EXP.NO:1 Study and installation of Flutter/Kotlin multi-platform environment

Aim:

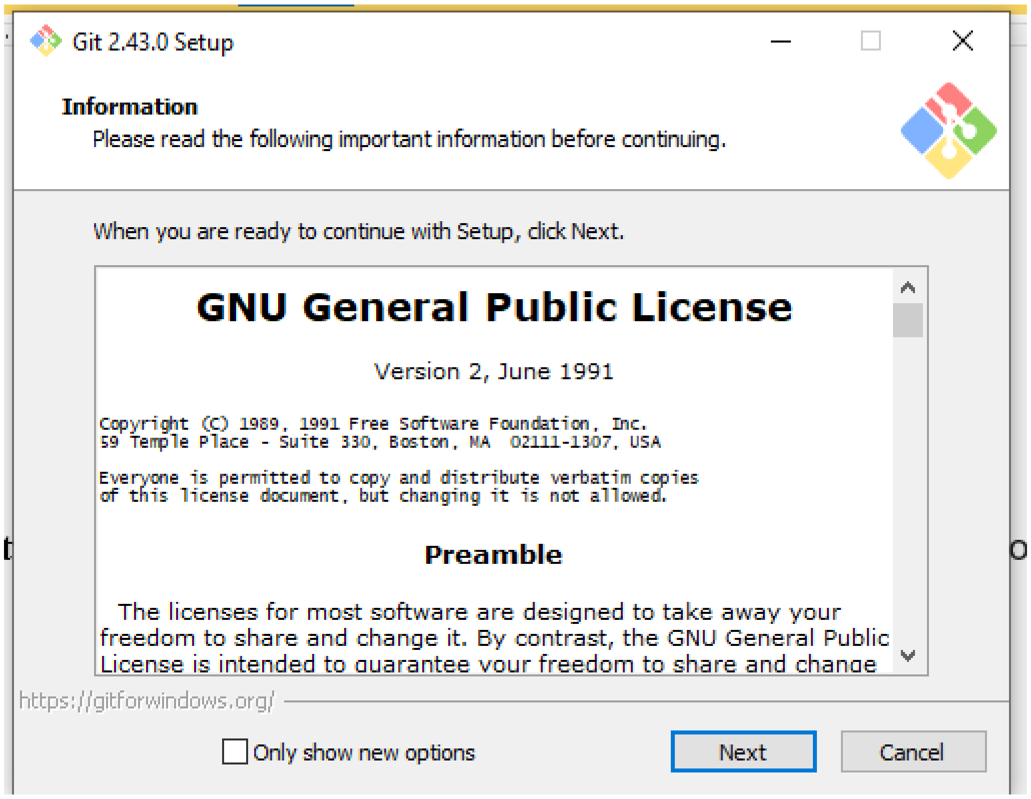
To install the flutter in Android Studio

Procedure:

Step 1: **Installation of GIT Application**

Google->type Flutter Install Download->select Flutter SDK->Choose Windows->Desktop->under Development tools

->choose **Git for Windows 2.4**



Git 2.43.0 Setup

Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

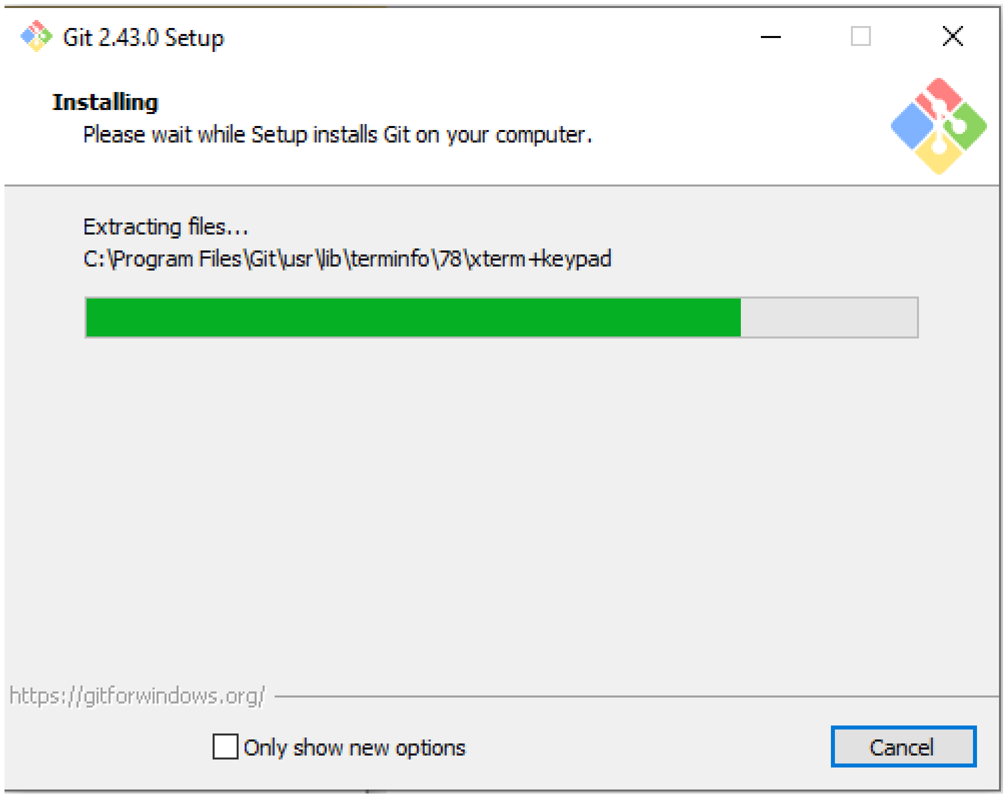
```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Click next->

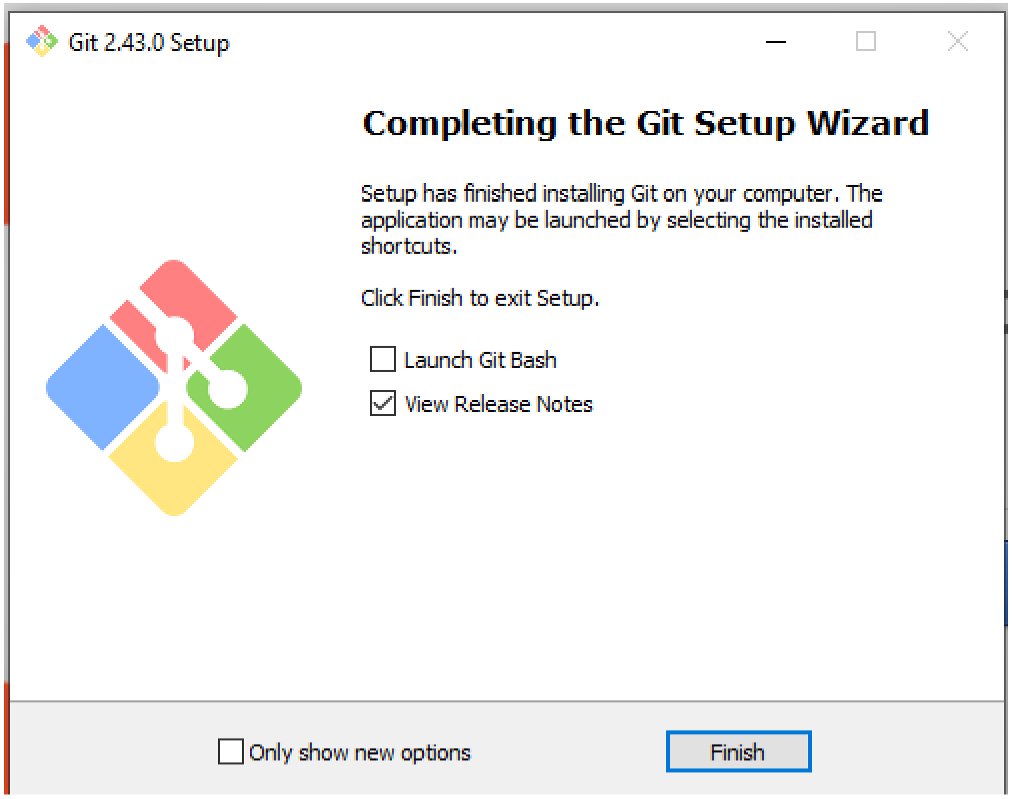

Git 2.43.0 Setup

Installing

Please wait while Setup installs Git on your computer.

Extracting files...
C:\Program Files\Git\usr\lib\terminfo\78\xterm+keypad

https://gitforwindows.org/


Git 2.43.0 Setup

Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

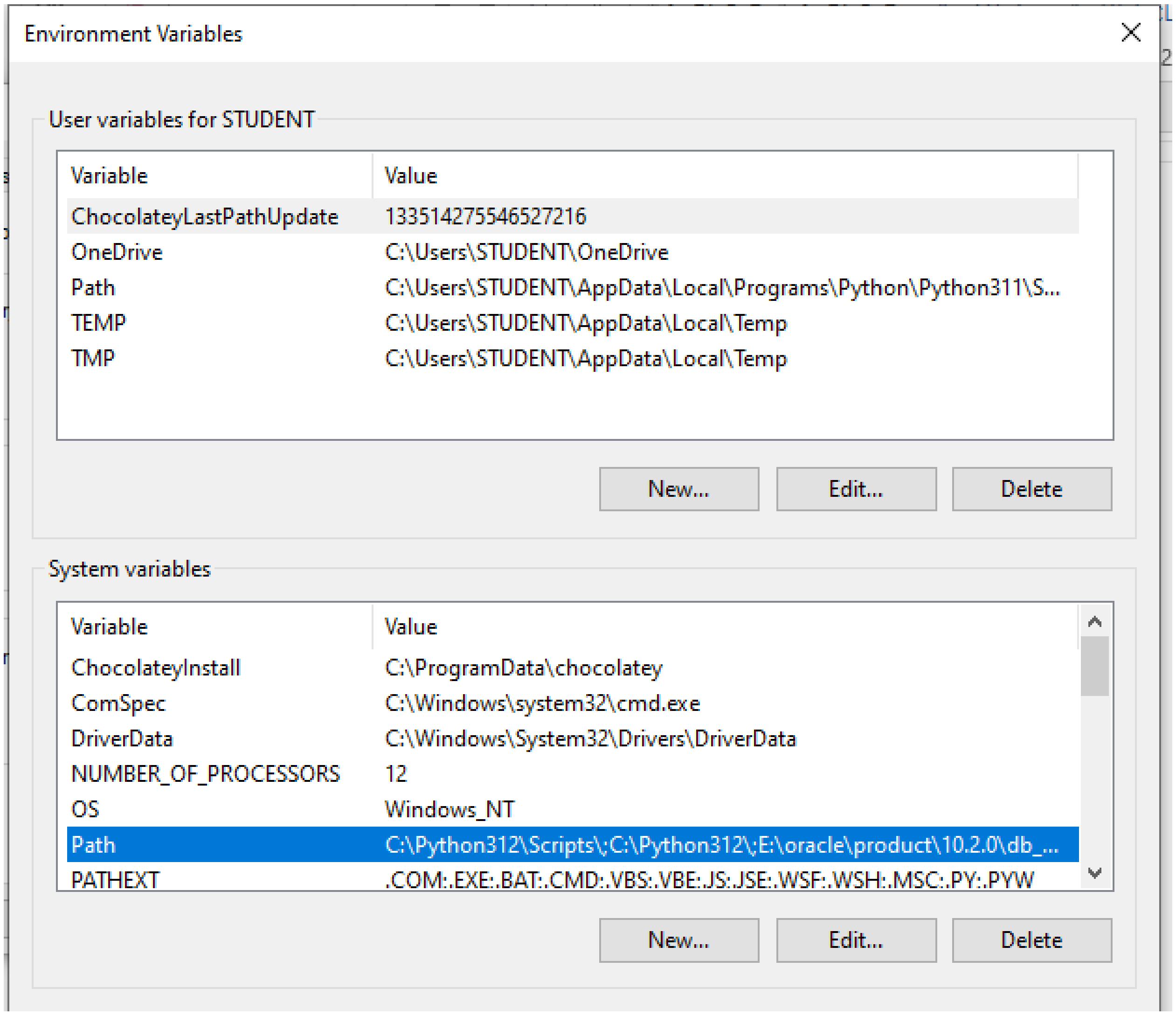
Click Finish to exit Setup.

☐ Launch Git Bash
☑ View Release Notes

Click finish

Step 2: Set the Environment Variable

Goto Start menu->Open Environmental Variable->choose path->edit->new-> paste the address c:\Program files\Git\cmd


Environment Variables

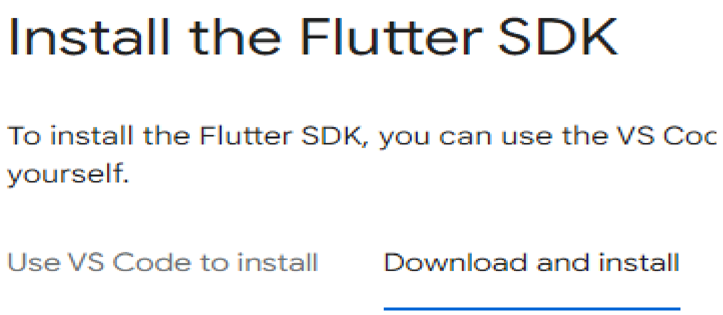
User variables for STUDENT

Variable	Value
ChocolateyLastPathUpdate	133514275546527216
OneDrive	C:\Users\STUDENT\OneDrive
Path	C:\Users\STUDENT\AppData\Local\Programs\Python\Python311\S...
TEMP	C:\Users\STUDENT\AppData\Local\Temp
TMP	C:\Users\STUDENT\AppData\Local\Temp

System variables

Variable	Value
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\Python312\Scripts\;C:\Python312\;E:\oracle\product\10.2.0\db_...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW

Step 3: Installation of Flutter SDK


Install the Flutter SDK

To install the Flutter SDK, you can use the VS Coc
yourself.

Download the following installation bundle to get the latest stable release of the Flutter SDK.

flutter_windows_3.16.9-stable.zip

then extract that zip file ->copy the Flutter Folder->goto c:->program files->create one new folder named as src->paste Flutter folder here

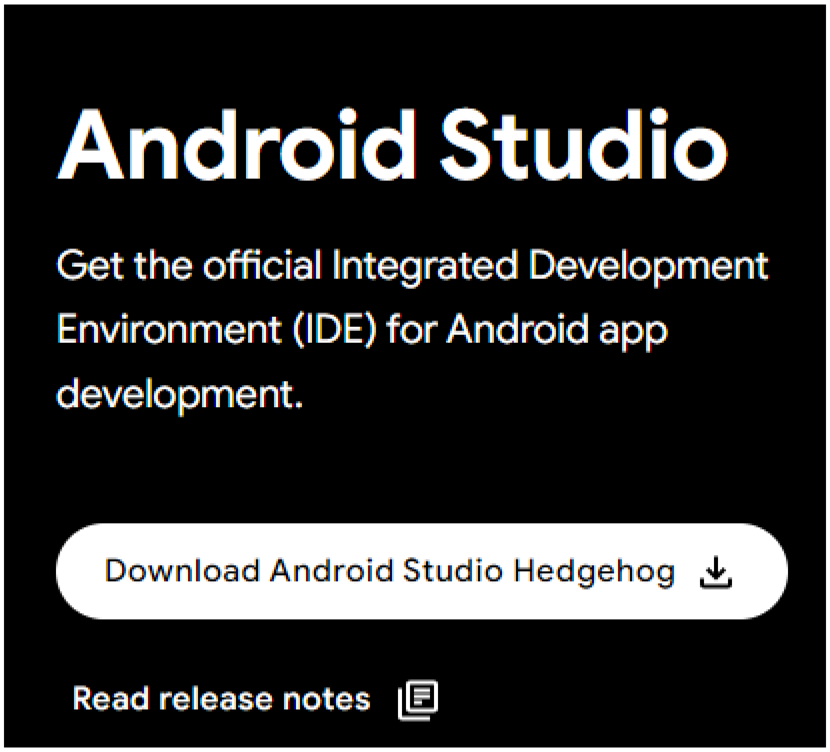
then copy the address ->set environment variable again which is same as git path

Step **3:Configure a text editor or IDE**

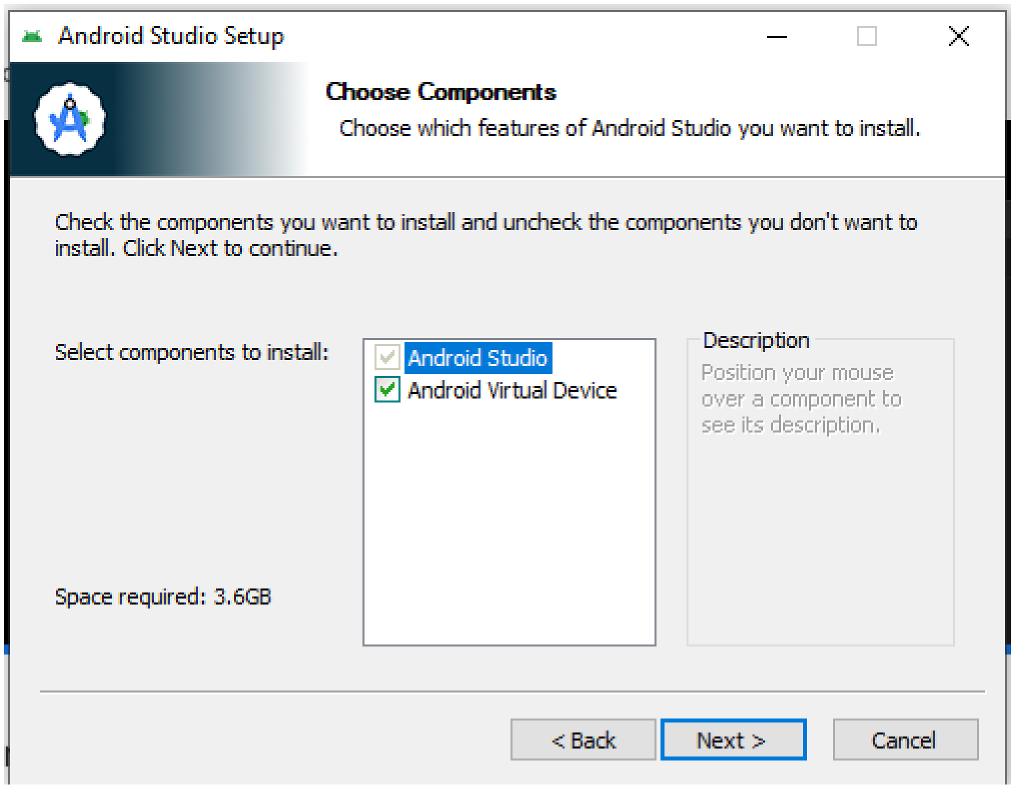
Goto on same web page Android Studio 2022.3 (Giraffe) or later with the Flutter plugin for IntelliJ. Click on it

Install Android Studio

Click Then download the latest version of Android Studio.


Android Studio
Get the official Integrated Development Environment (IDE) for Android app development.
Download Android Studio Hedgehog
Read release notes

Once its get downloaded ->click on it->yes->next->check


Android Studio Setup

Choose Components

Choose which features of Android Studio you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

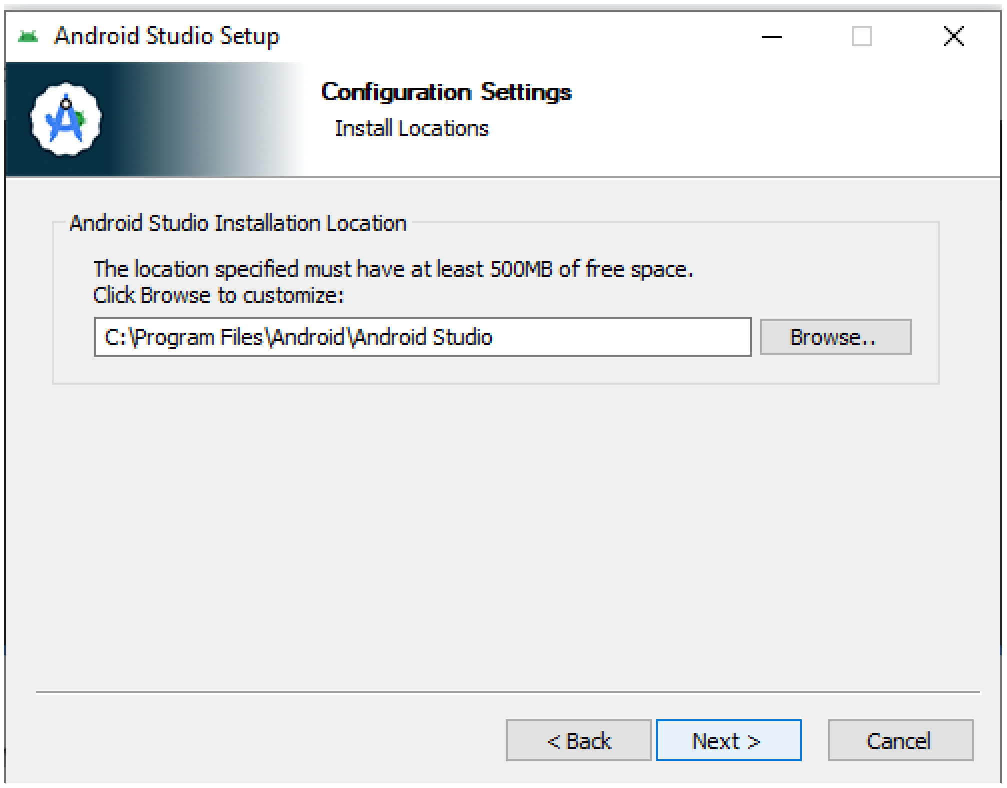
- [✓] Android Studio
- [✓] Android Virtual Device

Description

Position your mouse over a component to see its description.

Space required: 3.6GB

Click next->


Android Studio Setup

Configuration Settings

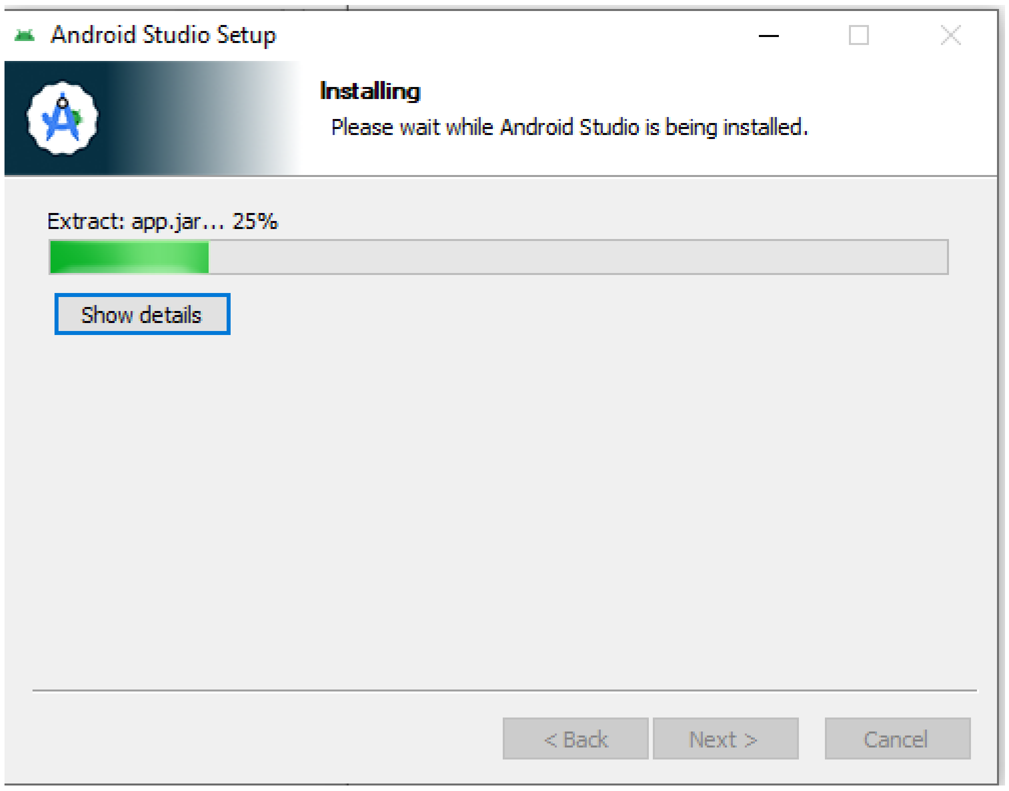
Install Locations

Android Studio Installation Location

The location specified must have at least 500MB of free space.
Click Browse to customize:

C:\Program Files\Android\Android Studio

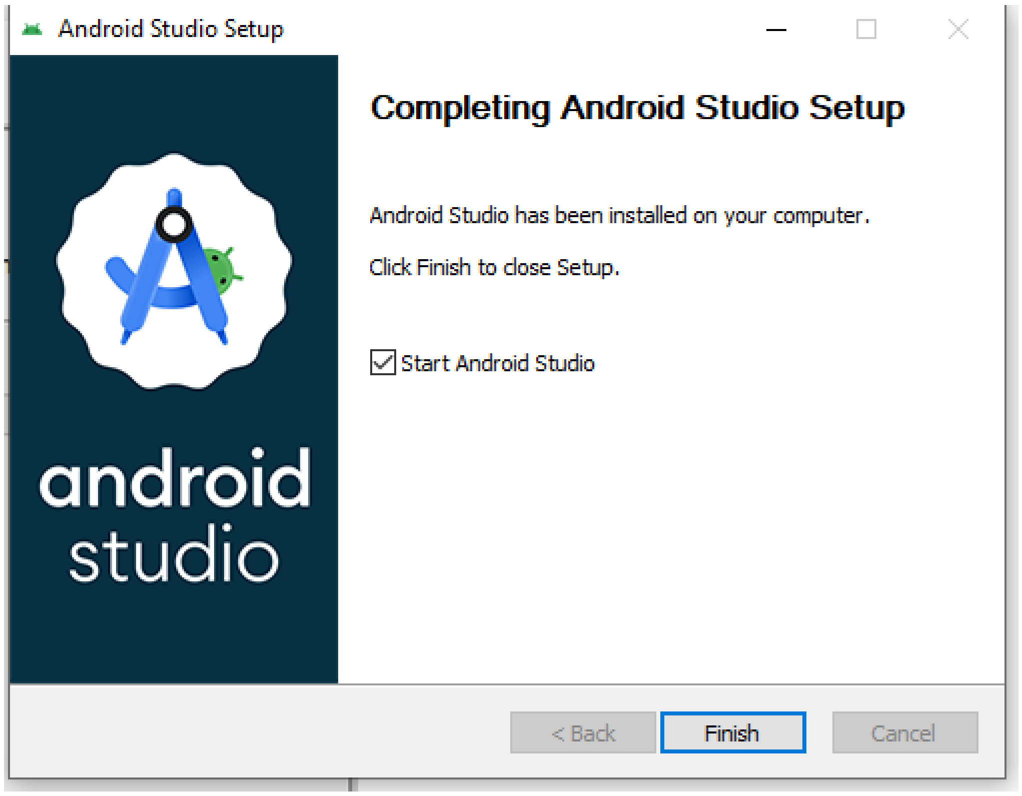
Click next->install


Android Studio Setup

Installing

Please wait while Android Studio is being installed.

Extract: app.jar... 25%


Android Studio Setup

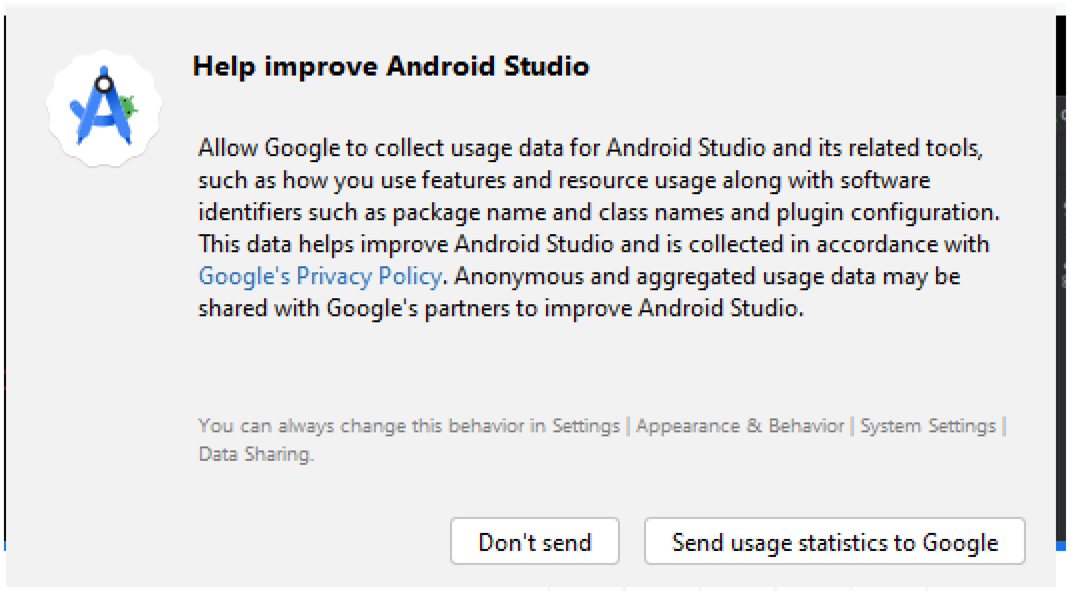
Completing Android Studio Setup

Android Studio has been installed on your computer.

Click Finish to close Setup.

☑ Start Android Studio

Then click finish->send usage statics to google

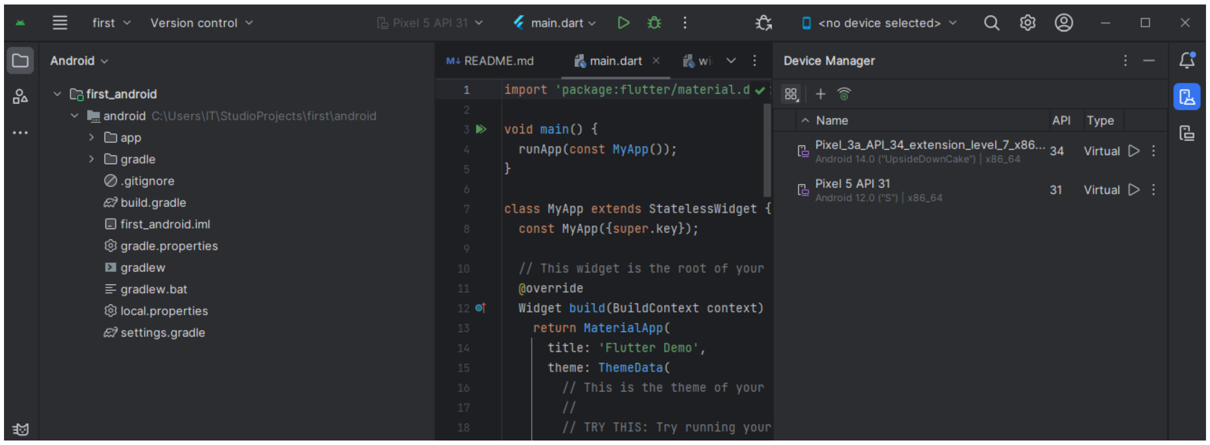

Help improve Android Studio

Allow Google to collect usage data for Android Studio and its related tools, such as how you use features and resource usage along with software identifiers such as package name and class names and plugin configuration. This data helps improve Android Studio and is collected in accordance with Google's Privacy Policy. Anonymous and aggregated usage data may be shared with Google's partners to improve Android Studio.

You can always change this behavior in Settings | Appearance & Behavior | System Settings | Data Sharing.

Step 4: Set the emulator to get output sytle

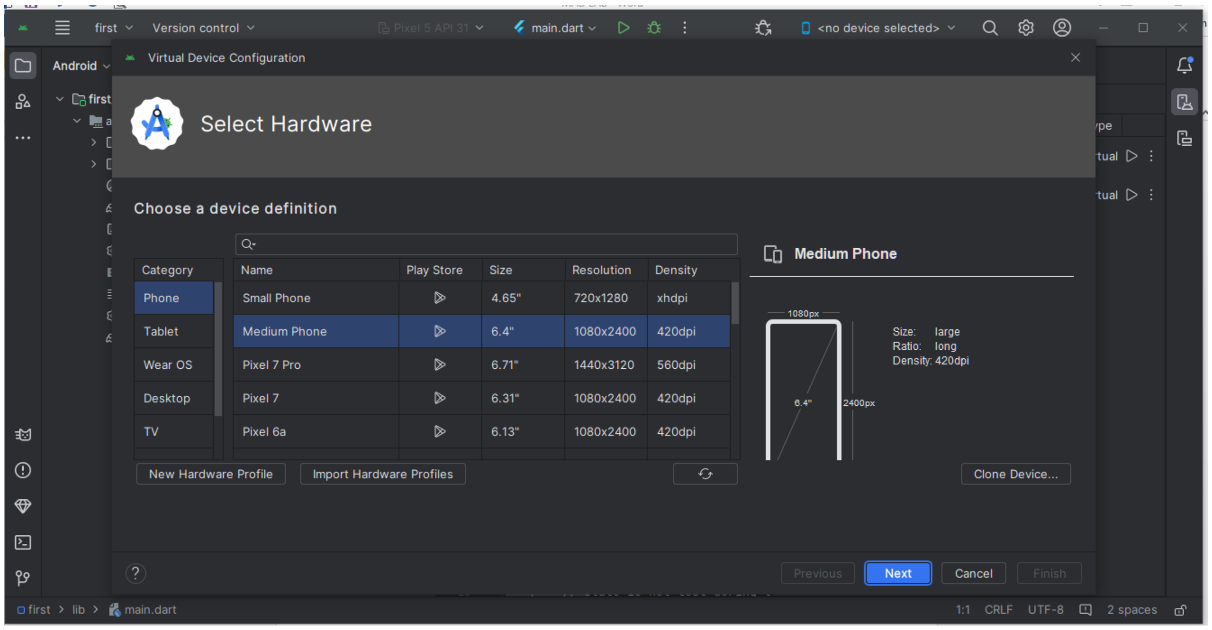
GOTO Device manager->click + sysmbol->



```
import 'package:flutter/material.d

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your
  @override
  Widget build(BuildContext context)
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your
        //
        // TRY THIS: Try running your
```

Name	API	Type
Pixel_3a_API_34_extension_level_7_x86... Android 14.0 ("UpsideDownCake") | x86_64	34	Virtual
Pixel 5 API 31 Android 12.0 ("S") | x86_64	31	Virtual

Click-> next


Virtual Device Configuration

Select Hardware

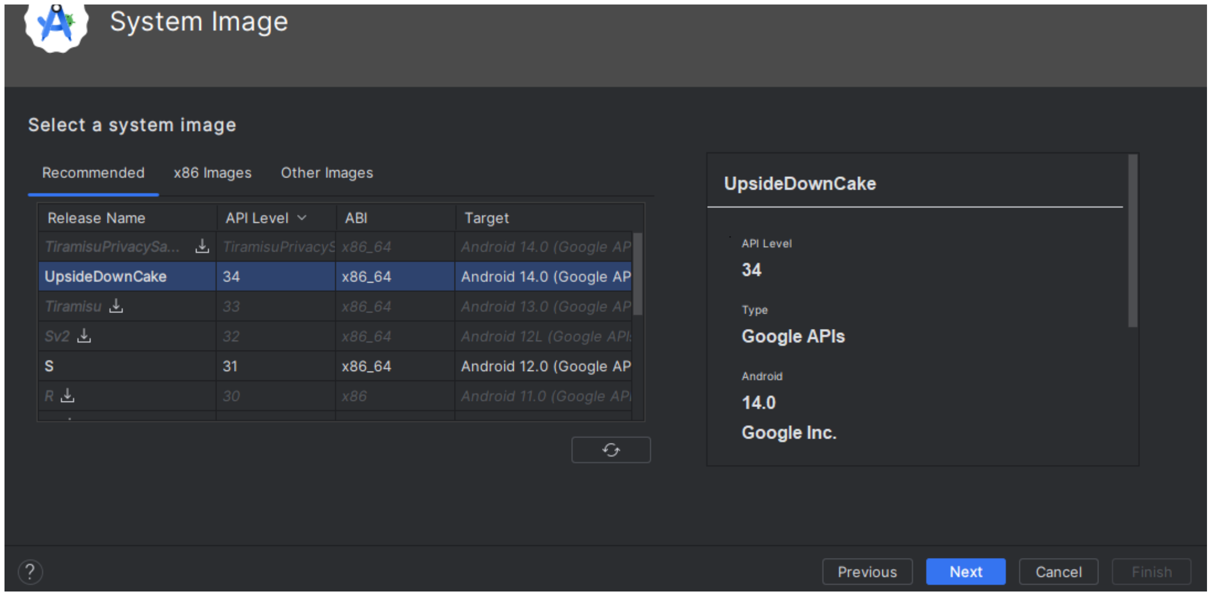
Choose a device definition

Category	Name	Play Store	Size	Resolution	Density
Phone	Small Phone	▷	4.65"	720x1280	xhdpi
Tablet	Medium Phone	▷	6.4"	1080x2400	420dpi
Wear OS	Pixel 7 Pro	▷	6.71"	1440x3120	560dpi
Desktop	Pixel 7	▷	6.31"	1080x2400	420dpi
TV	Pixel 6a	▷	6.13"	1080x2400	420dpi

Medium Phone

Size: large
Ratio: long
Density: 420dpi

Select phone->pixel 5->next->


System Image

Select a system image

Release Name	API Level	ABI	Target
TiramisuPrivacySa...	TiramisuPrivacyS	x86_64	Android 14.0 (Google AP
UpsideDownCake	34	x86_64	Android 14.0 (Google AP
Tiramisu	33	x86_64	Android 13.0 (Google AP
Sv2	32	x86_64	Android 12L (Google API
S	31	x86_64	Android 12.0 (Google AP
R	30	x86	Android 11.0 (Google API

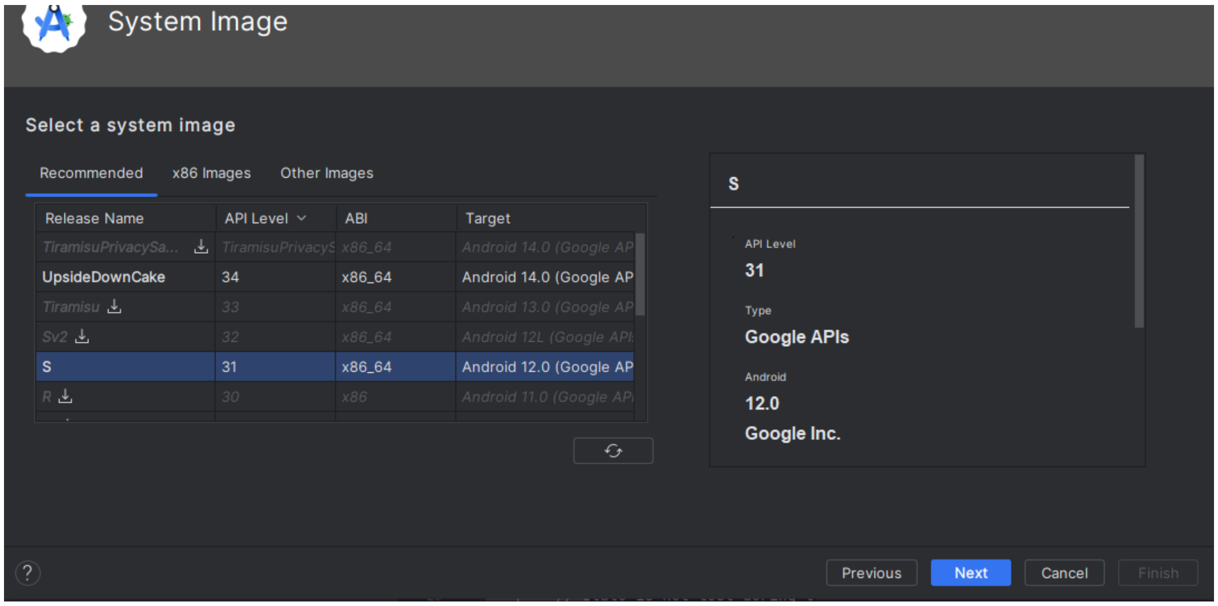
UpsideDownCake

API Level
34

Type
Google APIs

Android
14.0
Google Inc.

Click and select S download->next



System Image

Select a system image

Release Name	API Level	ABI	Target
TiramisuPrivacySa...	TiramisuPrivacyS	x86_64	Android 14.0 (Google AP
UpsideDownCake	34	x86_64	Android 14.0 (Google AP
Tiramisu	33	x86_64	Android 13.0 (Google AP
Sv2	32	x86_64	Android 12L (Google API
S	31	x86_64	Android 12.0 (Google AP
R	30	x86	Android 11.0 (Google AP

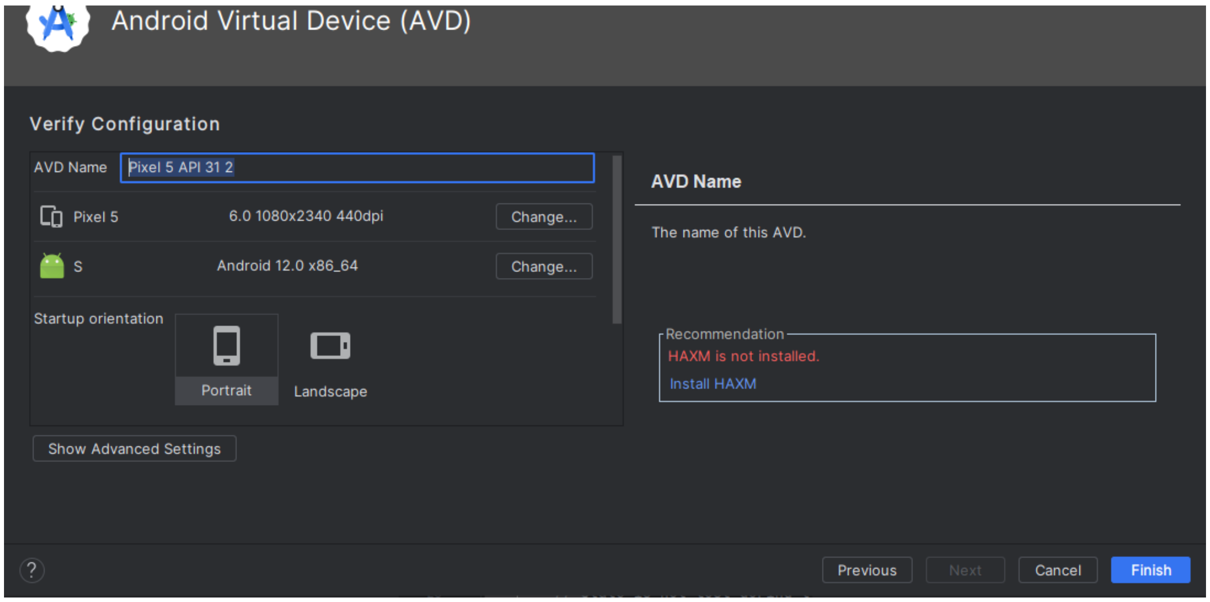
S

API Level
31

Type
Google APIs

Android
12.0
Google Inc.

Then give AVD Name->click finish



Android Virtual Device (AVD)

Verify Configuration

AVD Name: Pixel 5 API 31 2

Pixel 5 — 6.0 1080x2340 440dpi

S — Android 12.0 x86_64

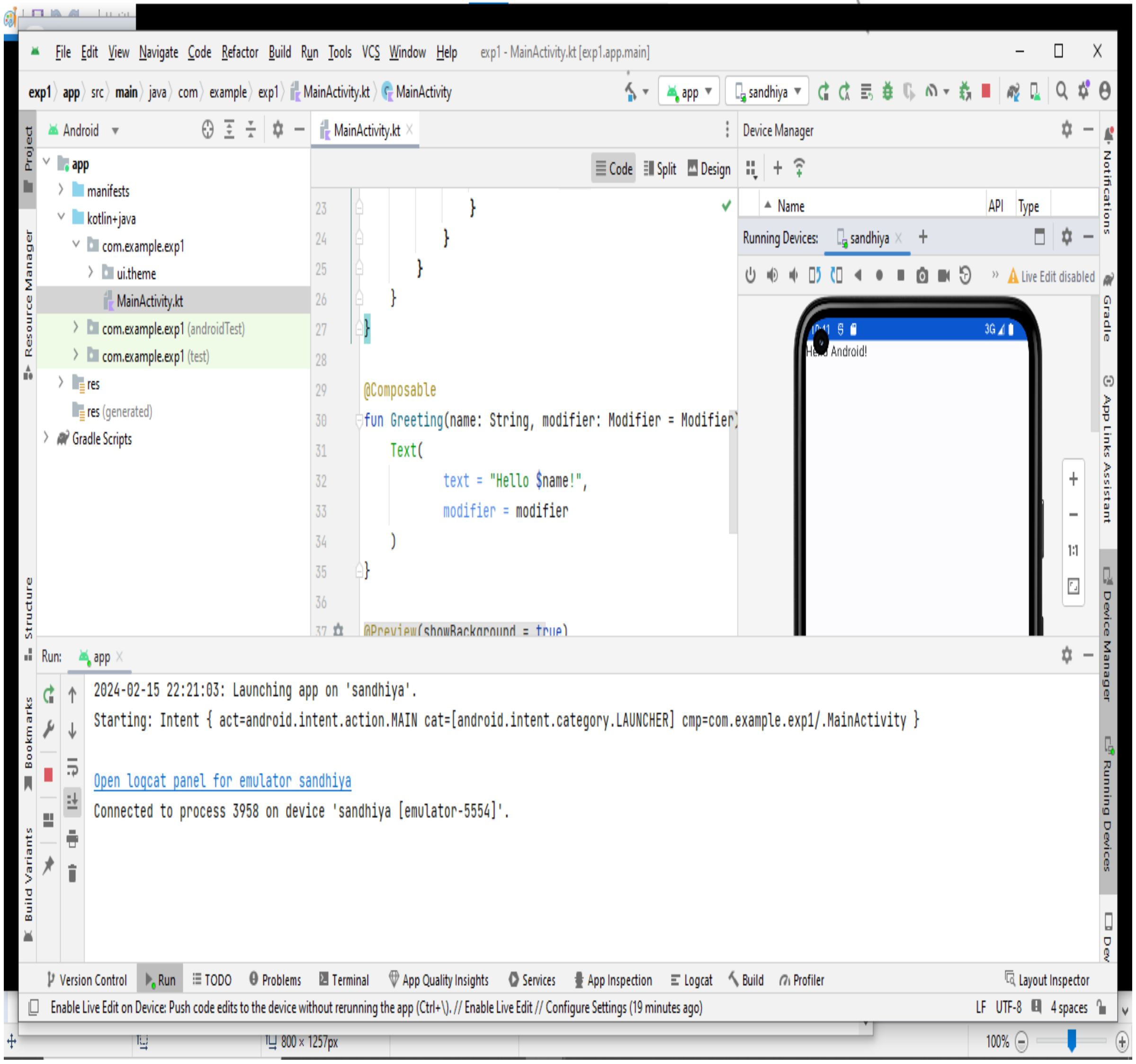
Startup orientation: Portrait / Landscape

AVD Name

The name of this AVD.

Recommendation
HAXM is not installed.
Install HAXM

Step 5:Once Installation Completed then you can run the Dart Program



```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

@Preview(showBackground = true)
```

```
2024-02-15 22:21:03: Launching app on 'sandhiya'.
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.example.exp1/.MainActivity }

Open logcat panel for emulator sandhiya
Connected to process 3958 on device 'sandhiya [emulator-5554]'.
```

Enable Live Edit on Device: Push code edits to the device without rerunning the app (Ctrl+\). // Enable Live Edit // Configure Settings (19 minutes ago)

RESULT:

Thus the Installation of Flutter/Kotlin Software Installed Successfully

Exp 2: Develop a application that uses Widgets, GUI component, fonts

Aim:

Creating an application that utilizes widgets, GUI components, and fonts can be a fun and educational project.

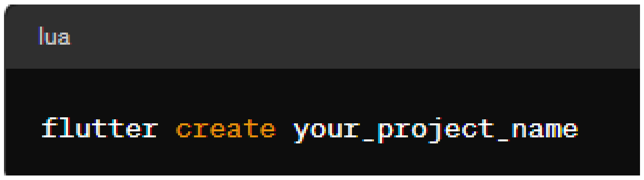
Procedure:

Setup Flutter Development Environment:

- Install Flutter by following the official documentation: Flutter Installation Guide
- Set up an IDE for Flutter development. Popular choices include Android Studio with the Flutter plugin, Visual Studio Code with the Flutter extension, or IntelliJ IDEA.

Create a New Flutter Project:

- Use the Flutter CLI or IDE to create a new project:



```
flutter create your_project_name
```

Design the User Interface:

- Define the UI layout using Flutter's widget system. Flutter provides a wide range of widgets for building UIs, such as Container, Row, Column, Text, Button, etc.
- Specify fonts for different text elements using the TextStyle class.

Implement GUI Components:

- Write Dart code to create and configure GUI components based on your UI design.
- Use widgets to handle user interactions and display information.

Customize Fonts:

- Specify custom fonts for text elements using the TextStyle class.
- Add font files (e.g., .ttf or .otf files) to your Flutter project and reference them in your code.

Implement Functionality:

- Write Dart code to add functionality to your application. This may include event handling, data processing, state management, etc.
- Bind GUI components to your functionality to update the UI based on user input or data changes.

Testing:

- Test your application on different devices and screen sizes to ensure it works correctly.
- Use Flutter's built-in testing framework or third-party testing libraries for automated testing.

Refinement:

- Refine your application based on user feedback and testing results. This may involve improving UI/UX, **optimizing performance, fixing bugs, etc.**

Documentation and Deployment:

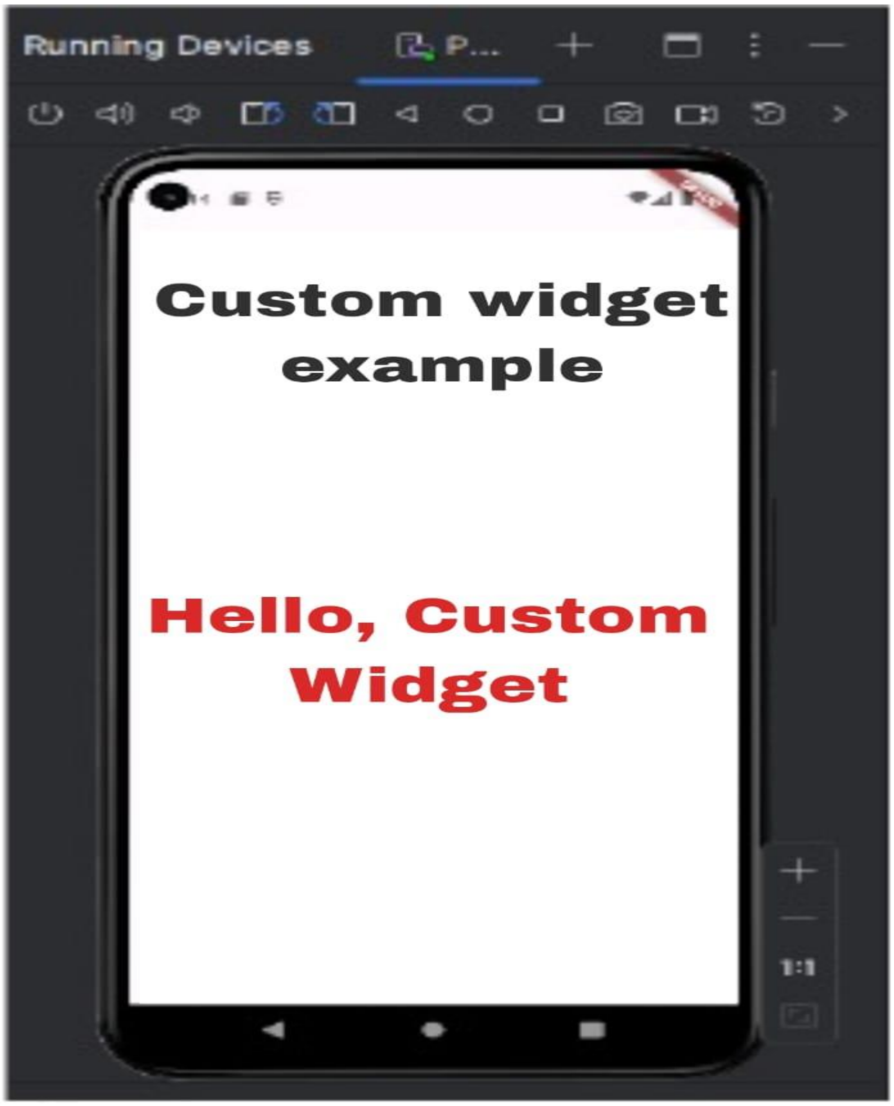
- Document your code and application functionality for future reference.
- Deploy your Flutter application to various platforms (e.g., Android, iOS, web, desktop) using Flutter's build tools and platform-specific configurations.

Coding:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Custom Widget Example',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        fontFamily: 'Roboto', // Setting custom font for the entire app
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('Custom Widget Example'),
        ),
        body: Center(
          child: CustomWidget(
            text: 'Hello, Custom Widget!',
            color: Colors.red, // Custom color
          ),
        ),
      ),
    );
  }
}
```

```
class CustomWidget extends StatelessWidget {
  final String text;
  final Color color;

  const CustomWidget({
    Key? key,
    required this.text,
    required this.color,
  }) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.all(16.0),
      decoration: BoxDecoration(
        color: Colors.grey[200],
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: Text(
        text,
        style: TextStyle(
          fontSize: 24.0,
          color: color, // Using custom color
          fontWeight: FontWeight.bold,
        ),
      ),
    );
  }
}
```

OUTPUT:



Custom widget example

Hello, Custom Widget

RESULT:

Thus the above application using Widgets, GUI component, fonts has been created successfully

Ex3: Develop a native calculator application

Aim:

To develop Flutter-based calculator application Using Dart and Flutter's cross-platform capabilities.

Procedure:

Initialize Variables:

Initialize variables to store the input expression (as a string), current value, previous value, and operator.

User Interface Setup:

- Design the user interface using Flutter widgets such as Column, Row, Container, Text, and FlatButton.
- Arrange buttons for digits (0-9), decimal point, basic arithmetic operators (+, -, *, /), equals (=), clear (C), and delete (DEL).
- Set up the layout to display the input expression and the current result.

Button Click Handling:

- Attach event handlers to buttons to capture user input.
- Update the input expression based on the user's button clicks.
- Implement functionality to handle special buttons like clear, delete, and equals.

Expression Parsing and Evaluation:

- Implement a function to parse the input expression into operands and operators.
- Evaluate the expression using Dart's built-in num functions or a custom expression evaluator.
- Handle errors and edge cases such as division by zero.

Display Result:

- Update the display to show the current input expression and the result of the calculation.
- Format the result to handle decimal places and large numbers appropriately.
- Memory Functions (Optional):
- Implement memory functions like memory add (M+), memory subtract (M-), and memory recall (MR) if desired.

Error Handling:

- Implement error handling to catch and display any errors that occur during calculation or input parsing.

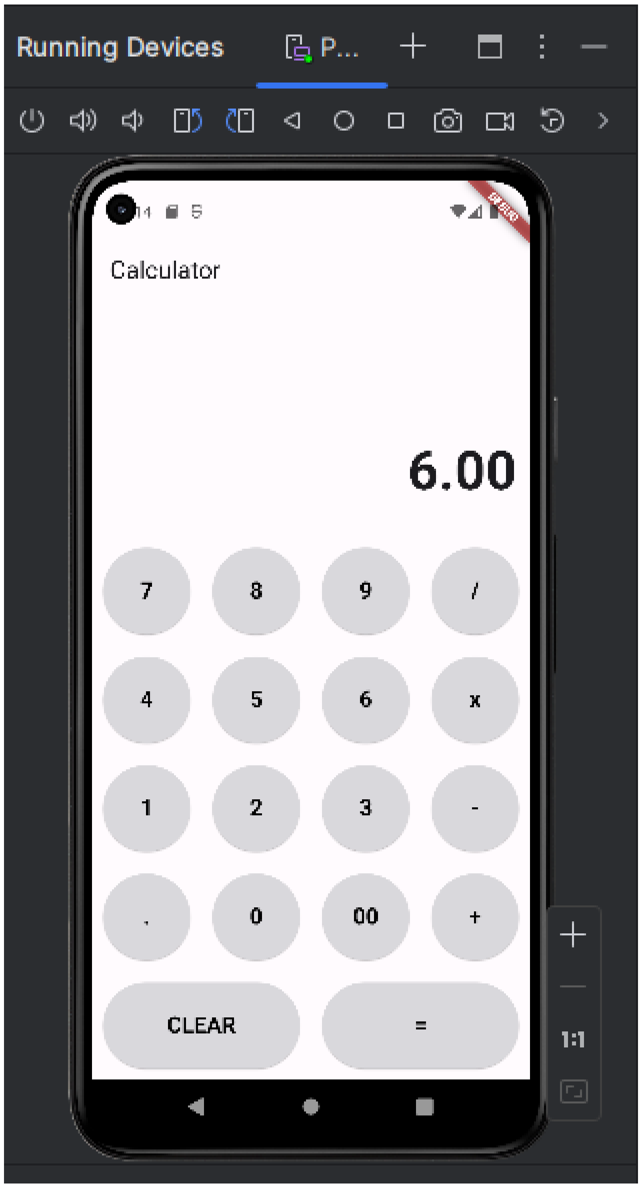
Coding:

Main.dart:

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

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

class CalculatorAppextends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
      title: 'Calculator',
      theme: ThemeData(
primarySwatch: Colors.blue,
      ),
      home: Calculator(),
    );
  }
}

class Calculator extends StatefulWidget {
@override
_CalculatorStatecreateState() =>_CalculatorState();
}

class _CalculatorStateextends State<Calculator> {
  String output = "0";
  String _output = "0";
  double num1 = 0.0;
  double num2 = 0.0;
  String operand = "";

buttonPressed(String buttonText) {
if (buttonText == "CLEAR") {
_output = "0";
num1 = 0.0;
num2 = 0.0;
operand = "";
    } else if (buttonText == "+" ||
buttonText == "-" ||
buttonText == "/" ||
buttonText == "x") {
num1 = double.parse(output);
operand = buttonText;
```

```
_output = "0";
    } else if (buttonText == ".") {
if (_output.contains(".")) {
return;
      } else {
_output = _output + buttonText;
      }
    } else if (buttonText == "=") {
num2 = double.parse(output);
if (operand == "+") {
_output = (num1 + num2).toString();
      }
if (operand == "-") {
_output = (num1 - num2).toString();
      }
if (operand == "x") {
_output = (num1 * num2).toString();
      }
if (operand == "/") {
_output = (num1 / num2).toString();
      }
num1 = 0.0;
num2 = 0.0;
operand = "";
    } else {
_output = _output + buttonText;
    }

setState(() {
output = double.parse(_output).toStringAsFixed(2);
    });
  }

  Widget buildButton(String buttonText) {
return Expanded(
      child: Padding(
        padding: EdgeInsets.all(10.0),
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            primary: Colors.grey[300], // background color
onPrimary: Colors.black, // foreground color
padding: EdgeInsets.all(24.0),
          ),
onPressed: () =>buttonPressed(buttonText),
          child: Text(
buttonText,
            style: TextStyle(
fontSize: 20.0,
fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    );
  }

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
        title: Text("Calculator"),
      ),
      body: Container(
        child: Column(
          children: <Widget>[
Expanded(
              child: Container(
                padding: EdgeInsets.symmetric(vertical: 24.0, horizontal: 12.0),
                alignment: Alignment.bottomRight,
                child: Text(
output,
                  style: TextStyle(
fontSize: 48.0,
fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
Column(
              children: <Widget>[
Row(
                  children: <Widget>[
buildButton("7"),
buildButton("8"),
buildButton("9"),
buildButton("/"),
                  ],
                ),
Row(
                  children: <Widget>[
buildButton("4"),
buildButton("5"),
buildButton("6"),
buildButton("x"),
                ],
              ),
Row(
                children: <Widget>[
buildButton("1"),
buildButton("2"),
buildButton("3"),
buildButton("-"),
                ],
              ),
Row(
                children: <Widget>[
buildButton("."),
buildButton("0"),
buildButton("00"),
buildButton("+"),
                ],
              ),
Row(
                children: <Widget>[
buildButton("CLEAR"),
buildButton("="),
                ],
              ),
            ],
          ),
        ],
      ),
    ),
  );
 }
}
```

OUTPUT:



RESULT:

Thus the above Dart Program for developing a native calculator application has been implemented successfully and verified

Exp: 4(a) Develop A Gaming Application That Uses 2-D Animations And Gestures.

Aim:

To develop flutter based A Gaming Application That Uses 2-D Animations and Gestures using dart language.

Procedure:

Initialize the Game Environment:

- Set up the initial game environment, including the game screen, player characters, obstacles, and any other elements necessary for gameplay.

Implement Gesture Recognition:

- Use Flutter's GestureDetector widget to recognize various gestures such as tap, drag, swipe, pinch, etc.
- Define callback functions to handle each type of gesture detected.

Animate Game Elements:

- Utilize Flutter's animation features to animate game elements such as player characters, enemies, power-ups, etc.
- Implement animations using AnimationController, Tween, and other animation-related classes provided by Flutter.

Define Game Logic:

- Implement the core game logic, including collision detection, scoring system, level progression, etc.
- Define classes and functions to handle game events such as player movement, enemy behavior, and interaction with game objects.

Integrate Gestures with Game Logic:

- Map detected gestures to corresponding actions in the game logic.
- For example, a swipe gesture might move the player character left or right, a tap gesture might make the character jump, etc.

Handle User Input:

- Process user input received through gestures and update the game state accordingly.
- Trigger animations, update player positions, apply game mechanics, etc., based on user input.

Implement 2D Animations:

- Design and create 2D animations for game elements using tools like Flutter's built-in animation capabilities, SpriteWidget, Flare, Rive, etc.
- Incorporate animations for player movement, enemy actions, background effects, etc., to enhance the visual appeal of the game.

Optimize Performance:

- Optimize the game's performance by minimizing unnecessary computations, reducing memory usage, and optimizing rendering processes.
- Use techniques like object pooling, texture atlases, and efficient data structures to improve performance.

Test and Debug:

- Test the game thoroughly to ensure smooth gameplay, accurate gesture recognition, and correct animation behavior.
- Debug any issues related to gameplay mechanics, animation glitches, or gesture responsiveness.

Polish and Refine:

- Polish the game by adding sound effects, background music, particle effects, and other visual/audio enhancements.
- Refine gameplay mechanics based on user feedback and playtesting results to improve overall user experience.

Documentation and Deployment:

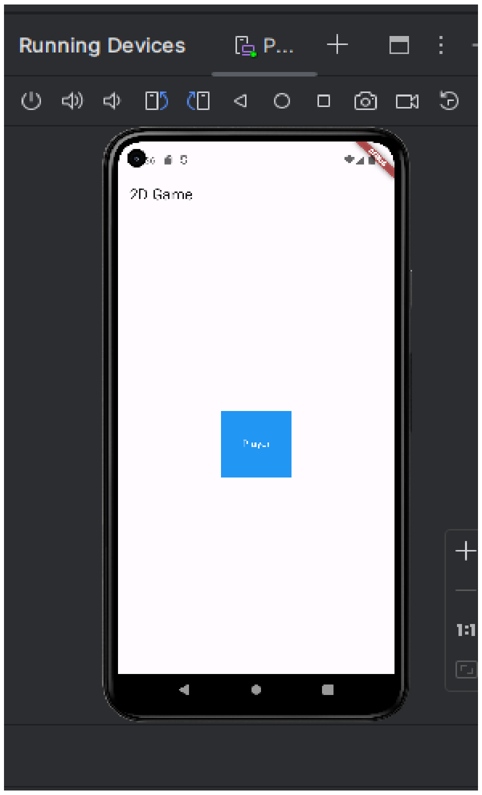
- Provide comprehensive documentation for developers and users, including instructions on how to play the game, troubleshooting tips, and information about gestures and animations used.

Coding : 2D game

Main.dart:

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

void main() =>runApp(GameApp());

class GameAppextends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
      home: GameScreen(),
    );
  }
}
class GameScreenextends StatefulWidget {
@override
_GameScreenStatecreateState() =>_GameScreenState();
}
class _GameScreenStateextends State<GameScreen>with SingleTickerProviderStateMixin {
late AnimationController_controller; // Added late keyword
late Animation<Offset>_offsetAnimation; // Added late keyword

@override
void initState() {
super.initState();
_controller = AnimationController(
vsync: this,
      duration: Duration(seconds: 1),
    );
_offsetAnimation= Tween<Offset>(
      begin: Offset.zero,
      end: Offset(2.0, 0.0),
    ).animate(CurvedAnimation(
      parent: _controller,
      curve: Curves.ease,
    ));
  }
@override
void dispose() {
_controller.dispose();
super.dispose();
  }
void _handleSwipe(DragUpdateDetails details) {
final double delta = details.primaryDelta?? 0; // Handle null
if (delta >0) {
// Swipe right
_controller.forward();
    } else if (delta <0) {
// Swipe left
_controller.reverse();
    }
  }

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
        title: Text('2D Game'),
      ),
      body: GestureDetector(
onHorizontalDragUpdate: _handleSwipe,
        child: Center(
          child: SlideTransition(
            position: _offsetAnimation,
            child: Container(
              width: 100.0,
              height: 100.0,
              color: Colors.blue,
              child: Center(
                child: Text(
'Player',
                  style: TextStyle(color: Colors.white),
                ),        ),
            ),         ),
        ),      ),
      );  }
  }
```

OUTPUT:

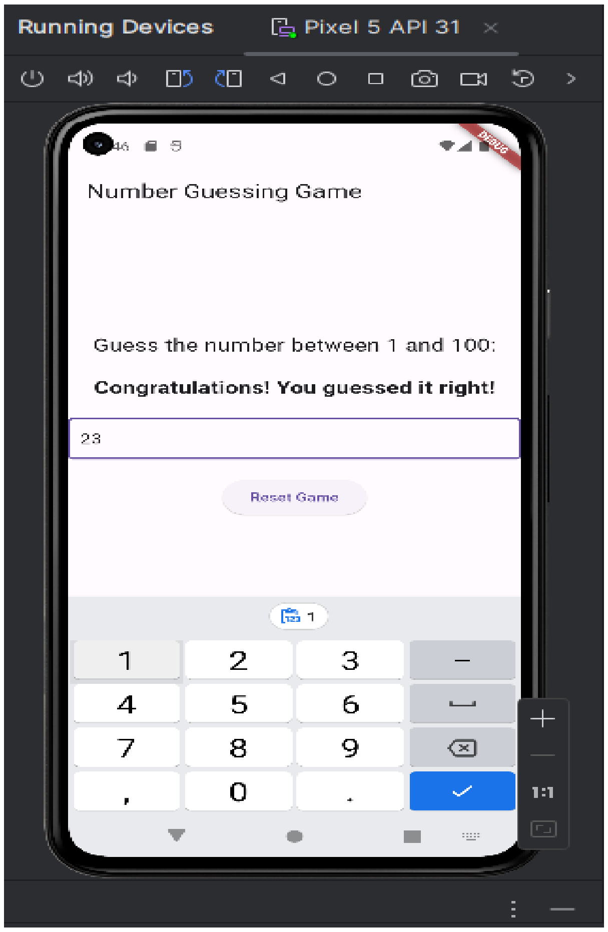


Exp.no :4(b) Number Game using Gestures.

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

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

class NumberGuessingGameextends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
      home: NumberGuessingScreen(),
    );
  }
}

class NumberGuessingScreenextends StatefulWidget {
@override
_NumberGuessingScreenStatecreateState() =>_NumberGuessingScreenState();
}

class _NumberGuessingScreenStateextends State<NumberGuessingScreen> {
final _random = Random();
  int _targetNumber= 0;
  int _guess = 0;
  String _result = '';

@override
void initState() {
super.initState();
    _initializeGame();
  }

void _initializeGame() {
_targetNumber= _random.nextInt(100) + 1;
_result = '';
_guess = 0;
  }

void _makeGuess(int guess) {
setState(() {
_guess = guess;
if (_guess == _targetNumber) {
_result = 'Congratulations! You guessed it right!';
```

```
      } else if (_guess <_targetNumber) {
_result = 'Try a higher number.';
      } else {
_result = 'Try a lower number.';
      }
    });
  }

void _resetGame() {
setState(() {
      _initializeGame();
    });
  }

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
        title: Text('Number Guessing Game'),
      ),
      body: Center(
        child: Column(
mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
Text(
'Guess the number between 1 and 100:',
              style: TextStyle(fontSize: 20),
            ),
SizedBox(height: 20),
Text(
_result,
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
SizedBox(height: 20),
TextField(
keyboardType: TextInputType.number,
onChanged: (value) {
                _makeGuess(int.tryParse(value) ?? 0);
              },
              decoration: InputDecoration(
hintText: 'Enter your guess',
contentPadding: EdgeInsets.symmetric(horizontal: 10),
                border: OutlineInputBorder(),
              ),
            ),
SizedBox(height: 20),
ElevatedButton(
onPressed: _resetGame,
              child: Text('Reset Game'),
            ),
          ],
        ),      ),    );
  }}
```

OUTPUT:



Number Guessing Game

Guess the number between 1 and 100:

Congratulations! You guessed it right!

23

RESULT:

Thus the above Dart Program for developing 2D gaming application can be implemented successfully and verified

EXP.NO 5 Develop a movie rating application (similar to IMDB)

Aim:

To develop a flutter based application that creates movie rating application (similar to IMDB) using Dart language.

Procedure:

Initialize Flutter Project:

- Open Android Studio and create a new Flutter project.
- Set up the project with appropriate configurations.

Design User Interface (UI):

- Design the UI for the movie rating application.
- Include features such as a list of movies, movie details page, search functionality, rating system, etc.
- Utilize Flutter widgets like Scaffold, AppBar, ListView, Card, TextFormField, etc., to create the UI components.

Set Up Data Model:

- Define a data model to represent movie information, including attributes like title, description, release year, rating, etc.
- Create Dart classes to represent movie objects.

Fetch Movie Data:

- Implement functionality to fetch movie data from an API or a local database.
- Utilize packages like http or dio to make network requests if fetching data from an API.
- Parse the response and populate movie objects with fetched data.
- Display Movie List:
- Display the list of movies fetched from the data source.
- Use Flutter widgets like ListView.builder to dynamically display movie items in a scrollable list.
- Each movie item should display essential information such as title, release year, and average rating.

Implement Movie Details Page:

- Create a separate page to display detailed information about a selected movie.
- Navigate to this page when a user taps on a movie item in the movie list.
- Display additional details such as description, cast, genre, etc., on the movie details page.

Implement Search Functionality:

- Implement search functionality to allow users to search for specific movies.
- Use Flutter's built-in search widgets or custom search implementation to filter movie list based on user input.

Add Rating System:

- Implement a rating system to allow users to rate movies.
- Utilize Flutter widgets like StarRating or custom rating widgets to display and capture user ratings.
- Store user ratings locally or remotely to update movie ratings.

Integrate User Authentication (Optional):

- Implement user authentication to allow users to log in and save their preferences, such as favorite movies or rated movies.
- Utilize packages like Firebase Authentication for user authentication.

Implement Offline Support (Optional):

- Implement offline support to allow users to access movie data even when offline.
- Utilize local database solutions like sqflite to cache movie data locally for offline access.

Optimize Performance:

- Optimize the performance of the application by minimizing unnecessary re-renders and optimizing network requests.
- Implement pagination or lazy loading to efficiently load large amounts of movie data.

Testing and Debugging:

- Test the application on various devices and screen sizes to ensure compatibility and responsiveness.
- Debug any issues or errors and ensure the application functions as expected.

Documentation and Deployment:

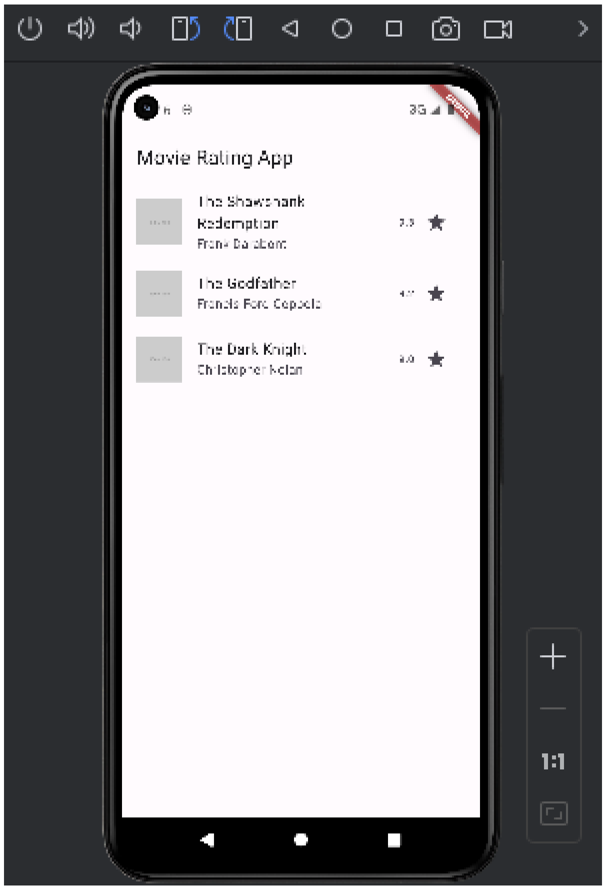
- Provide comprehensive documentation for developers and users, including instructions on how to use the application and information about features and functionalities.

Coding:

Main.Dart:

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

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

class Movie {
final String title;
final String director;
final String imageUrl;
  double rating;

  Movie({
required this.title,
required this.director,
required this.imageUrl,
this.rating= 0.0,
  });
}

class MovieRatingAppextends StatelessWidget {
```

```
final List<Movie>movies = [
Movie(
      title: 'The Shawshank Redemption',
      director: 'Frank Darabont',
imageUrl: 'https://via.placeholder.com/150',
      rating: 9.3,
    ),
Movie(
      title: 'The Godfather',
      director: 'Francis Ford Coppola',
imageUrl: 'https://via.placeholder.com/150',
      rating: 9.2,
    ),
Movie(
      title: 'The Dark Knight',
      director: 'Christopher Nolan',
imageUrl: 'https://via.placeholder.com/150',
      rating: 9.0,
    ),
// Add more movies as needed
];

@override
Widget build(BuildContext context) {
return MaterialApp(
      title: 'Movie Rating App',
      home: Scaffold(
appBar: AppBar(
          title: Text('Movie Rating App'),
        ),
        body: ListView.builder(
itemCount: movies.length,
itemBuilder: (context, index) {
final movie = movies[index];
return ListTile(
              leading: Image.network(
movie.imageUrl,
                width: 50,
                height: 50,
                fit: BoxFit.cover,
              ),
              title: Text(movie.title),
              subtitle: Text(movie.director),
              trailing: Row(
mainAxisSize: MainAxisSize.min,
                children: <Widget>[
Text(movie.rating.toString()),
IconButton(
                  icon: Icon(Icons.star),
onPressed: () {
                    _rateMovie(context, movie);
                  },
                ),              ],
              ),            );
          },        ),
        ),
      );  }

void _rateMovie(BuildContext context, Movie movie) {
showDialog(
      context: context,
      builder: (BuildContext context) {
return AlertDialog(
          title: Text('Rate ${movie.title}'),
          content: SingleChildScrollView(
            child: Column(
              children: <Widget>[
Text('Rate this movie:'),
Row(
mainAxisAlignment: MainAxisAlignment.center,
                  children: <Widget>[
IconButton(
iconSize: 40,
                      icon: Icon(Icons.star_border),
onPressed: () {
                        _setRating(context, movie, 1);
                      },
                    ),
IconButton(
iconSize: 40,
                      icon: Icon(Icons.star_border),
onPressed: () {
                        _setRating(context, movie, 2);
                      },
                    ),
IconButton(
iconSize: 40,
                      icon: Icon(Icons.star_border),
onPressed: () {
                        _setRating(context, movie, 3);
                      },
                    ),
IconButton(
iconSize: 40,
                    icon: Icon(Icons.star_border),
onPressed: () {
                      _setRating(context, movie, 4);
                    },
                  ),
IconButton(
iconSize: 40,
                    icon: Icon(Icons.star_border),
onPressed: () {
                      _setRating(context, movie, 5);
                    },
                  ),
                ],                ),
            ],            ),
         ),        );      },
    );  }

void _setRating(BuildContext context, Movie movie, double rating) {
Navigator.of(context).pop();
movie.rating= rating;
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
        content: Text('You rated ${movie.title} as $rating stars.'),
      ),
    );
  }
}
```



Movie Rating App

The Shawshank Redemption
Frank Darabont
[illegible] ★

The Godfather
Francis Ford Coppola
[illegible] ★

The Dark Knight
Christopher Nolan
[illegible] ★

Result:

Thus the above Dart Program for Development of movie rating application (similar to IMDB) in flutter has been successfully implemented and verified

Ex.No:6 Develop an application to connect to a web service and to retrieve data with HTTP

Aim:

Developing an application to connect to a web service and retrieve data using HTTP is a common task, especially in mobile and web development. Here's a step-by-step guide to achieve this using Dart and Flutter

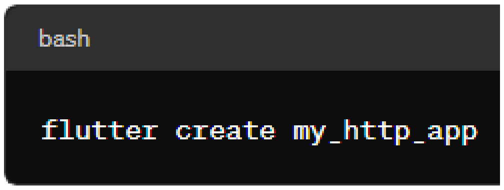
Procedure:

Set Up Flutter Environment:

- Ensure you have Flutter installed and configured on your system. You can follow the instructions on the Flutter website: Flutter Installation Guide.

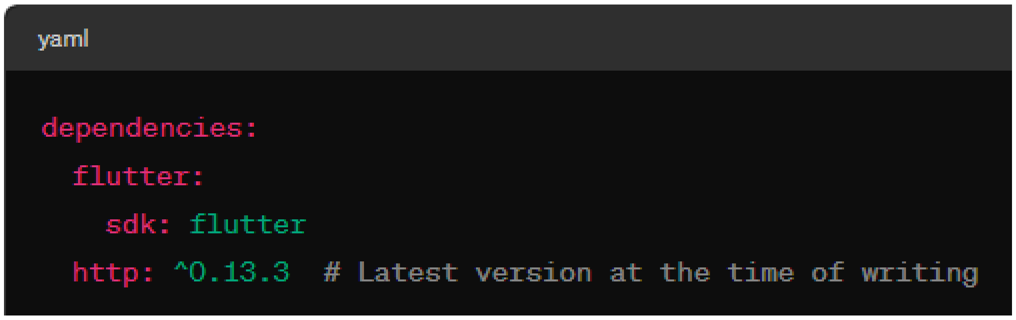
Create a New Flutter Project:

- Use the Flutter CLI or your IDE to create a new Flutter project.



```bash
flutter create my_http_app
```

- Add HTTP Package Dependency: Flutter provides the http package for making HTTP requests. Add it as a dependency in your pubspec.yaml file:



```yaml
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3  # Latest version at the time of writing
```

Then, run flutter pub get to fetch the dependencies.

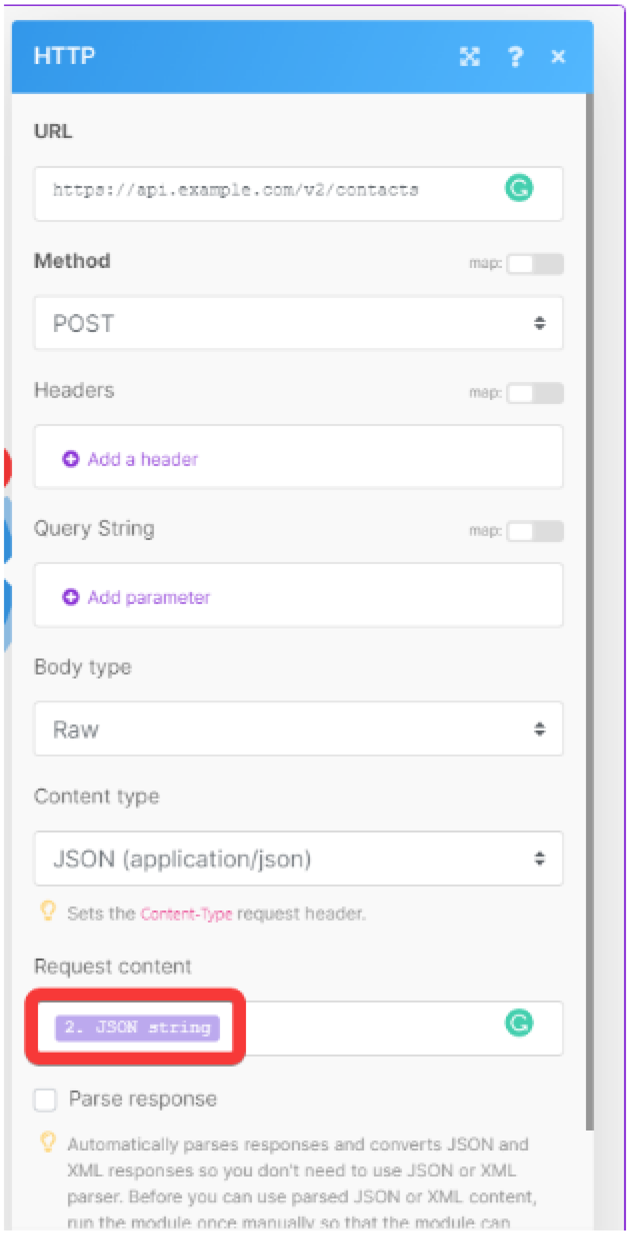
Implement HTTP Request:

- Import the http package in your Dart file.
- Use the get() function from the http package to make a GET request to the web service.

Coding:

```
import 'dart:convert';
import 'package:http/http.dart' as http;
void main() {
  fetchData();
}
void fetchData() async {
  try {
    // Replace 'https://jsonplaceholder.typicode.com/posts/1' with the actual URL of the web service
    var apiUrl = 'https://jsonplaceholder.typicode.com/posts/1';
    var response = await
http.get(Uri.parse('https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwjCvpL1xrmEAxWdgksFH
afiAAgYABAAGgJzZg&ase=2&gclid=EAIaIQobChMIwr6S9ca5hAMVnYJLBR2n4gAIEAAYASAAEgIoH_D_BwE&ohost
=www.google.com&cid=CAASJuRo_NCsnnfMwbV6XHFEjcrMA1CqnQ38oytOmWxHzstazYVq8pj0&sig=AOD64_2
-
KjBRcJAoNh2GCzH7WqHZMktJrg&q&nis=4&adurl&ved=2ahUKEwj7rIz1xrmEAxW5TmwGHWwcDykQ0Qx6BAgP
EAE'));
    // Check if the request was successful (status code 200)
    if (response.statusCode == 200) {
      // If the server returns a 200 OK response, parse the JSON
      var jsonData = jsonDecode(response.body);
      // Process the jsonData further
    } else {
      // If the server returns an error response, handle it accordingly
      print('Request failed with status: ${response.statusCode}');
    }
  } catch (e) {
    print("An error occurred: $e");
  }
}
```

OUTPUT:



HTTP

URL

https://api.example.com/v2/contacts

Method: POST

Headers

Add a header

Query String

Add parameter

Body type: Raw

Content type: JSON (application/json)

Sets the Content-Type request header.

Request content: 2. JSON string

Parse response

Automatically parses responses and converts JSON and XML responses so you don't need to use JSON or XML parser. Before you can use parsed JSON or XML content, run the module once manually so that the module can

RESULT:

Thus the above Dart program to develop an application to connect to a web service and to retrieve data with HTTP

Exp.No:7 Develop a simple shopping application

Aim:

Developing a simple shopping application involves several steps, from designing the user interface to implementing functionality for browsing products, adding items to a cart, and checking out. Here's an outline to

Procedure:

Design the User Interface (UI):

- Sketch out the UI layout including screens for browsing products, viewing product details, cart, and checkout.
- Decide on the navigation structure and user flow within the app.

Set Up Flutter Project:

- Create a new Flutter project using the Flutter CLI or IDE.

Create Screens:

- Implement individual screens for browsing products, viewing product details, cart, and checkout.
- Use Flutter widgets like ListView, GridView, Card, AppBar, BottomNavigationBar, etc., to build the UI.

Manage State:

- Determine how to manage state within the app. You can use Flutter's built-in state management solutions like setState, Provider, Bloc, or GetX.

Implement Backend (Optional):

- If you want to simulate a real shopping experience, you can create a simple backend using Firebase, Node.js, or any other technology.
- Set up endpoints for fetching products, managing carts, and processing orders.

Fetch Product Data:

- Fetch product data from a local JSON file or a remote server.
- Use HTTP requests (GET) to retrieve product information from an API.

Display Products:

- Display the list of products on the product browsing screen.
- Show product images, names, prices, and any other relevant information.

Product Details:

- Implement a screen to display detailed information about a selected product.
- Include options for selecting product variants (e.g., size, color).

Add to Cart:

- Implement functionality to add products to the shopping cart.
- Store cart items locally or use state management to keep track of the cart state.

View Cart:

- Create a screen to display the contents of the shopping cart.

- Show the list of items with quantities and prices.

Checkout:

- Implement the checkout process, allowing users to review their cart and proceed to payment.
- You can simulate the payment process or integrate a payment gateway if desired.

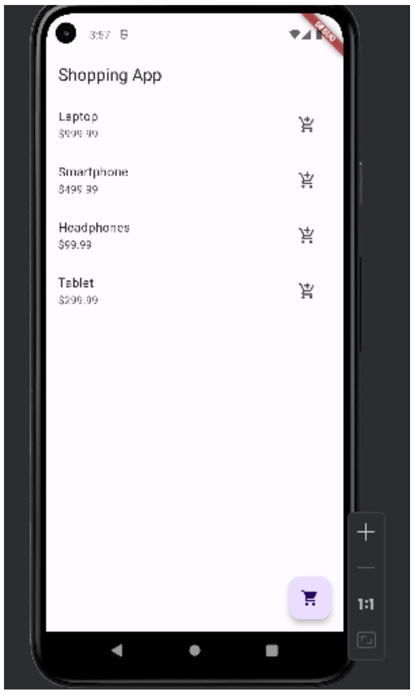
Handle User Authentication (Optional):

- Implement user authentication to allow users to sign in, view their order history, and manage their account.

Coding:

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

void main() {
  runApp(MyApp());
}
class Product {
  final String name;
  final double price;
  Product(this.name, this.price);
}
class ShoppingCart {
  List<Product> _items = [];
  void addItem(Product item) {
    _items.add(item);
  }
  void displayItems(BuildContext context) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text("Items in your shopping cart"),
          content: _items.isEmpty
              ? Text("Your shopping cart is empty.")
              : Column(
                  mainAxisSize: MainAxisSize.min,
                  children: _items.map((item) {
                    return ListTile(
                      title: Text(item.name),
                    subtitle: Text("\$${item.price}"),
                  );                  }).toList(),
                ),
            actions: <Widget>[
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                },
                child: Text("Close"),
              ),          ],        );      },
        );  }    }
class MyApp extends StatelessWidget {
  final List<Product> products = [
    Product("Laptop", 999.99),
    Product("Smartphone", 499.99),
    Product("Headphones", 99.99),
    Product("Tablet", 299.99),
  ];
  final ShoppingCart cart = ShoppingCart();
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Shopping App',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Shopping App'),
        ),
        body: ListView.builder(
          itemCount: products.length,
          itemBuilder: (BuildContext context, int index) {
            return ListTile(
              title: Text(products[index].name),
              subtitle: Text("\$${products[index].price}"),
              trailing: IconButton(
                icon: Icon(Icons.add_shopping_cart),
                onPressed: () {
                  cart.addItem(products[index]);
                  ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(content: Text('Added to cart')),
                  );                },
              ),            );
            },        ),
          floatingActionButton: FloatingActionButton(
            onPressed: () {
              cart.displayItems(context);
            },
            tooltip: 'View Cart',
            child: Icon(Icons.shopping_cart),
          ),      ),    );  }
```

OUTPUT:



```
3:57

Shopping App

Laptop
$999.99

Smartphone
$499.99

Headphones
$99.99

Tablet
$299.99
```

RESULT:

Thus the above Dart program to develop an simple shopping application has been created successfully

Exp.NO:8 Design a web server supporting push notifications

Aim:

Designing a web server supporting push notifications in Dart involves creating a server that can establish and maintain WebSocket connections with clients and deliver notifications over these connections.

Procedure:

Choose the Technology Stack:

- Use the Dart programming language with the shelf package for building HTTP servers.
- Use the web_socket_channel package for handling Web Socket connections.

Set Up a Dart Project:

- Create a new Dart project using your preferred IDE or the Dart SDK.
- Add dependencies for shelf and web_socket_channel in your pubspec.yaml file.

Implement User Authentication and Authorization (Optional):

- Design and implement authentication mechanisms if needed to ensure that only authenticated users can receive push notifications.
- Use tokens, sessions, or OAuth for user authentication and authorization.

Set Up WebSocket Support:

- Create an HTTP server using shelf.
- Use the shelf_web_socket package to upgrade HTTP connections to WebSocket connections.
- Define a WebSocket endpoint to handle WebSocket connections from clients.

Handle WebSocket Connections:

- Implement logic to handle incoming WebSocket connections from clients.
- Maintain a list of connected clients and manage their subscriptions to notification channels.

Manage Subscriptions:

- Design a mechanism for users to subscribe to specific notification channels or topics.
- Store user subscriptions in memory or a database.

Implement Push Notification Logic:

- Design and implement server-side logic to trigger push notifications based on events or criteria.
- When an event occurs that triggers a notification, send the notification to all subscribed clients over WebSocket connections.

Handle Error and Retry Mechanisms:

- Implement error handling and retry mechanisms to ensure the reliability of push notifications.
- Handle network errors, service outages, and transient failures gracefully.

Implement Monitoring and Analytics (Optional):

- Set up logging, monitoring, and analytics to track the performance and usage of your push notification system.
- Use tools like Prometheus, Grafana, ELK Stack, or New Relic for monitoring and analytics.

Secure Your System:

- Secure your web server and WebSocket connections using TLS/SSL encryption to protect data in transit.
- Implement security best practices such as rate limiting, input validation, and firewall rules to prevent attacks.

Test Your System:

- Test your push notification system thoroughly to ensure reliability, scalability, and security.
- Perform unit tests, integration tests, and end-to-end tests to validate the functionality of your system.

Deploy Your Web Server:

- Deploy your Dart web server to a reliable hosting environment, such as cloud platforms like AWS, Google Cloud Platform, or Azure.
- Configure auto-scaling, load balancing, and monitoring to handle varying traffic loads.

Develop Client Applications:

- Develop client applications (e.g., web, mobile) that can receive and handle push notifications.
- Implement client-side logic to subscribe to push notifications and handle incoming messages.

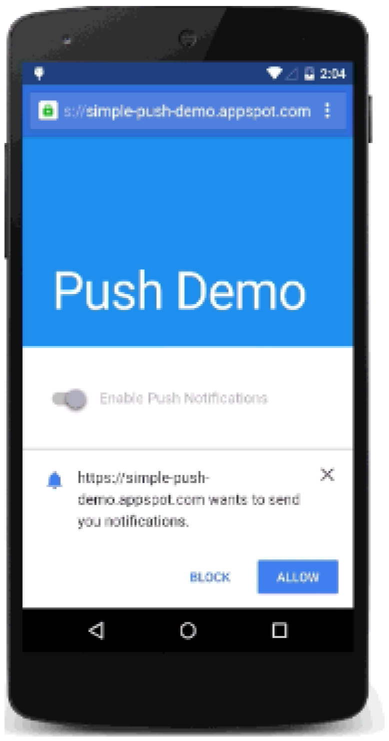
Compliance and Privacy:

- Ensure compliance with privacy regulations (e.g., GDPR) when collecting and processing user data for push notifications.
- Provide users with control over their notification preferences and opt-out mechanisms.

Coding:

```
import 'dart:convert';

import 'dart:io';

import 'package:shelf/shelf.dart' as shelf;

import 'package:shelf/shelf_io.dart' as io;

import 'package:firebase_messaging/firebase_messaging.dart';

// Firebase Cloud Messaging (FCM) server key

const String serverKey = 'your_server_key_here';

// Initialize Firebase Messaging

final FirebaseMessaging messaging = FirebaseMessaging();

void main() async {

  // Create a Shelf handler

  var handler = const shelf.Pipeline()

      .addMiddleware(shelf.logRequests())
      .addHandler(_echoRequest);

  // Start server
  var server = await io.serve(handler, 'localhost', 8080);
  print('Server running on localhost:${server.port}');
}
// Function to handle requests
shelf.Response _echoRequest(shelf.Request request) {
  if (request.url.path == '/sendNotification') {
    // Extract title and body from the request
    final queryParams = request.url.queryParameters;
    final title = queryParams['title'];
    final body = queryParams['body'];

    // Send notification to all subscribed devices
    sendNotification(title, body);
        return shelf.Response.ok('Notification sent');
  } else {
    return shelf.Response.notFound('Not Found');
  }
}
// Function to send push notification
void sendNotification(String title, String body) {
  messaging.sendToTopic('your_topic_name_here', {
    'notification': {
      'title': title,
      'body': body,
    },
  });
}
```

OUTPUT:


2:04
s://simple-push-demo.appspot.com
Push Demo
Enable Push Notifications
https://simple-push-demo.appspot.com wants to send you notifications.
BLOCK
ALLOW

RESULT:

Thus the above dart program for designing a web server supporting push notifications has been developed successfully.

Exp.NO:9 Develop an application by integrating Google maps

Aim:

To develop a Dart application that integrates Google Maps. This application could be a location-based service, navigation tool, or any other type of app that leverages mapping functionality.

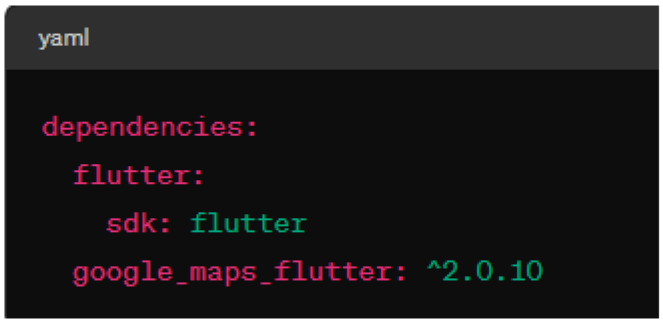
Procedure:

Set Up a New Flutter Project:

- Create a new Flutter project using the Flutter CLI or your preferred IDE.
- Make sure your development environment is set up correctly for Flutter development.

Add Google Maps Dependency:

- Add the google_maps_flutter dependency to your pubspec.yaml file:



yaml

```yaml
dependencies:
  flutter:
    sdk: flutter
  google_maps_flutter: ^2.0.10
```

Run flutter pub get to install the dependency.

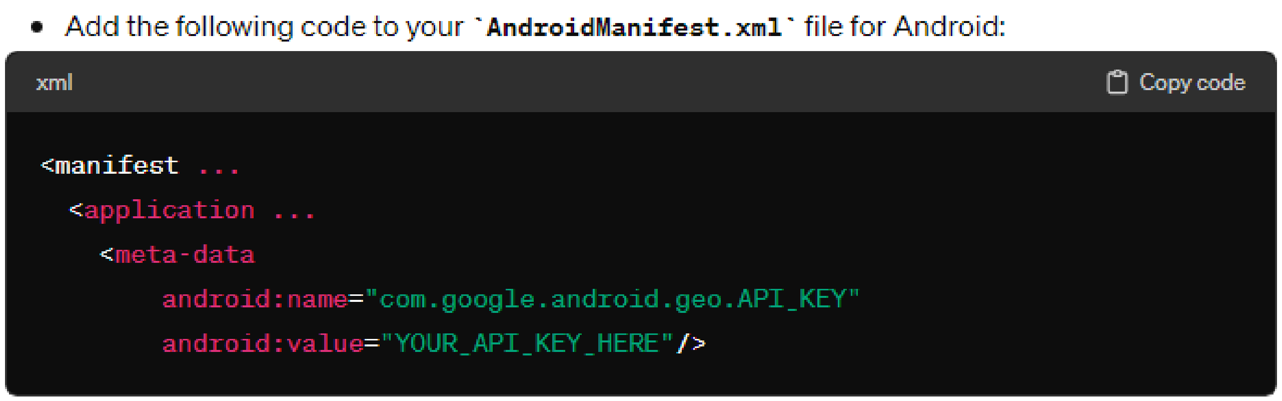
Get Google Maps API Key:

- Obtain an API key from the Google Cloud Console by following the instructions provided in the official documentation: Get API Key.

Initialize Google Maps in Your App:

- Initialize Google Maps in your app by providing the API key and enabling the necessary permissions.

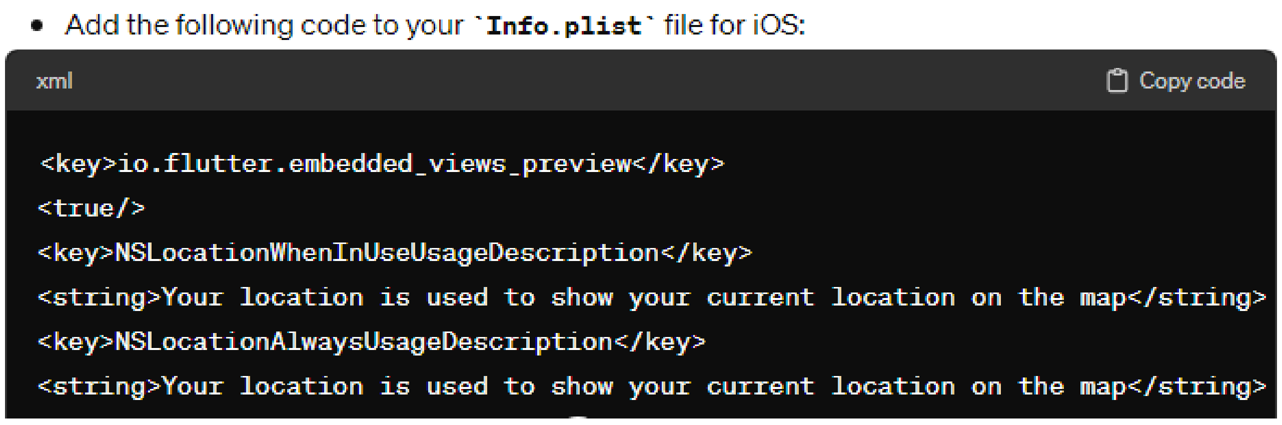
Add the following code to your AndroidManifest.xml file for Android:



- Add the following code to your `AndroidManifest.xml` file for Android:

xml Copy code

```xml
<manifest ...
  <application ...
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="YOUR_API_KEY_HERE"/>
```



- Add the following code to your `Info.plist` file for iOS:

xml Copy code

```xml
<key>io.flutter.embedded_views_preview</key>
<true/>
<key>NSLocationWhenInUseUsageDescription</key>
<string>Your location is used to show your current location on the map</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>Your location is used to show your current location on the map</string>
```

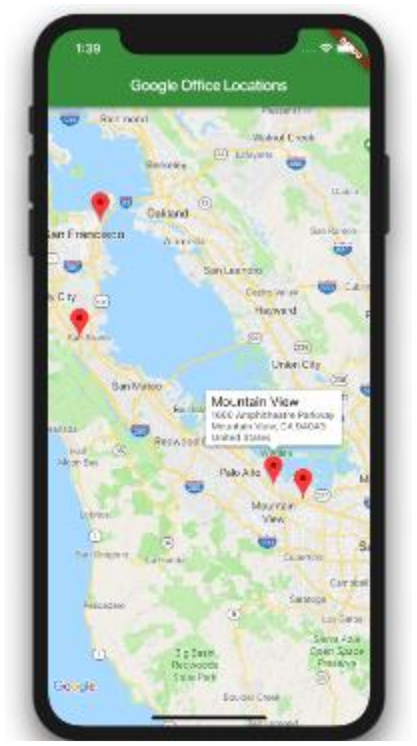
Implement Google Maps Widget:

- Create a new Dart file (e.g., maps_screen.dart) where you'll define the UI for displaying Google Maps.
- Use the GoogleMap widget from the google_maps_flutter package to display the map.

Coding:

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

import 'package:google_maps_flutter/google_maps_flutter.dart';

class MapsScreen extends StatefulWidget {

  @override

  _MapsScreenState createState() => _MapsScreenState();

}

class _MapsScreenState extends State<MapsScreen> {

  GoogleMapController? _controller;

  @override

  Widget build(BuildContext context) {

    return Scaffold(
      appBar: AppBar(
        title: Text('Google Maps'),
      ),
      body: GoogleMap(
        onMapCreated: (controller) {
          setState(() {
            _controller = controller;
          });       },
        initialCameraPosition: CameraPosition(
          target: LatLng(37.7749, -122.4194), // San Francisco coordinates
          zoom: 12.0,
        ),     ),
    );  }  }
```

OUTPUT:



RESULT:

Thus the above dart program for developing an application by integrating Google maps has been created successfully.

Exp.No:10 Mini Projects involving Flutter/Kotlin multi-platform (Implementation of a chat application using Dart)

Aim:

Creating a simple text-based chat interface that allows users to send and receive messages in real-time.

Procedure:

Set Up Server:

- Create a Dart server using the dart:io library to handle incoming connections from clients.
- Set up a TCP server socket to listen for incoming connections.
- Handle each client connection by creating a separate Socket object.

Client Connection:

- Create a Dart client application that connects to the server using a TCP socket.
- Handle user input to send messages to the server.
- Receive and display messages from other clients.

Implement Chat Protocol:

- Define a simple chat protocol for communication between the client and server.
- For example, each message could be sent as a string with a predefined format (e.g., <username>: <message>).

Handle Multiple Clients:

- Modify the server code to handle multiple client connections concurrently.
- Use asynchronous programming techniques (e.g., async, await, Future) to handle multiple client connections efficiently.

Error Handling and Graceful Shutdown:

- Implement error handling to handle unexpected situations gracefully (e.g., client disconnects, server errors).
- Ensure that the server can be stopped gracefully without losing any data or connections.

Testing and Debugging:

- Test the chat application with multiple clients to ensure that messages are delivered correctly and in real-time.
- Debug any issues related to network communication, message formatting, or concurrency.

Enhancements:

- Once the basic chat functionality is working, consider adding additional features such as user authentication, message encryption, or support for multimedia messages (e.g., images, files).

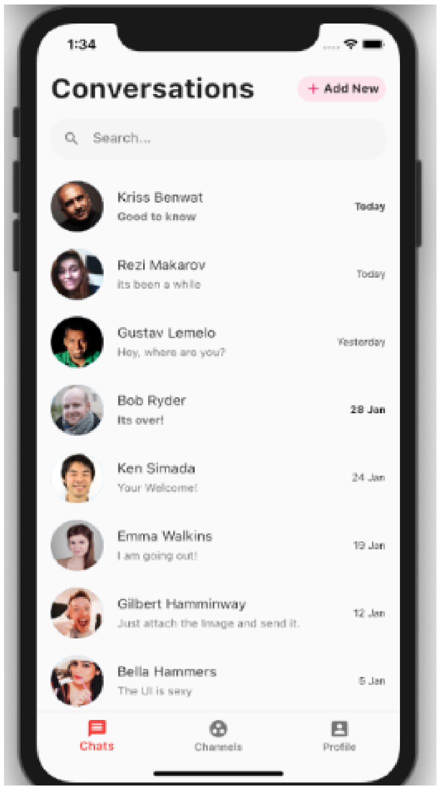
Coding: Server side

```
import 'dart:io';
void main() async {
  final server = await ServerSocket.bind('127.0.0.1', 4041);
  print('Server running on ${server.address}:${server.port}');
  await for (var socket in server) {
    handleClient(socket);
  }  }
void handleClient(Socket socket) async {
  print('Client connected: ${socket.remoteAddress}:${socket.remotePort}');
  socket.listen(
    (data) {
      print('Received: ${String.fromCharCodes(data)}');
      // Broadcast message to all clients
      // Modify this part to implement the chat protocol
      for (var client in server) {
        client.write(data);
      }
    },
    onDone: () {
      print('Client disconnected: ${socket.remoteAddress}:${socket.remotePort}');
      socket.close();
    },
    onError: (error) {
      print('Error: $error');
      socket.close();
    },
  );
}
```

Client side

```
import 'dart:io';
void main() async {
  final socket = await Socket.connect('127.0.0.1', 4041);
  print('Connected to server');
  // Handle user input and send messages to the server
  stdin.listen((data) {
    socket.write(String.fromCharCodes(data).trim());
  });
  // Receive messages from the server and display them
  socket.listen((data) {
    print('Received: ${String.fromCharCodes(data)}');
  }, onError: (error) {
    print('Error: $error');
    socket.close();
  }, onDone: () {
    print('Disconnected from server');
    socket.close();
  }); }
```

OUTPUT:



1:34

Conversations

+ Add New

Search...

Kriss Benwat
Good to know
Today

Rezi Makarov
its been a while
Today

Gustav Lemelo
Hey, where are you?
Yesterday

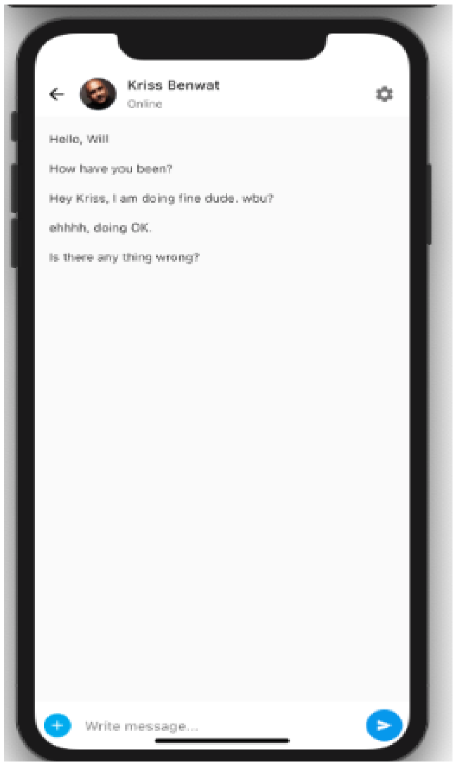
Bob Ryder
Its over!
28 Jan

Ken Simada
Your Welcome!
24 Jan

Emma Walkins
I am going out!
19 Jan

Gilbert Hamminway
Just attach the image and send it.
12 Jan

Bella Hammers
The UI is sexy
5 Jan



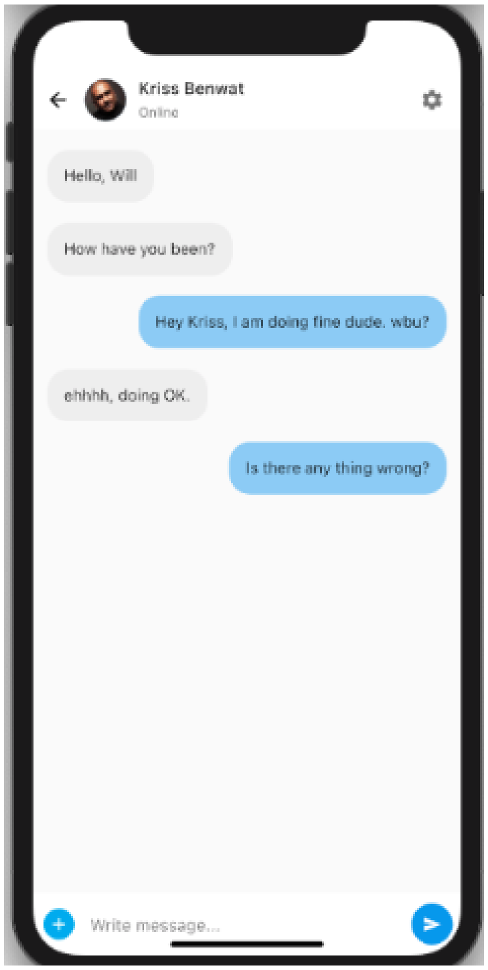
Hello, Will

How have you been?

Hey Kriss, I am doing fine dude. wbu?

ehhhh, doing OK.

Is there any thing wrong?



Hello, Will

How have you been?

Hey Kriss, I am doing fine dude. wbu?

ehhhh, doing OK.

Is there any thing wrong?

RESULT:

Thus the above dart program for mini project-Chat application has been created successfully