Mobile app design

Practical experiments

Third Class, SW branch, Second Semester.
(2024-2025)

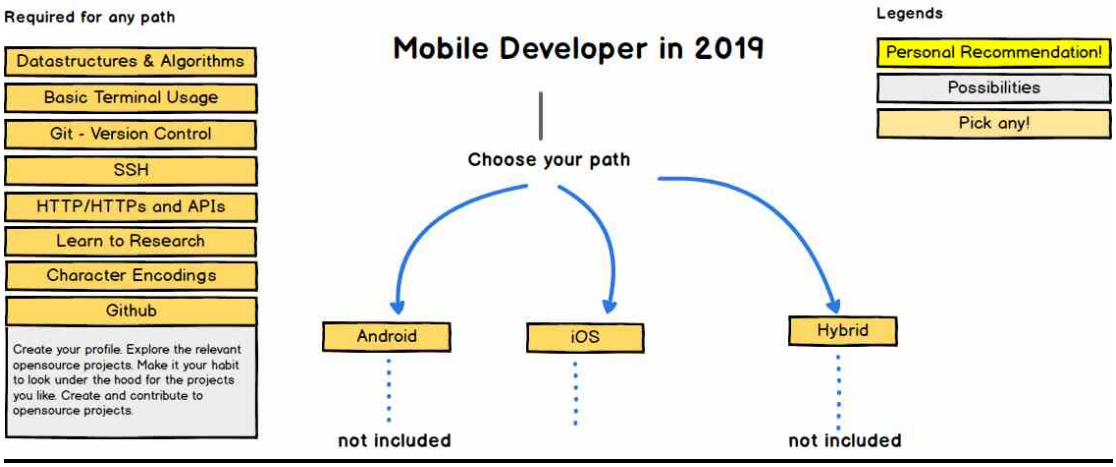
By: Teaba Wala aldeen Khairi.

University of technology, Computer science department.
E-Mail: 110053@uotechnology.edu.iq

Second course 3rd / Mobile app design

Lecture 1 Android studio & Vs code

Mobile platform



Android studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development. Android Studio is the official IDE (Integrated Development Environment) for Android app development and it is based on JetBrains' IntelliJ IDEA software. Android Studio provides many excellent features that enhance productivity when building Android apps, such as:

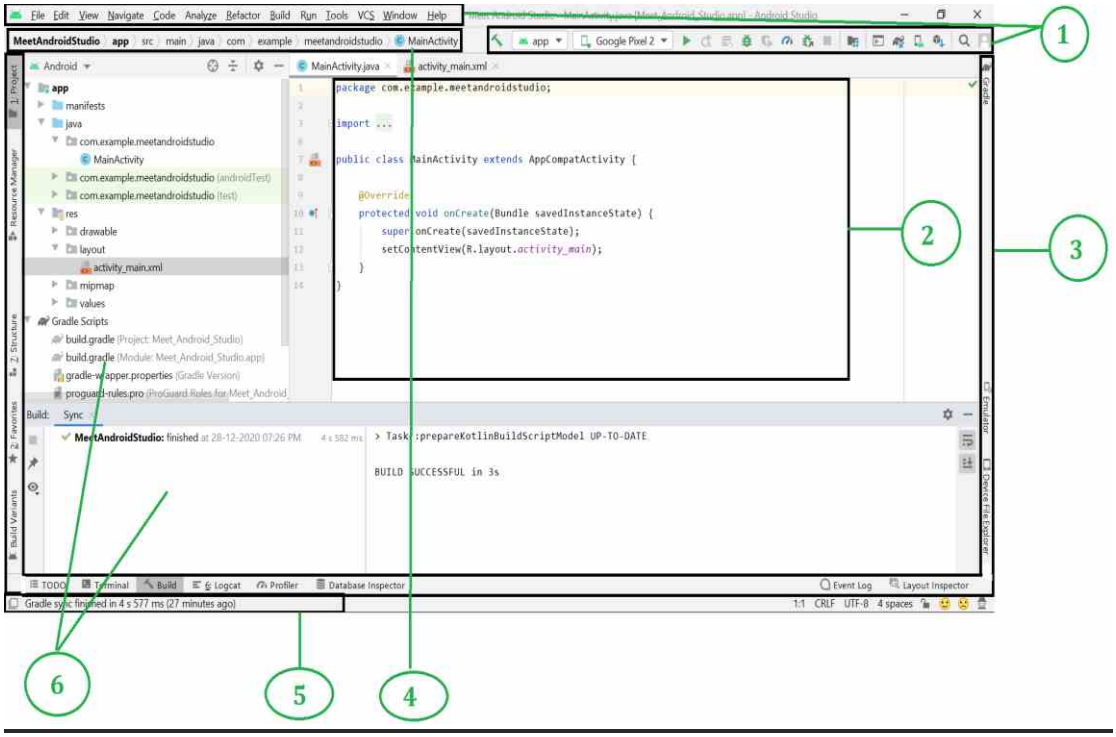
- A blended environment where one can develop for all Android devices
- Apply Changes to push code and resource changes to the running app without restarting the app
- A flexible Gradle-based build system

- A fast and feature-rich emulator
- GitHub and Code template integration to assist you to develop common app features and import sample code
- Extensive testing tools and frameworks
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine, and many more.
- Provides GUI tools that simplify the less interesting parts of app development .
- Easy integration with real time database ‘firebase.’

System Requirements :

- Microsoft Windows 7/8/10 (32-bit or 64-bit
- 4 GB RAM minimum, 8 GB RAM recommended (plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum, 4 GB recommended (500 MB for IDE plus 1.5 GB for Android SDK and emulator system image
- 1280 x 800 minimum screen resolution

Download Android Studio & App Tools - Android Developers



1. Toolbar

One can find the Toolbar section at the top of the android studio. It contains a wide range of functions including creating a new project which is inside the File Section, Reformat your code which is inside the Code Section, Rebuild your project which is inside the Build section, running your android app which is inside the Run section, and many more. In fact, the Toolbar is the most important part of the android studio.

2. Editor Window

In this window, the users can create, write, and modify their code. This editor window changes depending on the current file type. For example, if you are viewing a layout file(Here activity_main.xml file) then the editor displays the layout editor. Similarly, if you are writing the backend code then the editor displays the Java/Kotlin file (Here MainActivity.java file) depending upon the language you have chosen during the creation of the project.

3. Tool Window Bar

The tool window bar operates around the outside of the IDE window and includes the buttons that allow users to expand or collapse individual tool windows. For example, in the above diagram, we have expanded the Project and the Build button.

4. Navigation Bar

The navigation bar helps the users to navigate throughout the project and open files for editing. It gives a more compressed view of the structure visible in the Project window. For example in the above diagram when we click on the MainActivity.java file in the editor window section then the navigation bar shows us where this file present inside the android studio. (Here app > src > main > java > com > example > meetandroidstudio > MainActivity). Thus it helps us to navigate and modify the required file easily.

5. Status Bar

The status bar displays the current status of the project and the IDE itself, as well as any warnings or messages during the execution of the project .

5. Tool Windows

The tool windows give access to specific tasks like project management, search, version control, and more. It is strongly related to the Tool Window Bar section. When you expand the Tool Window Bar then they are visible inside the Tool Windows section. So the user can also expand and collapse them .

Note: The user can organize the main window to get more screen space by hiding or moving toolbars and tool windows. The user can also use the shortcut keyboard to access most IDE features.

Android studio is a very standard IDE and there are many things to explore inside this tool. We have just discussed its Main Window here. It provides a lot of powerful things. For example, at the second last (From right) of the Toolbar section, the user can find a search icon where the users can search anything inside android studio such as searches for:

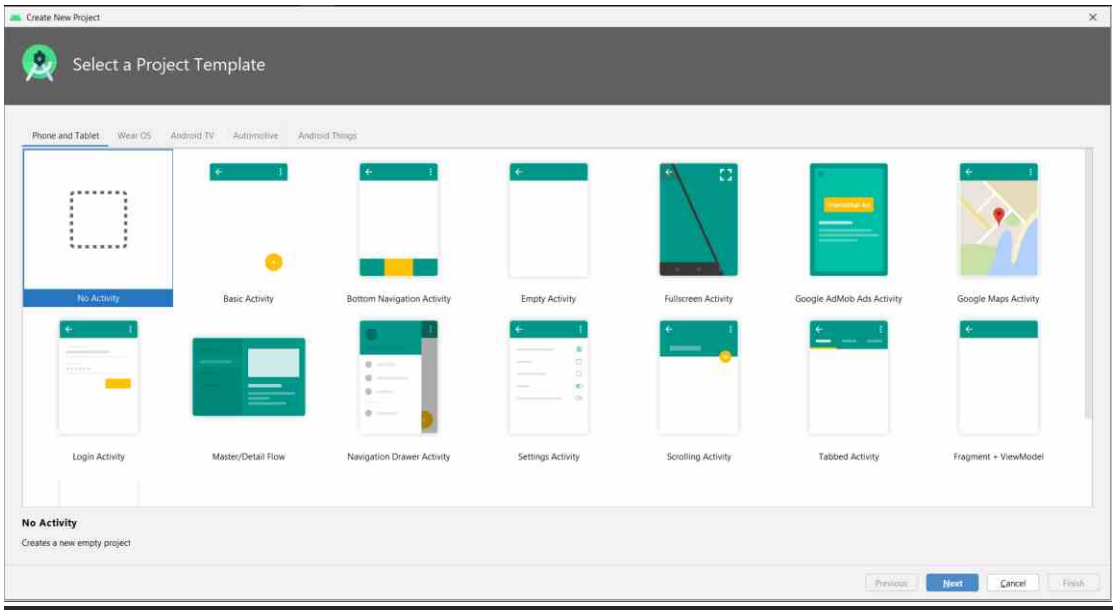
- Classes
- Files
- Tool Windows
- Actions
- Settings, etc

So before moving to the project development one should explore all the features of android studio vividly and this will be very beneficial for them during the project development .

Generally, when a developer wants to create a new project in the android studio he/she needs to select a project template which is consisting of many activities as shown in the below image. (Considering that the developer developing the android app for phone and tablet). So in this article, we are going to discuss what do these activities mean in brief. Here is the **list of activities:**

1. No Activity
2. Basic Activity
3. Bottom Navigation Activity
4. Empty Activity
5. Fullscreen Activity
6. Google Admob Ads Activity
7. Google Maps Activity
8. Login Activity
9. Master/Detail Flow
10. Navigation Drawer Activity

11.Settings Activity
12.Scrolling Activity
13.Tabbed Activity
14.Fragment + ViewModel
15.Native C++

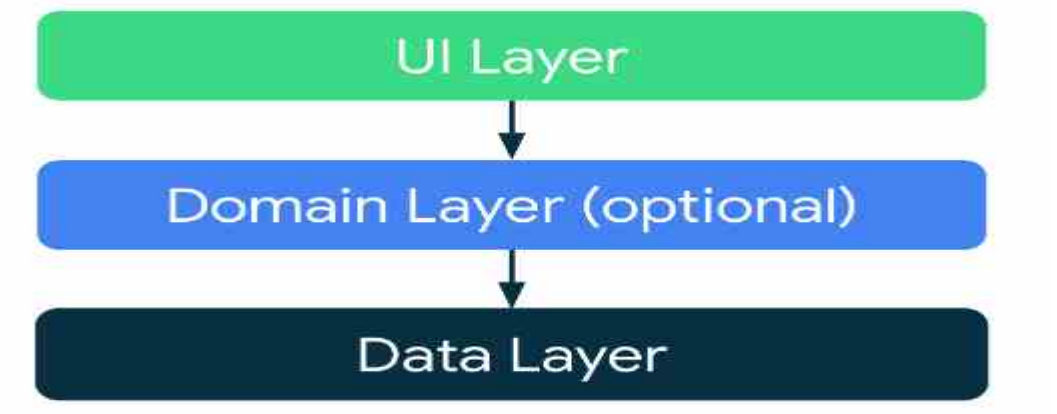


Mobible app arcitrcture

Considering the common architectural principles mentioned in the previous section, each application should have at least two layers:

- The *UI layer* that displays application data on the screen.
- The *data layer* that contains the business logic of your app and exposes application data.

You can add an additional layer called the *domain layer* to simplify and reuse the interactions between the UI and data layers.



Modern App Architecture

This *Modern App Architecture* encourages using the following techniques, among others:

- A reactive and layered architecture.
- Unidirectional Data Flow (UDF) in all layers of the app.
- A UI layer with state holders to manage the complexity of the UI.
- Coroutines and flows.
- Dependency injection best practices.

UI layer

The role of the UI layer (or *presentation layer*) is to display the application data on the screen. Whenever the data changes, either due to user interaction (such as pressing a button) or external input (such as a network response), the UI should update to reflect the changes.

The UI layer is made up of two things:

- UI elements that render the data on the screen. You build these elements using Views or Jetpack Compose functions.
- State holders (such as ViewModel classes) that hold data, expose it to the UI, and handle logic

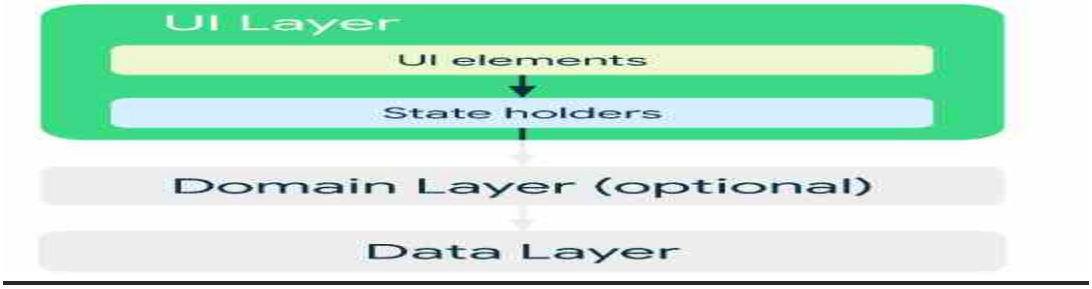


Figure 2. The UI layer's role in app architecture.

Data layer

The data layer of an app contains the *business logic*. The business logic is what gives value to your app—it's made of rules that determine how your app creates, stores, and changes data.

The data layer is made of *repositories* that each can contain zero to many *data sources*. You should create a repository class for each different type of data you handle in your app. For example, you might create a MoviesRepository class for data related to movies, or a PaymentsRepository class for data related to payments.

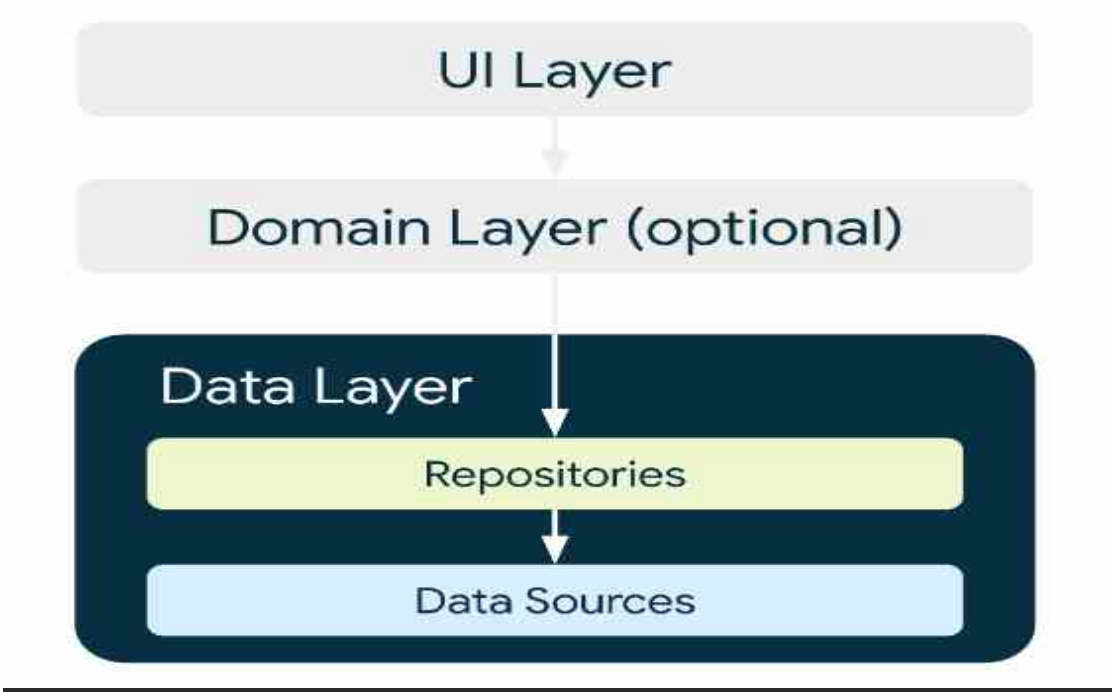


Figure 3. The data layer's role in app architecture.

Repository classes are responsible for the following tasks:

- Exposing data to the rest of the app.
- Centralizing changes to the data.
- Resolving conflicts between multiple data sources.
- Abstracting sources of data from the rest of the app.
- Containing business logic.

Each data source class should have the responsibility of working with only one source of data, which can be a file, a network source, or a local database. Data source classes are the bridge between the application and the system for data operations.

Domain layer

The domain layer is an optional layer that sits between the UI and data layers.

The domain layer is responsible for encapsulating complex business logic, or simple business logic that is reused by multiple ViewModels. This layer is optional because not all apps will have these requirements. You should use it only when needed—for example, to handle complexity or favor reusability.

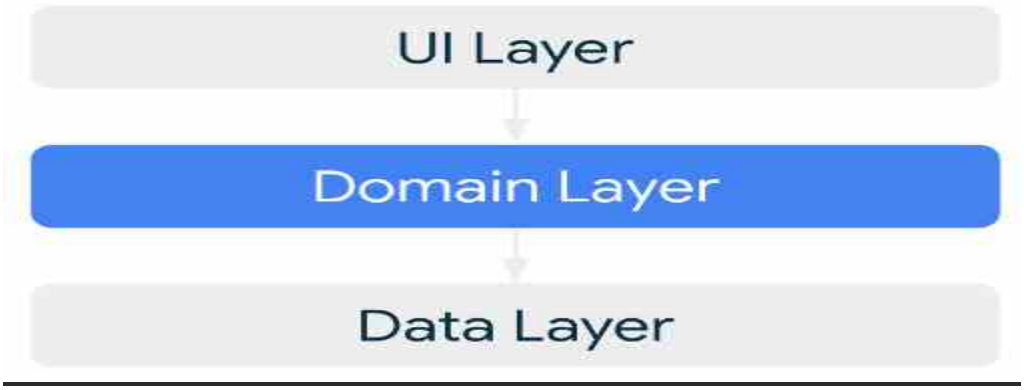


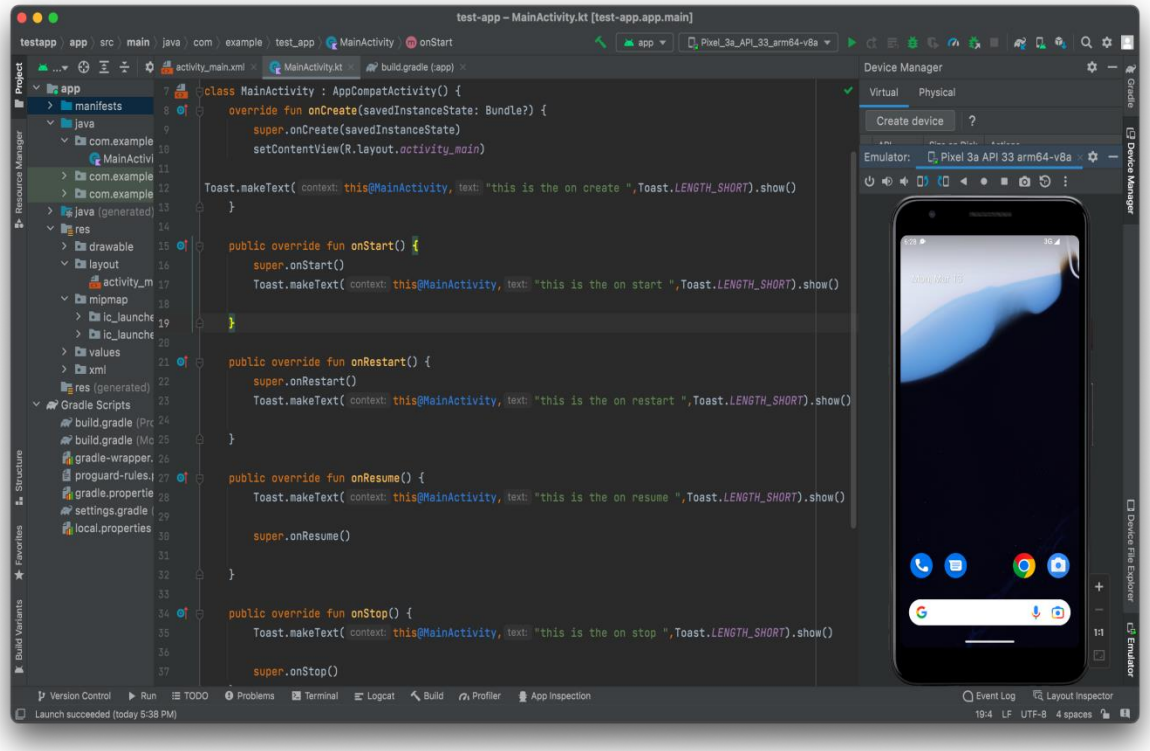
Figure 4. The domain layer's role in app architecture.

Classes in this layer are commonly called *use cases* or *interactors*. Each use case should have responsibility over a *single* functionality. For example, your app could have a GetTimeZoneUseCase class if multiple ViewModels rely on time zones to display the proper message on the screen.

Benefits of Architecture

Having a good Architecture implemented in your app brings a lot of benefits to the project and engineering teams:

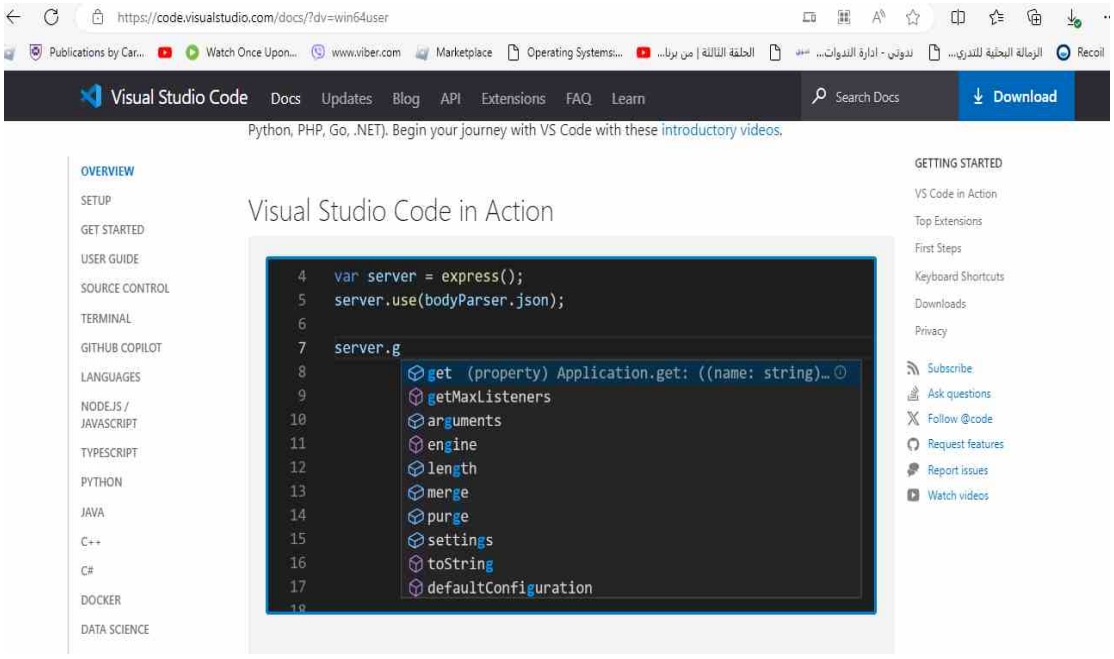
- It improves the maintainability, quality and robustness of the overall app.
- It allows the app to scale. More people and more teams can contribute to the same codebase with minimal code conflicts.
- It helps with onboarding. As Architecture brings consistency to your project, new members of the team can quickly get up to speed and be more efficient in less amount of time.
- It is easier to test. A good Architecture encourages simpler types which are generally easier to test.
- Bugs can be investigated methodically with well defined processes.



Visual studio code

To downloaded it use

[Documentation for Visual Studio Code]



Lecture 2 Flutter & Dart

WHAT IS FLUTTER?

Flutter allows you to build beautiful native apps on iOS and Android from a single codebase

- Open-source mobile app SDK
- Developed by Google
- Building high-performance apps for iOS and Android, from a single codebase

WHY USE FLUTTER?

Flutter makes it easy and fast to build beautiful mobile apps.

• Reactive framework

• Material and Cupertino widgets

• Hot reload

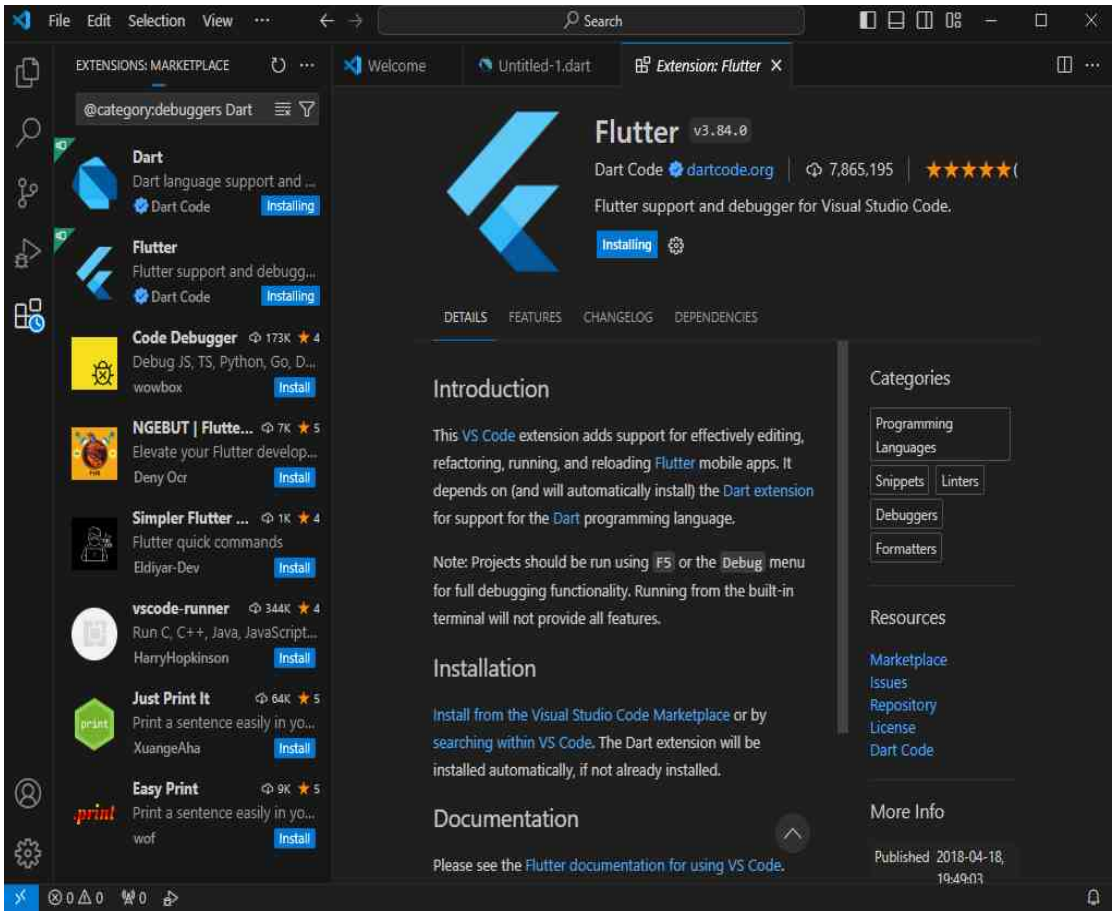
• Dart language and core libs

• Interop with mobile SDKs

• Android Studio/IntelliJ official IDE

• Debugger, Format

Install dart and flutter



What is widgets in Flutter

Widgets: Each element on a screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of an apps is a tree of widgets.

Category of Widgets:

There are mainly 14 categories in which the flutter widgets are divided. They are mainly segregated on the basis of the functionality they provide in a flutter application.

1. **Accessibility:** These are the set of widgets that make a flutter app more easily accessible.
2. **Animation and Motion:** These widgets add animation to other widgets.
3. **Assets, Images, and Icons:** These widgets take charge of assets such as display images and show icons.
4. **Async:** These provide async functionality in the flutter application.
5. **Basics:** These are the bundle of widgets that are absolutely necessary for the development of any flutter application.
6. **Cupertino:** These are the iOS designed widgets.
7. **Input:** This set of widgets provides input functionality in a flutter application.
8. **Interaction Models:** These widgets are here to manage touch events and route users to different views in the application.
9. **Layout:** This bundle of widgets helps in placing the other widgets on the screen as needed.
10. **Material Components:** This is a set of widgets that mainly follow material design by Google.
11. **Painting and effects:** This is the set of widgets that apply visual changes to their child widgets without changing their layout or shape.
12. **Scrolling:** This provides scrollability of to a set of other widgets that are not scrollable by default.
13. **Styling:** This deals with the theme, responsiveness, and sizing of the app.
14. **Text:** This displays text.

Flutter – State Management

Based on states, widgets

are divided into 2 categories:

1- Stateless Widget

2- Stateful Widget

3- Inherited Widget

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. This includes all the widgets that maintain the UI of the app including the buttons, text fonts, icons, animations, etc. So now as we know what are these states let's dive directly into our main topic i.e what are these stateful and stateless widgets and how do they differ from one another.

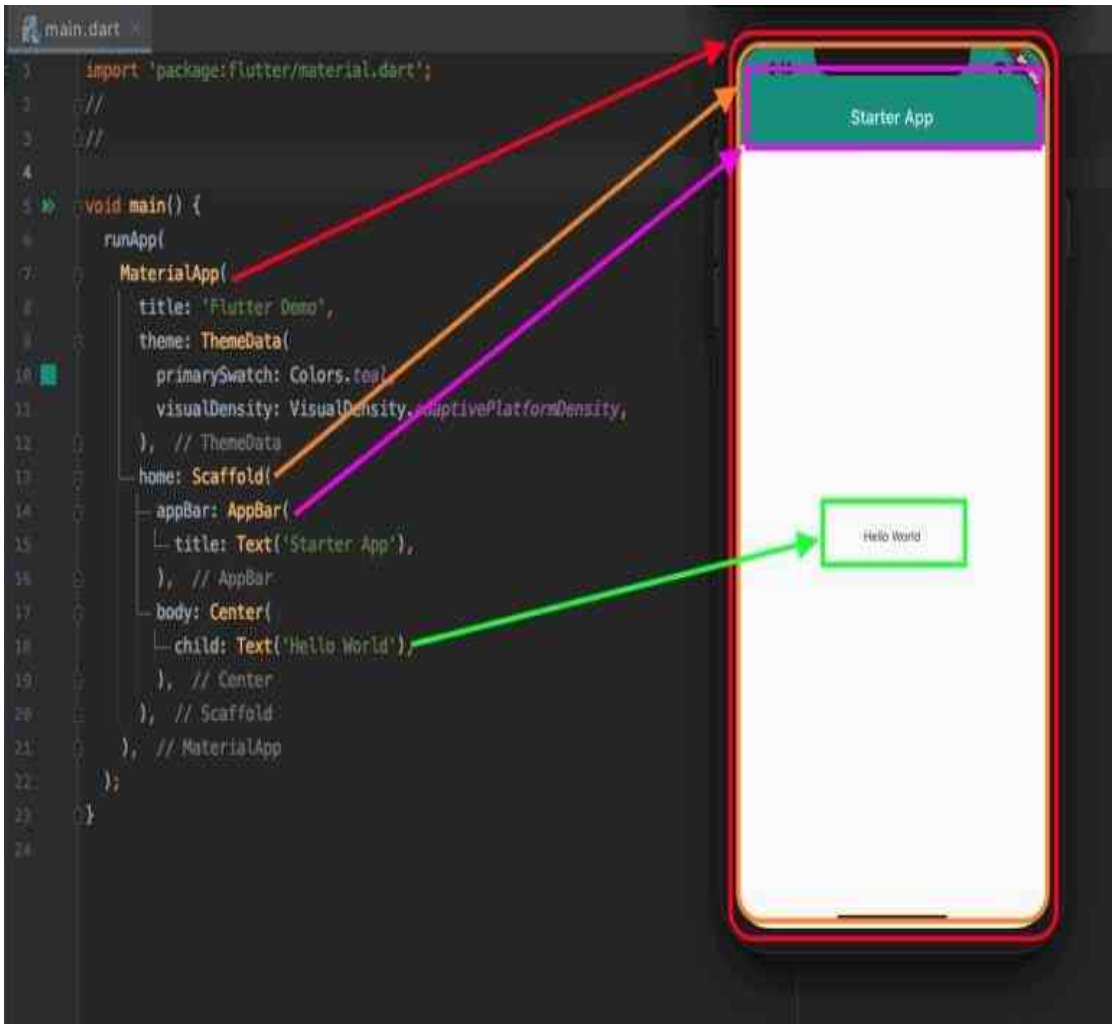
State: The State is the information that can be read synchronously when the widget is built and might change during the lifetime of the widget.
In other words, the state of the widget is the data of the objects that its properties (parameters) are sustaining at the time of its creation (when the widget is painted on the screen). The state can also change when it is used for example when a *CheckBox* widget is clicked a check appears on the box.

Stateless Widgets: The widgets whose state can not be altered once they are built are called stateless widgets. These widgets are immutable once they are built i.e any amount of change in the variables, icons, buttons, or retrieving data can not change the state of the app. Below is the basic structure of a *stateless widget. Stateless* widget overrides the *build*() method and returns a widget. For example, we use *Text* or the *Icon* in our flutter application where the state of the widget does not change in

the *runtime*. It is used when the UI depends on the information within the object itself. Other examples can be *Text, RaisedButton, IconButtons*.

You can run the codes using online

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



The difference between child: and children: property in Flutter

child takes a single widget

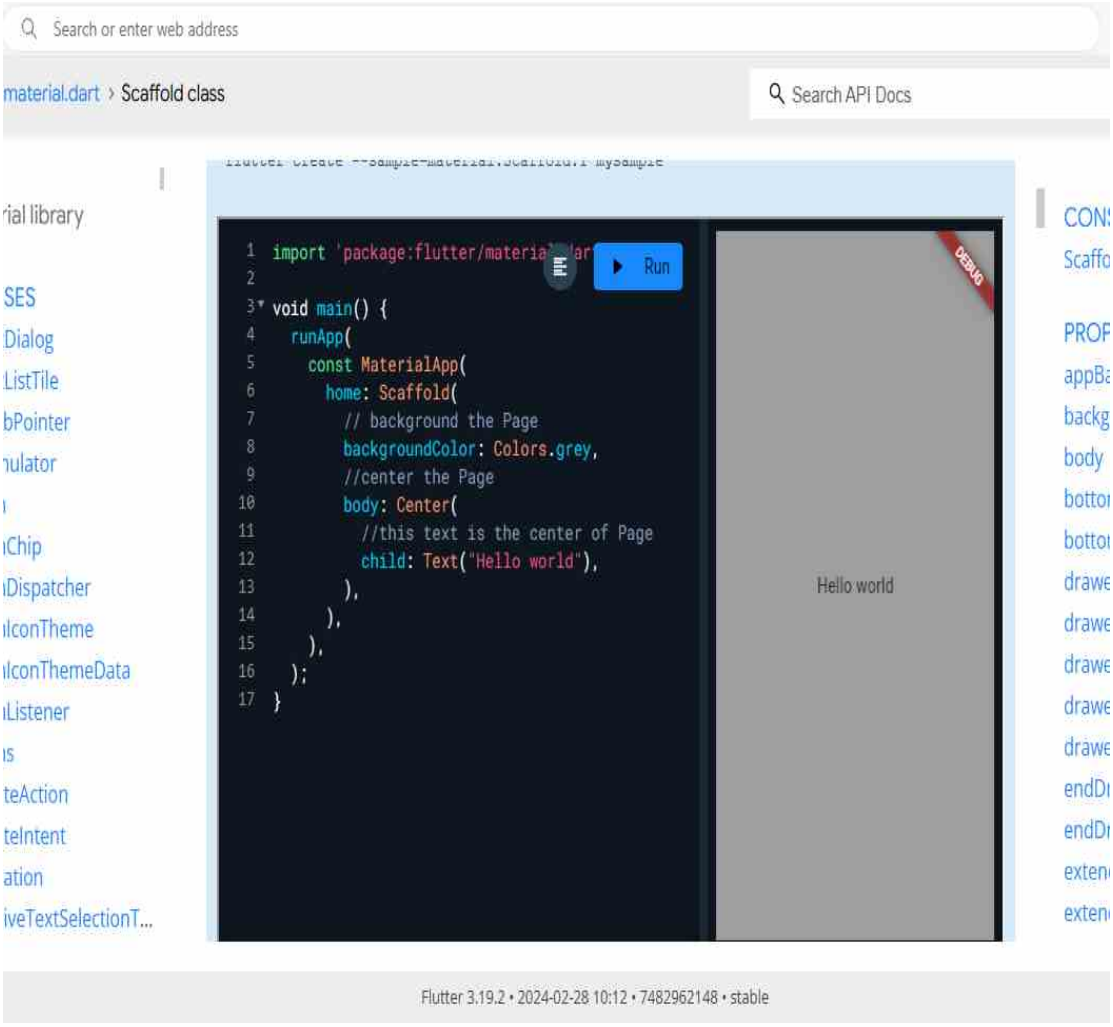
child: Text('foo')

children takes a list of widgets

children: <Widget>[Text('foo'), Text('bar')]

Project 1 Hello word on screen with gray background

```
import 'package:flutter/material.dart;'
void main} ()
  runApp)
    const MaterialApp)
      home: Scaffold)
 //        background the Page
        backgroundColor: Colors.grey,
//         center the Page
        body: Center)
//           this text is the center of Page
          child: Text("Hello world”) }
```

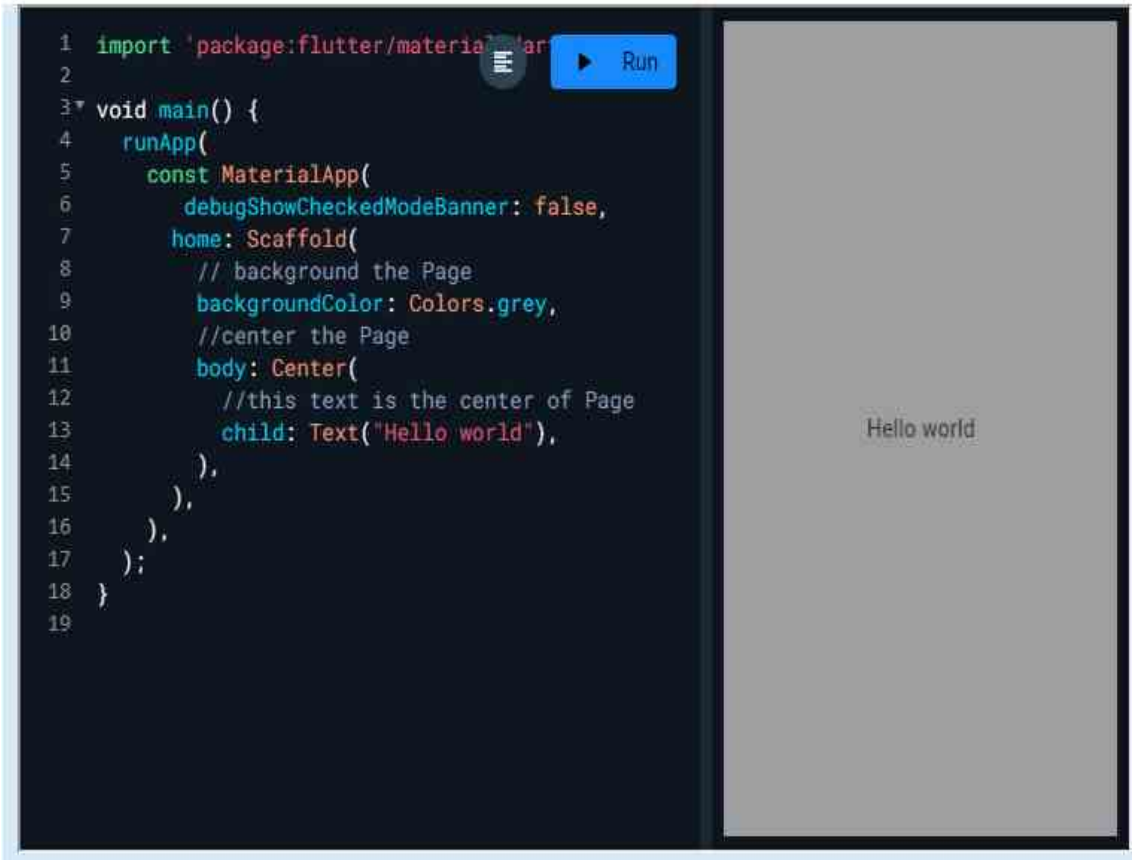


To remove banner we can use

debugShowCheckedModeBanner: false,

Project 1 Remove banner

```
import 'package:flutter/material.dart;'
void main} ()
  runApp)
    const MaterialApp)
debugShowCheckedModeBanner: false,
      home: Scaffold)
 //       background the Page
        backgroundColor: Colors.grey,
//        center the Page
        body: Center)
//          this text is the center of Page
          child: Text("Hello world")     ,
```


import 'package:flutter/materia
Run
void main() {
runApp(
const MaterialApp(
debugShowCheckedModeBanner: false,
home: Scaffold(
// background the Page
backgroundColor: Colors.grey,
//center the Page
body: Center(
//this text is the center of Page
child: Text("Hello world"),
),
),
),
);
}
Hello world

Lecture 3 Screen color & Text

MaterialApp class

MaterialApp is a predefined class or widget in a flutter. It is likely the main or core component of a flutter app. The MaterialApp widget provides a wrapper around other Material Widgets. We can access all the other components and widgets provided by Flutter SDK. Text widget, DropdownButton widget, AppBar widget, Scaffold widget, List View widget, StatelessWidget, StatefulWidget, IconButton widget, TextField widget, Padding widget, ThemeData widget, etc. are the widgets that can be accessed using MaterialApp class. There are many more widgets that are accessed using MaterialApp class. Using this widget, we can make an attractive app that follows the Material Design guidelines.

The MaterialApp configures the top-level Navigator to search for routes in the following order:

1. For the / route, the home property, if non-null, is used.
2. Otherwise, the routes table is used, if it has an entry for the route.
3. Otherwise, onGenerateRoute is called, if provided. It should return a non-null value for any *valid* route not handled by home and routes.
4. Finally if all else fails onUnknownRoute is called.

If a Navigator is created, at least one of these options must handle the / route, since it is used when an invalid initialRoute is specified on startup (e.g. by another application launching this one with an intent on Android; see dart:ui.PlatformDispatcher.defaultRouteName).

This widget also configures the observer of the top-level Navigator (if any) to perform Hero animations.

This example shows how to create a MaterialApp that disables the "debug" banner with a home route that will be displayed when the app is launched.

```
MaterialApp(
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Home'),    ),  ),
  debugShowCheckedModeBanner: false,)
```

Scaffold

It is a class in **flutter** which provides many widgets or we can say APIs like Drawer, Snack-Bar, Bottom-Navigation-Bar, Floating-Action-Button, App-Bar, etc. **Flutter** Scaffold is used to display a basic material design layout that contains application bar, body, bottom navigation bar, bottom sheet, drawer, floating action button, persistent footer buttons, etc. . The class Hierarchy is as follows:

Object

↳ Diagnosticable

↳ Diagnosticable Tree

↳ Widget

↳ StateFul Widget

↳ Scaffold

```
Widget build(BuildContext context(

}

return Scaffold)

        appBar: AppBar)

        title: const Text('Hello'),  ),
```

Colors class

Color and ColorSwatch constants which represent Material design's color palette.

Project 2 : Screen clocr and tex

وضع نص واحد في منتصف الشاشة

```
void main} ()

 runApp)

   const MaterialApp)

     debugShowCheckedModeBanner: false,

     home: Scaffold)

       backgroundColor: Colors.grey,

       body: Center)

         child: Text("hello world 1"),),),),),)}
```

وضع ثلاثة نصوص عموديا في منتصف الشاشة

```
void main} ()
 runApp)
   const MaterialApp)
    debugShowCheckedModeBanner: false,
    home: Scaffold)
      backgroundColor: Colors.grey,
      body: Center)
        child: Column)
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
```

```
        children] :
          Text("hello world 1,("
          Text("hello world 2,("
          Text("hello world 3,[   ,("
```

وضع ثلاث نصوص عموديا في منتصف الشاشة

```
void main} ()
 runApp)
   const MaterialApp)
    debugShowCheckedModeBanner: false,
    home: Scaffold)
      backgroundColor: Colors.grey,
      body: Center)
```

```
        child: Column)
          children] :
            Text("hello world 1,("
            Text("hello world 2,("
            Text("hello world 3,("
```

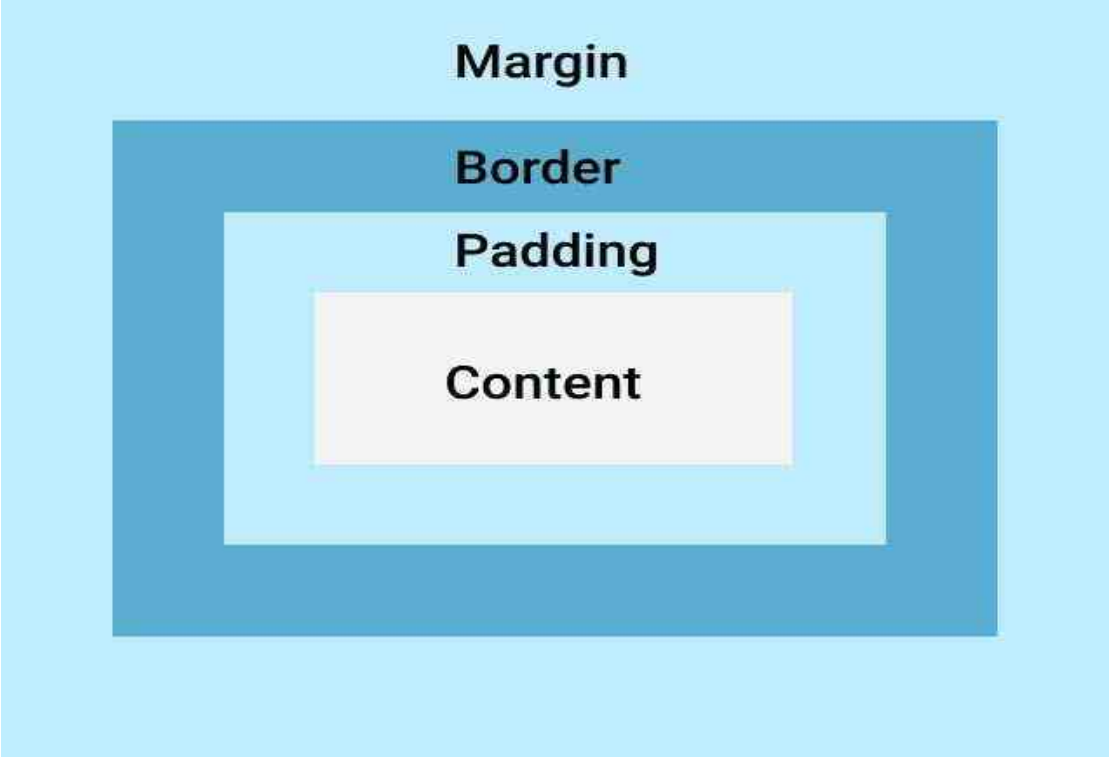
وضع ثلاثة نصوص عموديا في منتصف الشاشة ولكن بينهما فراغ مقدارة 50 بكسل

```
void main} ()
 runApp)
   const MaterialApp)
     debugShowCheckedModeBanner: false,
     home: Scaffold)
       backgroundColor: Colors.grey,
       body: Center)
```

```
        child: Column)
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children] :
            Text("hello world 1,("
            SizedBox)
              height: 50,
,(
            Text("hello world 2,("
            SizedBox)
              height: 50,(       ,
            Text("hello world 3       ,("
```

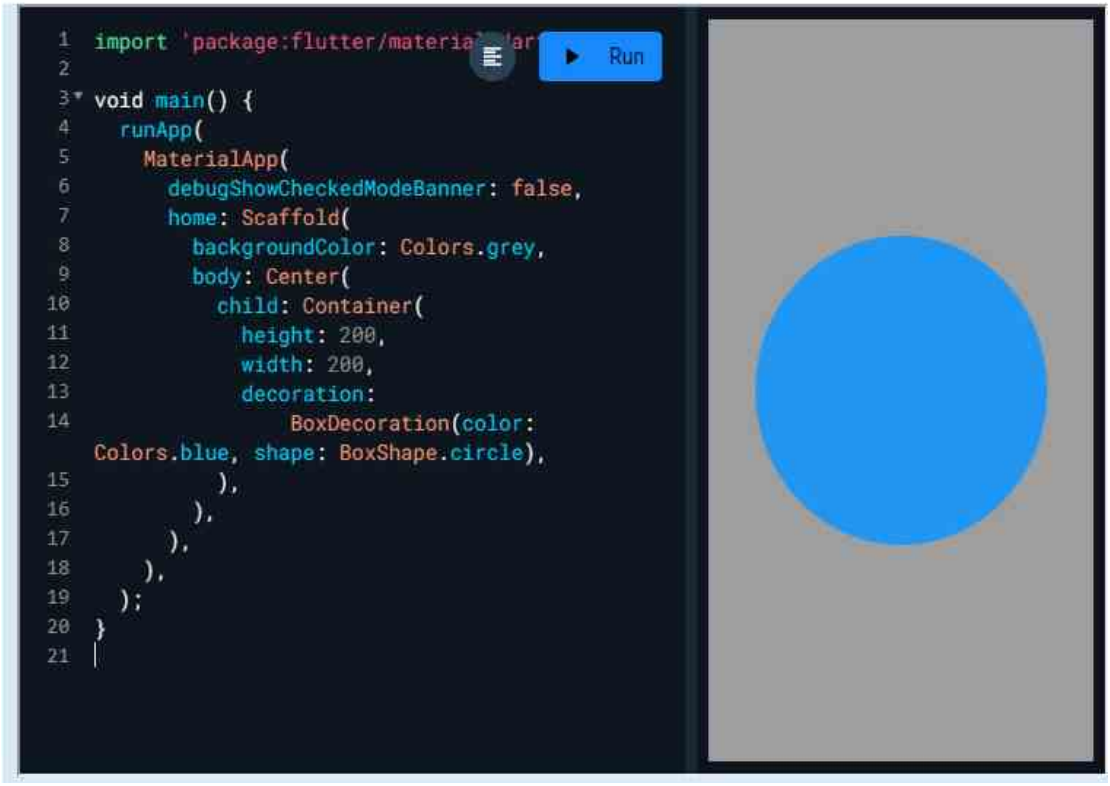
Container class in Flutter

Container class in flutter is a convenience widget that combines common painting, positioning, and sizing of widgets. A Container class can be used to store one or more widgets and position them on the screen according to our convenience. Basically, a container is like a box to store contents. A basic container element that stores a widget has a **margin**, which separates the present container from other contents. The total container can be given a **border** of different shapes, for example, rounded rectangles, etc. A container surrounds its child with **padding** and then applies additional constraints to the padded extent (incorporating the width and height as constraints, if either is non-null).



Project 3 Circle container in the middle

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

void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: Colors.grey,
        body: Center(
          child: Container(
            height: 200,
            width: 200,
            decoration:
                BoxDecoration(color: Colors.blue, shape: BoxShape.circle),
          ),
        ),
      ),
    ),
  );
}
```

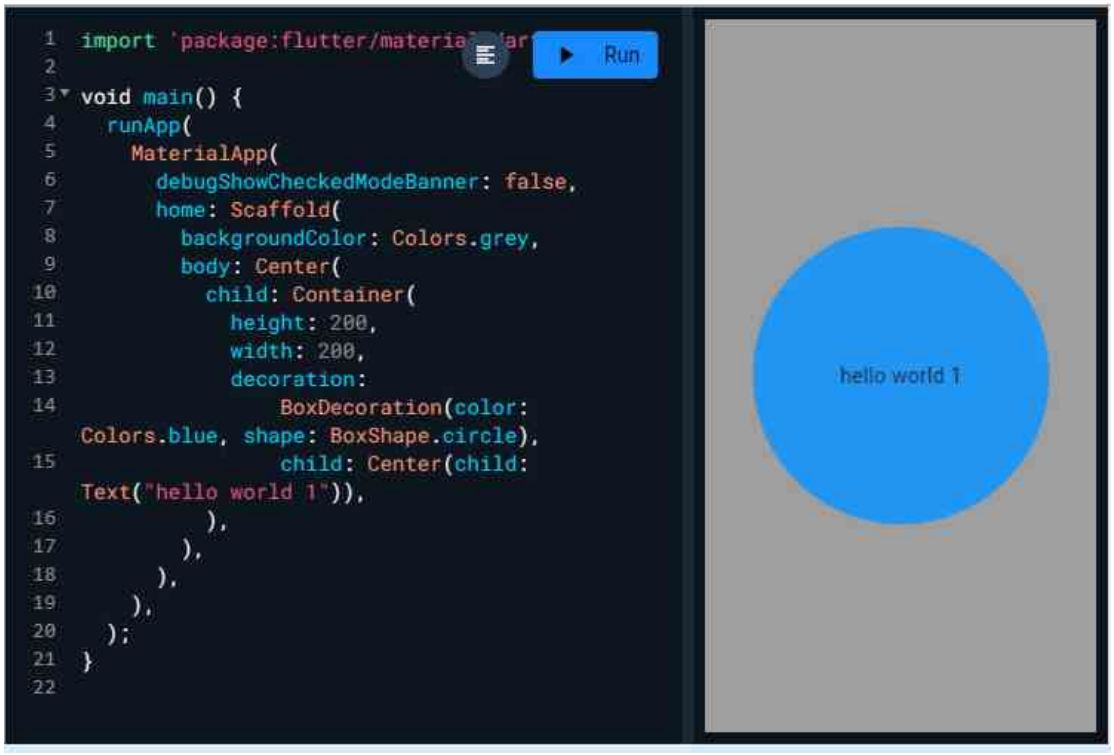


Another example

Project 3 Container with text in it

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


void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: Colors.grey,
        body: Center(
          child: Container(
            height: 200,
            width: 200,
            decoration:
                BoxDecoration(color: Colors.blue, shape:
BoxShape.circle),
```

```
                child: Center(child: Text("hello world 1")),
            ),
          ),
        ),
      ),
    );
}
```

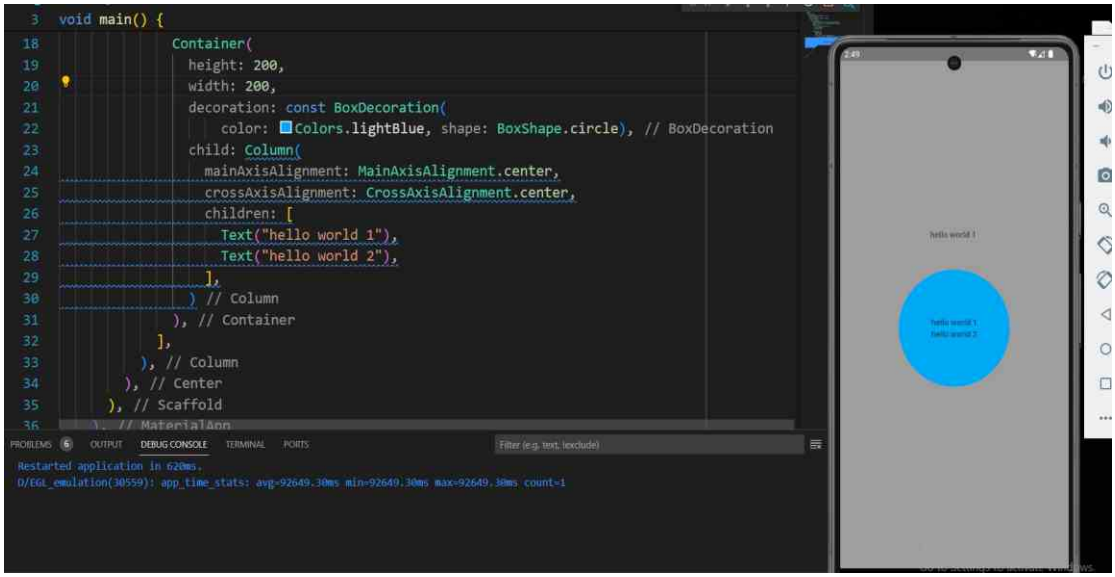


Another example

Project 3 Container with two vertical text in it

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

void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
```

```
        backgroundColor: Colors.grey,
        body: Center(
          child: Container(
              height: 200,
              width: 200,
              decoration:
                  BoxDecoration(color: Colors.blue, shape:
BoxShape.circle),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                crossAxisAlignment:
CrossAxisAlignment.center,
                children: [
                  Text("Hello world 1"),
                  Text("Hello world 2"),
                ],
              )),
        ),
      ),
    ),
  );
}
```



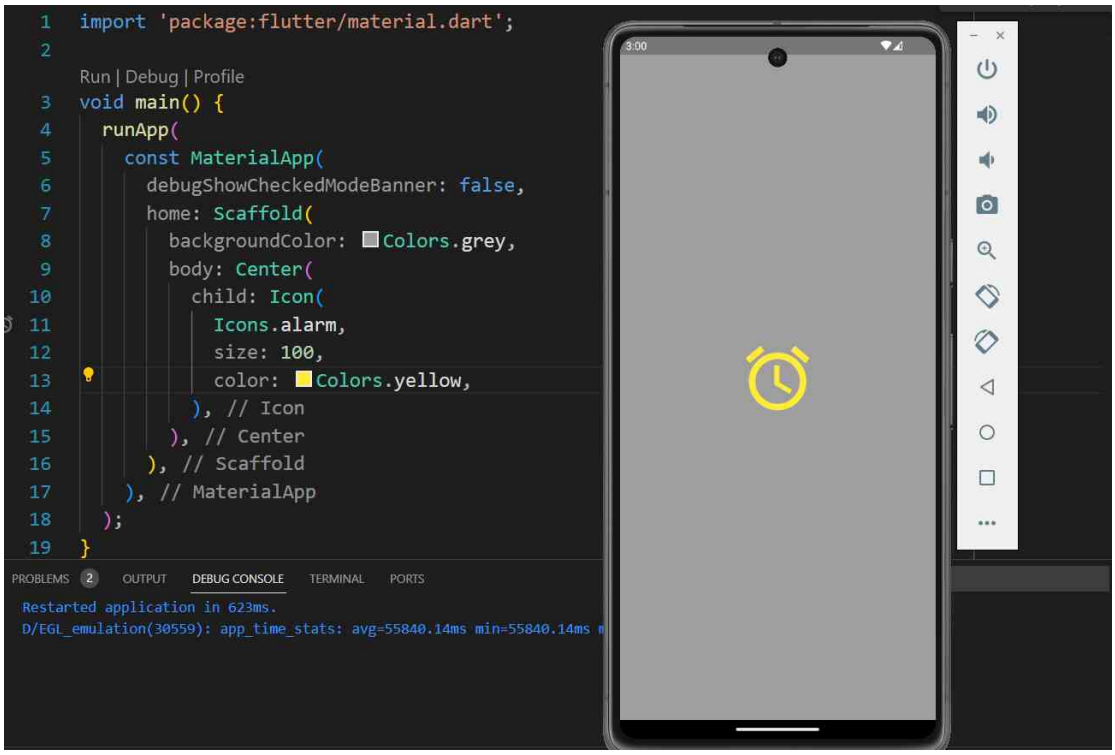
Project 3 Container with soft edges

```
void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: Colors.grey,
        body: Center(
            child: Container(
          height: 200,
          width: 200,
          decoration: BoxDecoration(
              color: Colors.blue, borderRadius:
BorderRadius.circular(20)),
        )),
      ),
    ),
  );
}
```

Project 3 Container with alarm Icon

```
void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: Colors.grey,
        body: Center(
            child: Icon(
          Icons.alarm,
          color: Colors.yellow,
```

```
            size: 100,
          )
),
        ),
      ),
    );
}
```


import 'package:flutter/material.dart';
Run | Debug | Profile
void main() {
runApp(
const MaterialApp(
debugShowCheckedModeBanner: false,
home: Scaffold(
backgroundColor: Colors.grey,
body: Center(
child: Icon(
Icons.alarm,
size: 100,
color: Colors.yellow,
), // Icon
), // Center
), // Scaffold
), // MaterialApp
);
}
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Restarted application in 623ms.
D/EGL_emulation(30559): app_time_stats: avg=55840.14ms min=55840.14ms

Lecture 5 Life cycle

Lifecycle Methods

- onCreate(Bundle savedInstanceState): create views, (re) initialize state
- onStart(): Restore *transient* state; one-time processing
- onResume(): Session-specific processing, restore *transient* state
- onPause(): Save persistent data, release resources, **quickly**! Last method guaranteed to be called.
- onStop(): Called optionally by runtime
- onDestroy(): If finish() is called, or object is being *temporarily* destroyed. Distinguish via isFinishing().

Lifecycle Methods

- onCreate(Bundle savedInstanceState): create views, (re) initialize state
- onStart(): Restore *transient* state; one-time processing
- onResume(): Session-specific processing, restore *transient* state
- onPause(): Save persistent data, release resources, **quickly**! Last method guaranteed to be called.
- onStop(): Called optionally by runtime
- onDestroy(): If finish() is called, or object is being *temporarily* destroyed. Distinguish via isFinishing().

Life cycle functions

MainActivity.kt

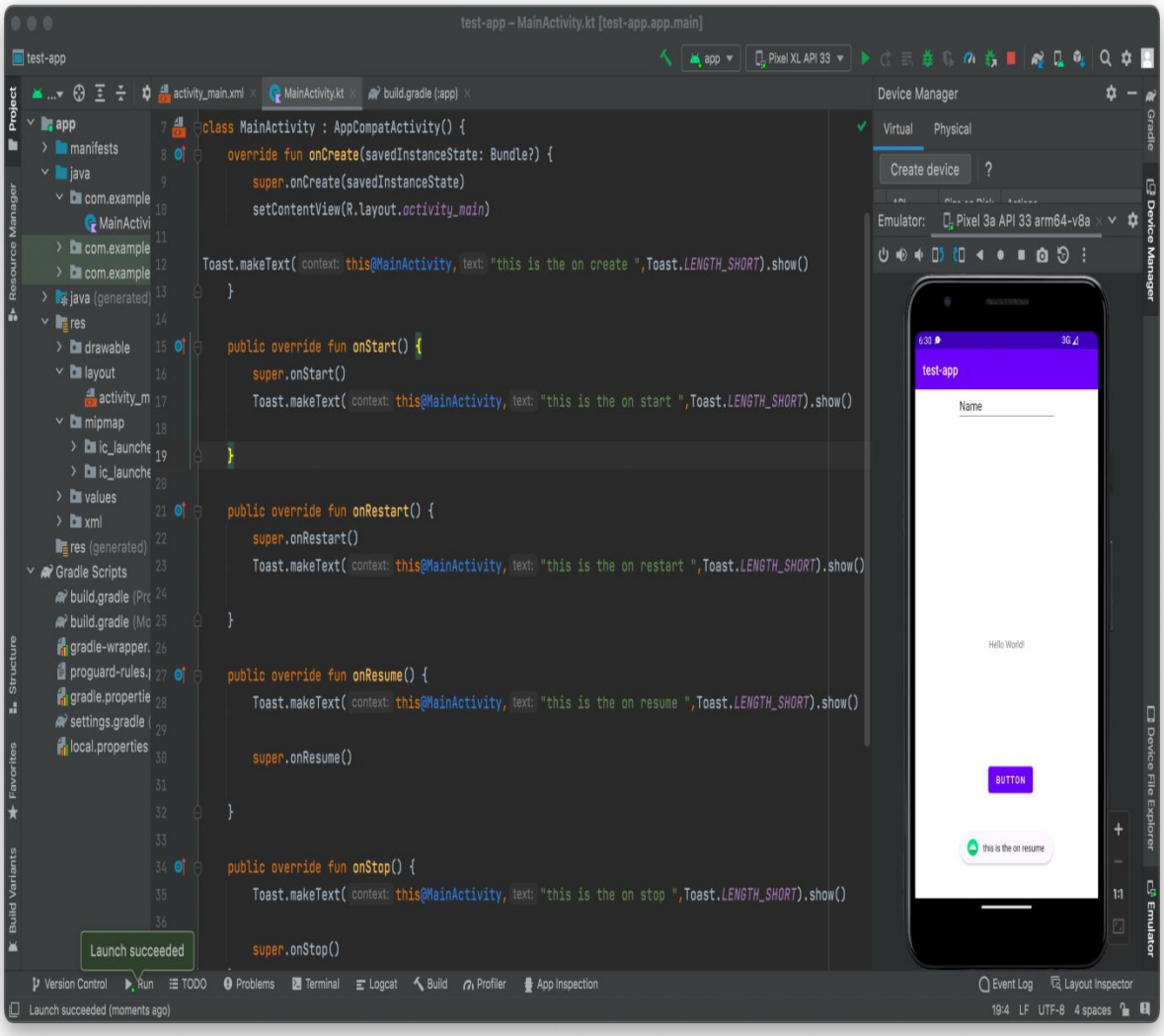
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

Toast.makeText(this@MainActivity,"this is the on create ",Toast.LENGTH_SHORT).show()

    }

    public override fun onStart() {
       Toast.makeText(this@MainActivity,"this is the on start ",Toast.LENGTH_SHORT).show()
        super.onStart()

    }

    public override fun onRestart() {
        super.onRestart()
     Toast.makeText(this@MainActivity,"this is the on restart ",Toast.LENGTH_SHORT).show()

    }

    public override fun onResume() {
      Toast.makeText(this@MainActivity,"this is the on resume ",Toast.LENGTH_SHORT).show()

        super.onResume()

    }

    public override fun onStop() {
       Toast.makeText(this@MainActivity,"this is the on stop ",Toast.LENGTH_SHORT).show()

        super.onStop()
    }

    public override fun onDestroy() {
     Toast.makeText(this@MainActivity,"this is the on destroy ",Toast.LENGTH_SHORT).show()

        super.onDestroy()
    }
}
```

Scrolling

file.dart

```
body:SingleChildScrollView(
            scrollDirection: Axis.vertical,
            physics: BouncingScrollPhysics(),
            child:
            Column()
            ),
```

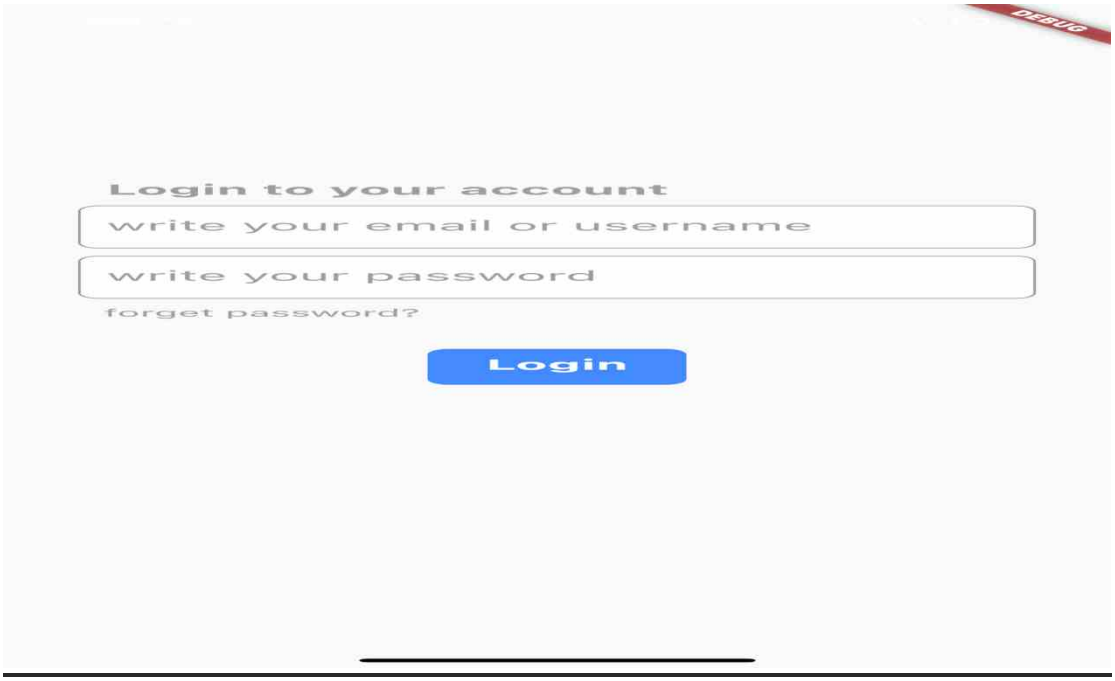
Main screen



Log in screen

main.dart

```dart
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
      body: SafeArea(
        child: Center(
          child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                SizedBox(
                  height: 200,
                ),
                Container(
                  alignment: Alignment.centerLeft,
                  padding: const EdgeInsets.only(left: 40, bottom: 10),
                  child: Text(
                    ('Login to your account'),
                    style: TextStyle(
                        fontSize: 20,
                        color: Colors.grey,
                        fontWeight: FontWeight.bold),
                  ),
                ),
                Container(
                  padding: const EdgeInsets.only(left: 10, top: 15),
                  decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius: BorderRadius.circular(8),
                      border: Border.all(color: Colors.grey)),
                  width: 370,
                  height: 60,
                  child: Text(
                    ('write your email or username'),
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.grey,
                    ),
                  ),
                ),
                Padding(padding: const EdgeInsets.only(top: 10)),
                Container(
                  padding: const EdgeInsets.only(left: 10, top: 15),
                  decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius: BorderRadius.circular(8),
                      border: Border.all(color: Colors.grey)),
                  width: 370,
                  height: 60,
                  child: Text(
                    ('write your password'),
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.grey,
                    ),
                  ),
                ),
                Container(
                    alignment: Alignment.centerLeft,
                    width: 350,
                    height: 40,
                    child: Text(
                      ('forget password?'),
                      style: TextStyle(color: Colors.grey, fontSize: 15),
                    )),
                Padding(padding: const EdgeInsets.only(top: 30)),
                Container(
                  padding: const EdgeInsets.only(top: 10),
                  decoration: BoxDecoration(
                      color: Colors.blueAccent,
                      borderRadius: BorderRadius.circular(8)),
                  width: 100,
                  height: 50,
                  child: Text(
                    ('Login'),
                    textAlign: TextAlign.center,
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
              ]),
        ),
      ),
    ));
  }
}
```


DEBUG
Login to your account
write your email or username
write your password
forget password?
Login

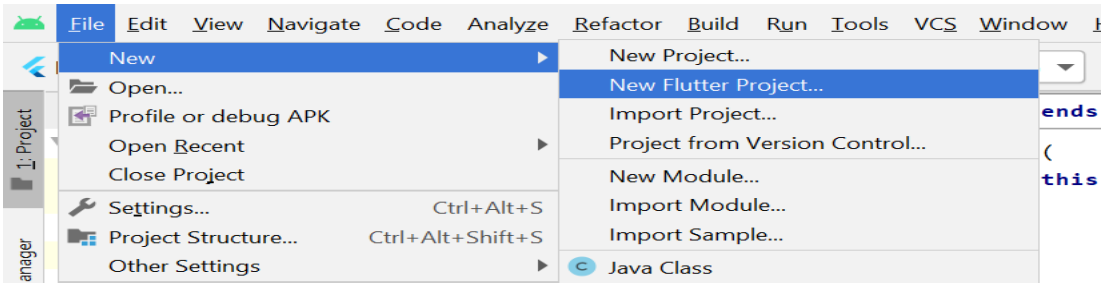
Lecture 7

Appendix

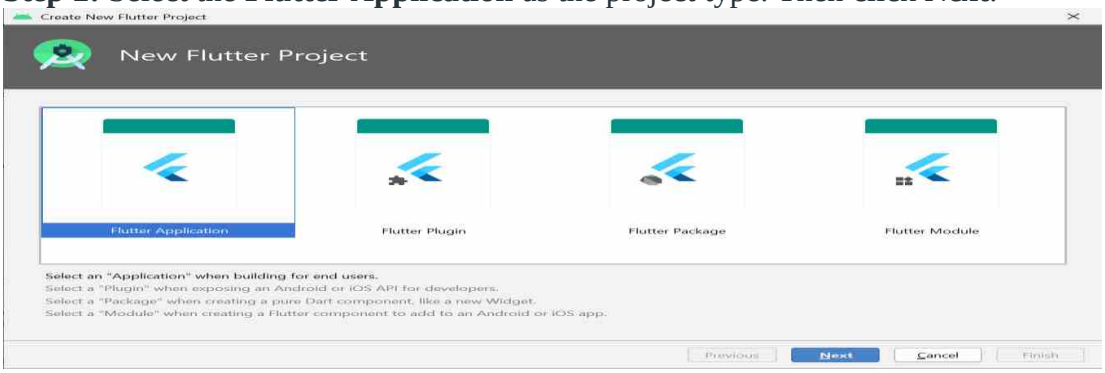
Create a New Flutter Project

Step 1: Open the **Android Studio IDE** and select Start a new **Flutter project**.

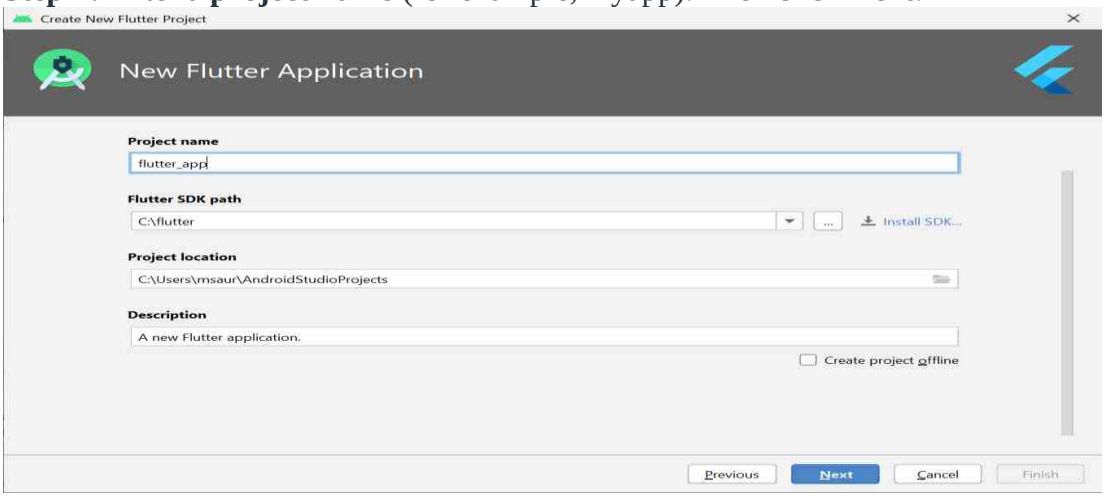
Note: *if you like to create a flutter project using terminal use the below command and jump right into* ***step 6***
$ ***flutter create flutter_app***
replace the ***' flutter_app '*** *with your project name*



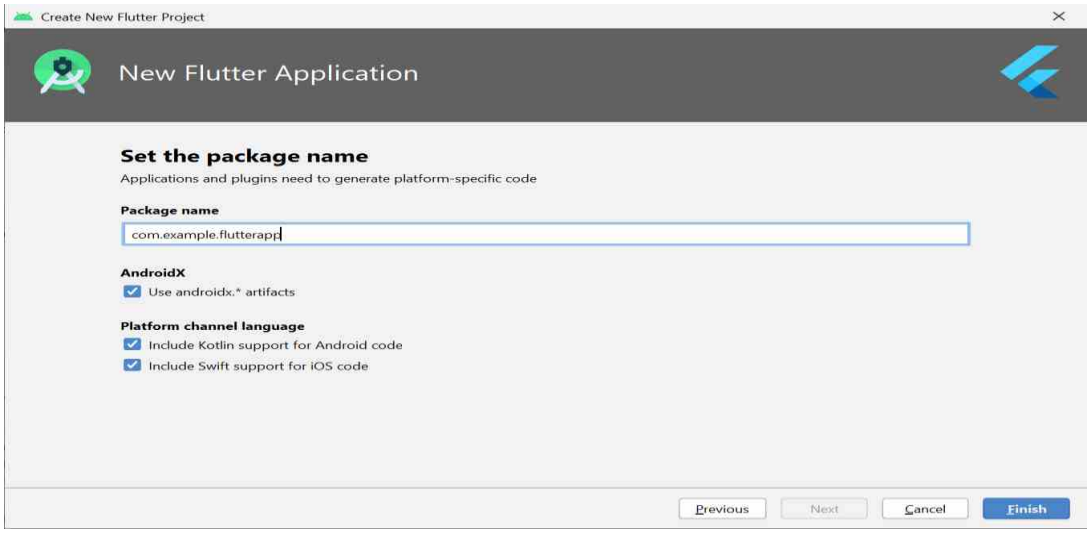
Step 2: Select the **Flutter Application** as the project type. Then click **Next**.



Step 3: Verify the **Flutter SDK path** specifies the SDK's location (**select Install SDK… if the text field is blank**).
Step 4: Enter a **project name** (for example, myapp). Then click **Next**.

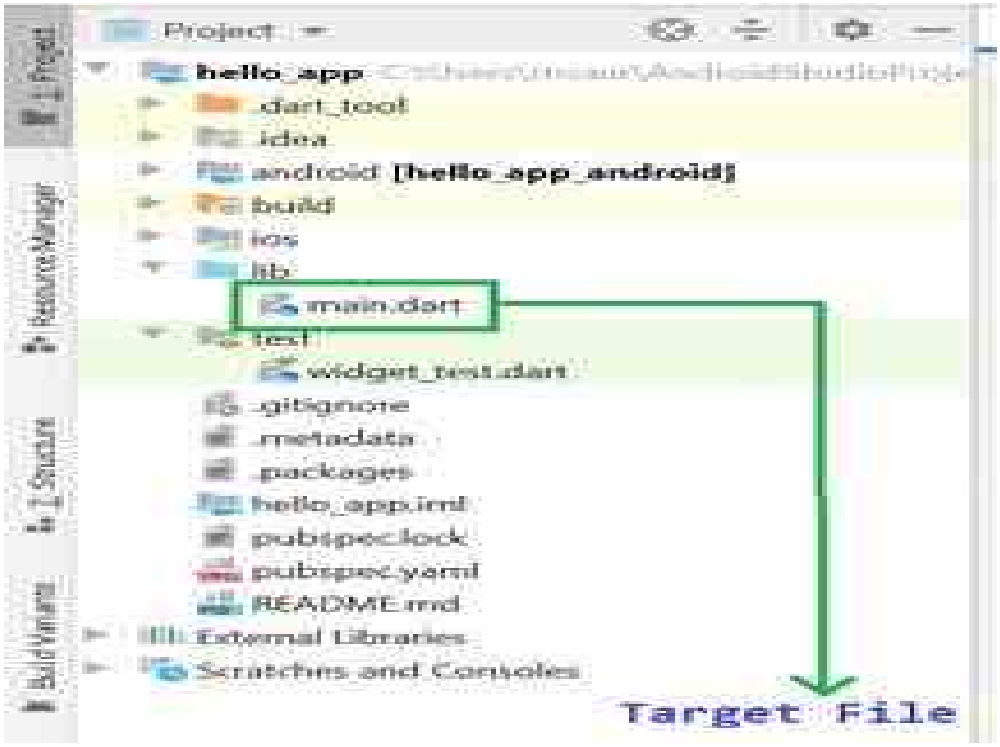


Note:
1. Project name: flutter_app
2. Flutter SDK Path: <path-to-flutter-sdk>
3. Project Location: <path-to-project-folder>
4. Description: Flutter based simple application
Step 5: Click **Finish** and wait till Android Studio creates the project.



Step 6: Edit the code
After successfully creating a file, we can edit the code of the application to show the output we want. Android Studio creates a fully working flutter application with minimal functionality. Let us check the structure of the application and then, change the code to do our task.

The structure of the application and its purpose are as follows?



We have to edit the code in ***main.dart*** as mentioned in the above image. We can see that Android Studio has automatically generated most of the files for our flutter app. Replace the dart code in the ***lib/main.dart*** file with the below code:

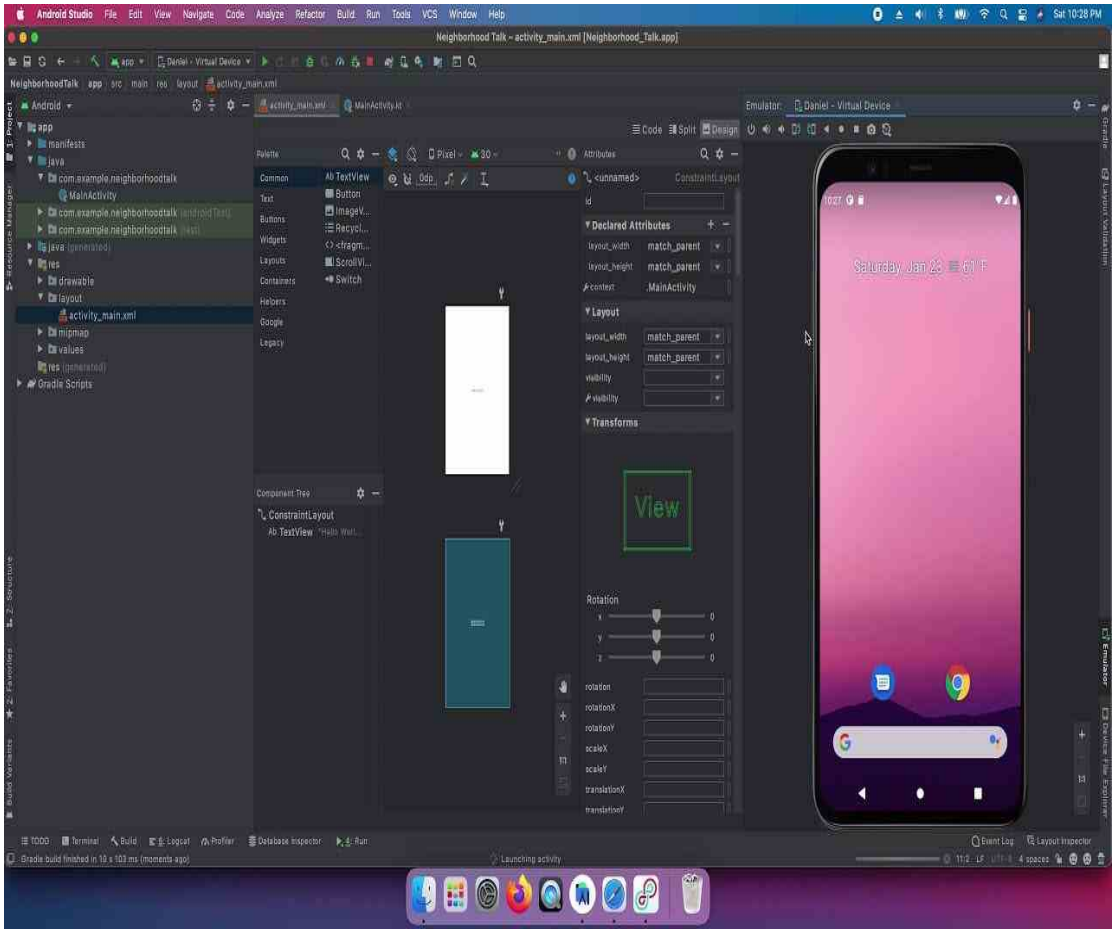
Example
// Importing important packages require to connect // Flutter and Dart

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

// Main Function
void main() {
// Giving command to runApp() to run the app.

/* The purpose of the runApp() function is to attach
the given widget to the screen. */
  runApp(const MyApp());
}

// Widget is used to create UI in flutter framework.

/* StatelessWidget is a widget, which does not maintain
any state of the widget. */

/* MyApp extends StatelessWidget and overrides its
build method. */
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

// This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(      // title of the application
      title: 'Hello World Demo Application',
      // theme of the widget
      theme: ThemeData(
        primarySwatch: Colors.lightGreen,      ),
      // Inner UI of the application
      home: const MyHomePage(title: 'Home page'),    );  }}

/* This class is similar to MyApp instead it
returns Scaffold Widget */
class MyHomePage extends StatelessWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);
  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      // Sets the content to the
      // center of the application page
      body: const Center(
          // Sets the content of the Application
          child: Text(
        'Welcome to Hello!',     )),    );  }}
```

Output:

Hello world using Flutter

Hello
import 'package:flutter/material.dart;' void main} () runApp(const hello;(() { class hello extends StatelessWidget} const hello({Key? key}) : super(key: key;(@override Widget build(BuildContext context} (return const MaterialApp) home: Center(child: Text('Hello World,((') {;(}


Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Neighborhood Talk – activity_main.xml [Neighborhood_Talk.app]
Daniel - Virtual Device
NeighborhoodTalk app src main res layout activity_main.xml
Android
activity_main.xml
MainActivity.kt
Code Split Design
Emulator: Daniel - Virtual Device
app
manifests
java
com.example.neighborhoodtalk
MainActivity
com.example.neighborhoodtalk (android Test)
com.example.neighborhoodtalk (test)
java (generated)
res
drawable
layout
activity_main.xml
mipmap
values
res (generated)
Gradle Scripts
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
Ab TextView
Button
ImageV...
Recycl...
<fragm...
ScrollVi...
Switch
Component Tree
ConstraintLayout
Ab TextView
Attributes
<unnamed>
ConstraintLayout
id
Declared Attributes
layout_width match_parent
layout_height match_parent
context .MainActivity
Layout
layout_width match_parent
layout_height match_parent
visibility
visibility
Transforms
View
Rotation
rotation
rotationX
rotationY
scaleX
scaleY
translationX
translationY
Saturday, Jan 25 61° F
TODO Terminal Build Logcat Profiler Database Inspector Run
Gradle build finished in 18 s 103 ms (moments ago)
Launching activity
Event Log Layout Inspector