBar, BottomNavigationBar, Drawer, FloatingActionButton, etc.

The Scaffold is designed to be the single top-level container for a MaterialApp although it is not necessary to nest a Scaffold.

10. Define Center Widget?

The Center widget is a simple layout widget that centers its child within itself.

It is useful when you want to center a single child widget in the middle of its parent.

Key Properties -

- child: The widget to be centered.

Example -

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

void main() {

runApp(MaterialApp(

home: Scaffold(

        appBar: AppBar(

                title: Text('Center Widget Example'),

        ),

        body: Center(

                child: Text('Centered Text'),
          ),
      ),
      ));
      }
```

11. Explain Text Widget ?

The `Text` widget in Flutter is used to display a string of text in the application.

It supports various customization options to control its appearance and behavior, making it versatile for different text presentation needs.

The Text widget in Flutter is used to display a string of text with a single style.

Properties:

- **data:** The string to display.
- **style:** Defines the style of the text (e.g., color, font size, font weight).
- **textAlign:** Aligns the text within the widget (e.g., left, right, center).
- **maxLines:** The maximum number of lines for the text.
- **overflow:** Defines how to handle text overflow (e.g., ellipsis, fade).

12. Explain Image Widget ?

The Image widget in Flutter is used to display images in the application. It supports loading images from various sources such as assets, network, and files, allowing for flexible and dynamic image rendering.

The Image widget in Flutter is used to display images in the application.

Properties:

- **image:** Specifies the image to display (from assets, network, or file).
- **width and height:** Define the dimensions of the image.

- **fit:** Defines how the image should fit within the bounds (e.g., fill, contain, cover).
- **alignment, AlignmentGeometry** – How to align the image within its bounds

13. Explain Icon Widget ?

The Icon widget in Flutter is used to display graphical icons. It provides a wide range of predefined icons from Material Design and custom icons, enabling consistent and visually appealing iconography in the app.

Description:

The Icon widget in Flutter is used to display a graphical icon.

Properties:

- **icon:** Specifies the icon to display (e.g., from Material Icons or custom icons).
- **color:** Sets the color of the icon.
- **size:** Sets the size of the icon.

14. Explain Row And Column Widget ?

Row Widget -

This widget arranges its child widgets in a horizontal direction on the screen.

A row widget does not appear scrollable because it displays the widgets within the visible view.

We can control how a row widget aligns its children based on our choice using the property crossAxisAlignment and mainAxisAlignment.

Column Widget -

This widget arranges its child widgets in a vertical direction on the screen.

We can also control how a column widget aligns its children using the property mainAxisAlignment and crossAxisAlignment.

The column's crossaxis will run horizontally, and the main axis will run vertically.

15. Explain Expanded, Align, Container, and Padding widget ?

Expanded Widget -

The Expanded widget expands a child widget to fill the available space along the main axis (for Row the main axis is horizontal & vertical for Column).

- **flex:** An integer that determines the ratio of space that the child occupies relative to other expanded children.
- **child:** The widget that is expanded to fill the available space.
- **fit:** This property controls how the child widget fills the available space.

Align Widget -

Align Widget is the widget that is used to align its child within itself and optionally sizes itself based on the child's size.

Properties of Align Widget:

- **alignment:** It sets the alignment.
- **child:** The child widget in the tree.
- **heightFactor:** It sets its height to the child's height multiplied by this heightFactor.
- **key:** It is used to controlling how one widget replaces the other one.
- **runtimeType:** It is a representation of runtime type.
- **widthFactor:** It sets its width to the child's width multiplied by this widthFactor.

Container Widget -

The container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc.

Properties:

- **alignment:** Aligns the child within the container. Uses Alignment class constants.

- **padding:** Creates space around the child widget inside the container.
- **margin:** Creates space around the outside of the container.
- **color:** Sets the background color of the container.
- **decoration:** Applies decoration like border, background image, etc., to the container

Padding Widget -

Padding widget adds padding or empty space around a widget or a bunch of widgets, We can apply padding around any widget by placing it as the child of the Padding widget.

Properties -

- **padding -** Defines the padding space
- **child -** This property simply takes a widget as the object to display is inside the padding widget on the screen.

16. Explain button widget ?

Buttons are the graphical control element that provides a user to trigger an event such as taking actions, making choices, searching things, and many more.

Properties:

1. child: The widget or text displayed as the button's content.

2. onPressed: A callback function that gets triggered when the button is pressed.

17. Explain the types of button widget ?

1. Elevated Button -

A material design button with elevation when pressed. i.e When an elevated button is pressed its elevation is increased to a certain value.

},

```
ElevatedButton(
onPressed: () {
        // Action to perform when button is pressed
},
        child: Text('Elevated Button'),
)
```

2. TextButton -

A text-based button with no elevation. They are regular text without any outline or boundary.

```
TextButton(
onPressed: () {
        // Action to perform when button is pressed
},
        child: Text('Text Button'),
)
```

3. OutlineButton -

It is similar to the flat button, but it contains a thin grey rounded rectangle border. Its outline border is defined by the shape attribute.

```
OutlinedButton(
onPressed: () {
// Action to perform when button is pressed
},
child: Text('Outlined Button'),
)
```

4. IconButton -

A button with an icon. An IconButton is a picture printed on the Material widget.

It is a useful widget that gives the Flutter UI a material design feel.

```
IconButton(
icon: Icon(Icons.star),
onPressed: () {
// Action to perform when button is pressed
},
)
```

5. FloatingActionButton -

The FloatingActionButton widget in Flutter is a button typically placed in a scaffold's Scaffold.floatingActionButton property, used for primary actions in the application.

17. List types of flutter buttons?

- Flat Button
- Raised Button
- Floating Button
- Drop Down Button
- Icon Button
- Inkwell Button
- PopupMenu Button

- Outline Button

Unit - 4

Gestures & State Management

1. Define Gestures?

In Flutter, a gesture is a way to capture and respond to user interactions like tapping, dragging, or swiping on the touch screen.

Gestures are generally defined as any physical action / movement of a user in the intention of activating a specific control of the mobile device.

2. Explain some of the widely used gestures ?

1. Tap -

Touching the surface of the device with fingertip for a short period and then releasing the fingertip.

Events -

- **onTapDown:** A pointer that might cause a tap has contacted the screen at a particular location.

- **onTapUp:** A pointer that triggers a tap has stopped contacting the screen at a particular location.
- **onTap:** The pointer that previously triggered the onTapDown has also triggered onTapUp which ends up causing a tap.
- **onTapCancel:** The pointer that previously triggered the onTapDown won't end up causing a tap.

2. Double Tap -

Two quick taps in succession.

Used for actions like zooming into an image

Event:

- **onDoubleTap:** The user has tapped the screen at the same location twice in quick succession.

3. Long Press -

A tap that is held down for a long duration.

Used for actions like showing a context menu.

Event -

- **onLongPress:** A pointer has remained in contact with the screen at the same location for a long period of time.

4. Vertical Drag -

A drag gesture that primarily moves up or down

Used for actions like scrolling or swiping vertically.

Events -

- onVerticalDragStart
- OnVerticalDragUpdate
- onVerticalDragEnd

5. Horizontal Drag -

A drag gesture that primarily moves left or right.

Used for actions like scrolling or swiping horizontally

Events -

- onHorizontalDragStart
- onHorizontalDragUpdate
- onHorizontalDragEnd

6. Pan -

A drag gesture that can move freely in any direction.

Used for actions like dragging a widget around the screen.

Events -

- onPanStart
- onPanUpdate
- onPanEnd

7. Dialogs -

Dialogs in Flutter are pop-up interfaces that provide important information, prompt the user to make a decision, or require user input without navigating away from the current screen.

Types of dialogs in a flutter

1. AlertDialog -

AlertDialog is one of the most commonly used dialog types in Flutter.

It typically consists of:

- A title
- A content section for messages or information
- Actions, usually in the form of buttons

2. SimpleDialog -

A simple dialog allows the user to choose from different choices. It contains the title which is optional and presented above the choices.

Properties:

- **Title:** It is always recommended to make our dialog title as short as possible.
- **Shape:** It is used to define the shape of our dialog box whether it is circular, curve, and many more.
- **backgroundcolor:** It is used to set the background color of our dialog box.
- **TextStyle:** It is used to change the style of our text

3. showDialog -

It basically used to change the current screen of our app to show the dialog popup. You must call before the dialog popup.

Properties:

- **Builder:** It returns the child instead of creating a child argument.
- **Barriercolor:** It defines the modal barrier color which darkens everything in the dialog.
- **useSafeArea:** It makes sure that the dialog uses the safe area of the screen only not overlapping the screen area.

3. Define State?

In Flutter, state is the information that can change over the lifetime of a widget and affects how the UI looks or behaves.

4. Define State Management ? Explain its types

A state management can be divided into two categories based on the duration the particular state lasts in an application.

Ephemeral - Last for a few seconds like the current state of an animation or a single page like current rating of a product. Flutter supports its through StatefulWidget.

app state - Last for entire application like logged in user details, cart information, etc., Flutter supports its through scoped_model.

5. Define Navigation And Routing?

Navigation refers to the process of switching between different screens or pages within an application.

While **routing** involves managing the flow and organization of these screens.

6. Define Routes, Navigator and Named Routes ?

- **Routes:** A route is a screen or a page in your app. Each route is typically represented by a Widget.
- **Navigator:** Navigator class manages routes using a stack-based approach, It provides methods to push (navigate to a new screen) and pop (go back to the previous screen) routes.
- **Named Routes:** Named routes are string-based identifiers that you can use to navigate to different screens.

7. Define matrialpageRoute?

MaterialPageRoute is a widget used to render its UI by replacing the entire screen with a platform specific animation.

8. Explain navigation in flutter ?

To navigate to a new screen in Flutter, you can use the **Navigator.push() method.**

This method allows you to push a new route onto the stack and transition to the new screen.

For example:

```
onPressed: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) => const SecondRoute()),
);
}),
```

To return to the previous screen, you can use the **Navigator.pop() method.**

This method pops the current route off the stack, revealing the previous route.

For example:

```
onPressed: () {
Navigator.pop(context);
}
```

9. Explain Named Routing ?

Named routing is another technique for implementing navigation in Flutter.

It allows you to refer to your routes using predefined string identifiers, known as “names,” instead of directly dealing with the routes themselves.

In named routing, navigation is achieved by calling Navigator.pushNamed(), passing the name of the route you want to navigate to.

For example:

```
Navigator.pushNamed(context, '/details');
```

To go back to the previous screen, you can simply pop the current route using Navigator.pop() method.

For example:

```
Navigator.pop(context);
```

10. Explain Stateful widget ?

Statefulwidget provides an option for a widget to create a state, State when the widget is created for the first time through createState method and then a method, setState to change the state whenever needed.

A Stateful Widget has its own mutable state that it needs to track. It is modified according to the user's input.

A Stateful Widget looks after two things primarily, the changed state based on its previous state and an updated view of the user interface.

11. Explain TextField Widget ?

The TextField widget in Flutter is used to create a basic text input field. It allows users to enter text, which can be used for various purposes like forms, search bars, and more.

Example -

```
        TextField (

                decoration: InputDecoration(

                border: InputBorder.none,

                labelText: 'Enter Name',
            hintText: 'Enter Your Name'
      ),
      );
```

Attributes of TextField Widget -

- **decoration:** It is used to show the decoration around TextField.
- **border:** It is used to create a default rounded rectangle border around TextField.
- **labelText:** It is used to show the label text on the selection of TextField.
- **hintText:** It is used to show the hint text inside TextField.
- **icon:** It is used to add icons directly to the TextField.
- **obscureText:** Hides the text being entered (useful for passwords).

12. Explain CheckBox Widget ?

The Checkbox widget allows the user to select multiple options from a set. It is either checked or unchecked.

A marked/checked checkbox means yes, and an unmarked/unchecked checkbox means no value.

Attributes -

- **value -** It is used whether the checkbox is checked or not.
- **onChanged -** It will be called when the value is changed.
- **titile -** It specified the main title of the list.
- **subtitle -** It specified the subtitle of the list. Usually, it is used to add the description.
- **activeColor -** It specified the color of the selected checkbox.
- **activeColor** - It specified the color of the selected checkbox.
- **selected -** By default, it is false. It highlights the text after selection.
- **secondary -** It is the widget, which is displayed in front of the checkbox.

Example -

```
Checkbox(
        value: isSelected,
        onChanged: (value) {
        setState(() {
        isSelected = value;
        });
},
)
```

13. Explain Radio Widget ?

The Radio widget allows the user to select a single option from a set of options. It is useful for mutually exclusive choices.

Attributes -

- **groupValue -** It specifies the currently selected item for the radio button group.
- **title -** We use it to specify the radio button label.
- **value -** Specifies the backhand value, represented by a radio button.
- **onChanged -** Called whenever the user selects a radio button.

14. Explain Date And Time Picker Widgets ?

1. DatePicker Widget -

The DatePicker widget in Flutter allows the user to select a date from a calendar-like interface.

Key Properties:

- initialDate: The initially selected date when the picker is opened.

- firstDate: The earliest date that can be selected.
- lastDate: The latest date that can be selected.

Usage:

To show a date picker, you use the showDatePicker function which returns a Future<DateTime?> that resolves to the selected date.

2. TimePicker Widget

The TimePicker widget allows the user to select a time.

Key Properties:

- initialTime: The initially selected time when the picker is opened.

Usage:

To show a time picker, you use the showTimePicker function which returns a Future<TimeOfDay?> that resolves to the selected time.

15. Explain LastView Widget ?

ListView is a scrollable list of widgets arranged in a linear fashion. It's a commonly used widget in Flutter for displaying a list of data.

ListView is highly flexible and can be customized to suit your needs. It can display a list of items, a grid, or even a combination of both.

ListView with ListTile

The ListTile widget is a convenient way to create a list item with a leading icon, title, subtitle, and trailing widget (like an icon or button).

Key Properties of ListTile:

- **leading:** A widget to display before the title, usually an icon.
- **title:** The main content of the list item.
- **subtitle:** Additional content below the title.

- **trailing:** A widget to display after the title, usually an icon or button.
- **onTap:** A callback when the tile is tapped.

ListView.builder()

The builder() constructor constructs a repeating list of widgets. The constructor takes two main parameters:

1. An itemCount for the number of repetitions for the widget to be constructed
2. An itemBuilder for constructing the widget which will be generated 'itemCount' times (compulsory).

If the itemCount is not specified, infinite widgets will be constructed by default.

Example -

```
ListView.builder(
    itemCount: 20,
    itemBuilder: (context, position) {
        return Card(
        child: Padding(
            padding: const EdgeInsets.all(20.0),
            child: Text(
            position.toString(),
            style: TextStyle(fontSize: 22.0),
            ),
        ),
    );
},
),
```

Unit - 5

Introduction to Dart Packages, Accessing REST API, Database Concept

1. Define Packages in Flutter?

Packages in Flutter are pre-built collections of code that provide more features and capabilities to your app. Flutter packages are designed to be reusable and modular, allowing developers to use existing code instead of writing it from scratch.

Types of packages -

- **Pub Packages:** Available on the pub.dev repository. They can be easily added to any Dart project.
- **User-defined Packages:** Custom packages created by developers to share specific functionalities across their own projects.

2. List and explain types of flutter packages ?

1. Dart Package:

- This type of package contains general-purpose Dart code that can be used in both mobile and web environments.
- It is written in Dart language and may include some Flutter-specific functionalities.
- Examples of Dart packages include libraries for handling HTTP requests, managing state, or parsing JSON data.

2. Flutter Package:

- Flutter packages are generic Dart code specifically designed for mobile app development using the Flutter framework.
- These packages rely solely on the Flutter framework and cannot be used in web environments.
- Examples of Flutter packages include UI component libraries, state management solutions, or utilities optimized for Flutter apps.

3. Flutter Plugin:

- Flutter plugins are generic code that interfaces with platform-specific code and the Flutter framework.
- These packages typically provide access to native platform features and functionalities not directly available in Flutter.
- Examples of Flutter plugins include packages for accessing device cameras, GPS location, or platform-specific APIs like Google Maps.

3. List benefits of using packages ?

- Reusability
- Modularity
- Community Support

4. How to add packages in Flutter ?

Finding and Adding Packages from pub.dev

Finding Packages

- Visit pub.dev to search for packages.
- keywords or browse categories to find the desired package.

Adding Dependencies to pubspec.yaml

Dependencies:

- yaml:
- http: ^0.14.0

Installing Packages

Run flutter pub get in the terminal to install the packages specified in pubspec.yaml.

Adding Dependencies to pubspec.yaml

Understanding pubspec.yaml

The pubspec.yaml file is the configuration file for Dart and Flutter projects, specifying project dependencies and other settings.

Example of Adding Dependencies

```
name: my_app

description: A new Flutter project.

dependencies:

flutter:

sdk: flutter
```

provider: ^6.0.0

http: ^0.14.0

5. What is pubspec.yaml in Flutter ?

pubspec.yaml is a configuration file in Flutter projects where you define metadata about your project, including its name, description, dependencies (including Flutter packages), and other settings.

6. Explain creating and publishing package in flutter ?

Creating a package -

1. Create Package Directory Structure

- Use the following command to create a new package:
- Command : dart create -t package-simple my_package

2. Implementing Package Functionality

Example:

Create a simple utility function in lib/my_package.dart: Dart library my_package;

int add(int a, int b) {

return a + b;

}

Publishing a Package -

- Update pubspec.yaml with necessary information.
- Run dart pub publish to publish the package to pub.dev.

7. List and explain some core packages in Flutter ?

Flutter/material:

This package provides the Material Design widgets and components, allowing developers to create visually appealing and responsive user interfaces.

Flutter/cupertino:

This package provides the Cupertino widgets and components, which follow the iOS design guidelines.

Flutter/widgets:

This package contains the foundation of Flutter's widget framework. It include essential widgets for building user interfaces, handling gestures, layout, animation, and more.

Flutter/rendering:

This package handles the low-level rendering aspects of Flutter. It provides classes and APIs for painting, layout, and compositing the visual elements of the user interface

8. Define Web server ?

A web server is a software application that handles HTTP requests from clients (typically web browsers) and serves HTTP responses.

Popular web servers include Apache, Nginx, and Microsoft IIS.

9. Define Host and HTTP Protocol?

Host -

In the context of web services, a host refers to the domain name or IP address of the server where the web application is deployed.

For example, in the URL http://www.example.com, www.example.com is the host.

HTTP Protocol -

HTTP (Hypertext Transfer Protocol) is the foundation of any data exchange on the web.

It is an application layer protocol for transmitting hypermedia documents, such as HTML.

10. Define JSON(Javascript Object Notation)?

JSON is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.

JSON Syntax -

- Objects are enclosed in curly braces {} and contain key/value pairs.
- Arrays are enclosed in square brackets [] and contain values.
- Key/Value Pairs are written as "key": value.

11. Define API ?

API acts as a mediator between the end user and backend resources and services

12. What is REST API ?

REST (Representational State Transfer) API or the web API conforms to the constraints of REST architectural style and allows app interactions with the RESTful web services.

13. How does REST API Work ?

- REST API uses simple HTTP calls to communicate with JSON data.
- http class provides functionality to perform all types of HTTP requests.
- http methods accept a url, and additional information through Dart Map (post data, additional headers, etc.,). It requests the server and collects the response back in async/await pattern.
- For example, the below code reads the data from the specified url and print it in the console.

```
print(await http.read('https://flutter.dev/'));
```

14. List and explain HTTP methods ?

- **read –** Request the specified url through GET method and return back the response as Future<String>.
- **get –** Request the specified url through GET method and return back the response as Future<Response>. Response is a class holding the response information.
- **post –** Request the specified url through POST method by posting the supplied data and return back the response as Future<Response>.
- **put –** Request the specified url through PUT method and return back the response as Future <Response>
- **head –** Request the specified url through HEAD method and return back the response as Future<Response>
- **delete –** Request the specified url through DELETE method and return back the response as Future<Response>.

15. Write steps to integrate REST API in the Flutter App ?

- **Step 1:** Get the API URL and endpoints.
- **Step 2:** Add relevant packages into the app (http, dio, chopper, etc.).
- **Step 3:** Create a constant file that stores URLs and endpoints.
- **Step 4:** Create a model class to parse the JSON.
- **Step 5:** Create a file that handles the API call, and write specific methods to fetch and parse data.
- **Step 6:** Use the data in your app.

16. Define SQLite?

SQLite is a popular database software library that provides a relational database management system for local/client storage. It is a light-weight and time-tested database engine and contains features like self-contained, server-less, zero-configuration, transactional SQL database engine.

17. List methods of SQLite Db Provider ?

- **openDatabase:** Opens a connection to the database
- **execute:** Creates a table in the database
- **where:** Deletes specific rows from a table