प्रमाणपत्र सं. / Certificate No.:
LD-20250176794

सत्यमेव जयते

प्रतिलिप्याधिकार कार्यालय, भारत सरकार | Copyright Office, Government Of India

प्रतिलिप्यधिकार प्रमाणपत्र | Copyright Certificate

प्रतिलिप्यधिकार नियम का नियम 70 | Rule 70 of The Copyright Rules

प्रतिलिप्यधिकार रजिस्टर से उद्धरण | Extracts from the Register of Copyrights

आवेदन सं. / Application No.: LD-26153/2025-CO **दाखिल करने की तिथि/ Date of Filing:** 30/06/2025

एतद्वारा यह प्रमाणित किया जाता है कि प्रतिलिप्यधिकार अधिनियम, 1957 के प्रावधानों के अनुसार, उपर्युक्त आवेदन में प्रकट किए गए "IT3681 - Mobile Application Development Laboratory Manual" नामक कार्य के लिए प्रतिलिप्यधिकार प्रदान किया गया है।

This is to certify that a copyright has been registered for the work titled "IT3681 - Mobile Application Development Laboratory Manual" as disclosed in the below mentioned application in accordance with the provisions of the Copyright Act,1957.

रचयिता / Authors:

सं. / SNO.	नाम / Name	पता / Address
1	DR.V.ANUSUYA	HEAD OF INFORMATION TECHNOLOGY, RAMCO INSTITUTE OF TECHNOLOGY, RAJAPALAYAM-626117
2	MRS.M.RETHINAKUMARI	ASSISTANT PROFESSOR, DEPARTMENT OF INFORMATION TECHNOLOGY, RAJAPALAYAM-626117
3	MRS.G.SIVASATHIYA	ASSISTANT PROFESSOR, DEPARTMENT OF INFORMATION TECHNOLOGY, RAMCO INSTITUTE OF TECHNOLOGY, RAJAPALAYAM-626117

स्वामी / Owners:

सं. / SNO.	नाम / Name	पता / Address
1	RAMCO INSTITUTE OF TECHNOLOGY	NORTH VENGANALLUR VILLAGE, RAJAPALAYAM-626117

प्रकाशक विवरण / Publisher Details: N.A.

कार्य की भाषा / Language of the Work: English

कार्य का विवरण / Description of the Work: The work is laboratory manual of Mobile Application Development laboratory course describing the working procedure of various exercises

(शर्तें / टिप्पणियाँ, यदि कोई हो) / (Conditions/ Remarks, if any): N.A.

उमा शंकर पंडित

Registrar of Copyrights

आर. ओ. सी. जारी होने की तिथि / Date of ROC: 07/11/2025

IT3681 MOBILE APPLICATION DEVELOPMENT

LABORATORY MANUAL

MASTER RECORD

B.Tech. INFORMATION TECHNOLOGY

III Year / VI Semester

Regulation-2021

DEPARTMENT OF INFORMATION TECHNOLOGY

RAMCO INSTITUTE OF TECHNOLOGY

RAJAPALAYAM-626117.

TAMILNADU.

Prepared by

Mrs. M.RETHINA KUMARI, AP/IT

Approved by

Dr. V. ANUSUYA, HOD/IT

VISION AND MISSION

Vision of the Institute

To evolve as an Institute of international repute in offering high-quality technical education, Research and extension programmes in order to create knowledgeable, professionally competent and skilled Engineers and Technologists capable of working in multi-disciplinary environment to cater to the societal needs.

Mission of the Institute

- To accomplish its unique vision, the Institute has a far-reaching mission that aims:
- To offer higher education in Engineering and Technology with highest level of quality, Professionalism and ethical standards
- To equip the students with up-to-date knowledge in cutting-edge technologies, wisdom, creativity and passion for innovation, and life-long learning skills
- To constantly motivate and involve the students and faculty members in the education process for continuously improving their performance to achieve excellence.

Vision of the Department

To be developed global standards in high-quality technical education, Research and Development in Information Technology by adapting the rapid technological advancement.

Mission of the Department

- Making quality IT engineers with good technical proficiency by effective teaching to adapt the dynamic needs of industries.
- To take up creative research in collaboration with leading Industries and professional societies to make the nation as a knowledge-power.
- Developing the employability and entrepreneurial skills of the students by conducting industry-collaborated training with the strong ethical values and professionalism to serve society with responsibility.

Program Educational Objectives (PEOs)

- PEO 1: Graduates will be proficient in leading IT industries, pursue higher study and research, and progress as entrepreneurs by utilizing the fundamental knowledge of various streams in engineering and technology.
- PEO 2: Graduates will think logically and pursue lifelong learning skills to understand technical issues related to emerging computing systems and to provide prime or best solutions.
- PEO 3: Graduates will be able to develop system applications by understanding the importance of socially committed engineers with high ethical values, business and environmental needs in the social context.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

- **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **Problem analysis:** Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, [illegible]ces, and engineering sciences.
- **[illegible]opment of solutions:** Design solutions for complex engineering problems and

design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

- **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

After successful completion of the degree, the students will be able to:

- PSO1: Apply mathematical and computing skills including Data Science and algorithms to identify and provide solutions for complex computational problems.
- PSO2: Develop the skills in emerging technologies such as Embedded Systems and IoT, Mobile Application Development and Security, and data analytics aids in anticipating research findings and providing the reliable IT solutions.
- PSO3: Apply standard Engineering practices and strategies in software projects using open source environments to deliver a quality product for business and industry attainment.

INSTRUCTIONS TO STUDENTS

- Students should wear Uniforms and Coats neatly during the lab session
- Students should maintain silence during lab hours; roaming around the lab during lab session is not permitted
- Programs should be written in the manual and well prepared for the current exercise before coming to the session
- Experiments should be completed within the Specified Lab Session
- Before Every Session, Last Session lab exercise & record should be completed and get it verified by the faculty
- In the Record Note, Flow Chart and Outputs should be written on the left side, while Aim, Algorithm & Result should be written on the right side.
- Programs (Printed) should be placed on the right side
- Marks for each lab exercise is awarded as follows:

Performance	25 Marks
Viva	10 Marks
Record	15 Marks
Total	**50 Marks**

CONTENTS		Page No.
Syllabus		
Course Objectives and Outcomes		
CO – Mapping		
Ex. No.	**Name of the Lab Exercises**	
1.	Study and installation of Flutter/Kotlin multi-platform environment	
2.	Develop an application that uses Widgets, GUI components, Fonts, and Colors.	
3.	Develop a native calculator application	
4.	Develop a gaming application that uses 2-D animations and gestures.	
5.	Develop a movie rating application (similar to IMDB)	
6.	Develop an application to connect to a web service and to retrieve data with HTTP	
7.	Develop a simple shopping application.	
8.	Design a web server supporting push notifications	
9.	Develop an application by integrating Google maps	
10.	Mini Projects involving Flutter/Kotlin multi-platform	

IT3681 MOBILE APPLICATIONS DEVELOPMENT LABORATORY **L T P C**
0 0 3 1.5

COURSE OBJECTIVES:

The objective of this course is to enable the students to

- Use Flutter/Kotlin multi-platform environment for building cross-platform mobile applications.
- Demonstrate the knowledge of different programming techniques and patterns for mobile application development.
- Identify the components and structure of mobile application development frameworks.
- Understand the capabilities and limitations of different platforms.
- Design and develop real-time mobile applications.

LIST OF EXPERIMENTS:

1. Study and installation of Flutter/Kotlin multi-platform environment
2. Develop an application that uses Widgets, GUI components, Fonts, and Colours.
3. Develop a native calculator application.
4. Develop a gaming application that uses 2-D animations and gestures.
5. Develop a movie rating application (similar to IMDB)
6. Develop an application to connect to a web service and to retrieve data with HTTP.
7. Develop a simple shopping application.
8. Design a web server supporting push notifications.
9. Develop an application by integrating Google maps
10. Mini Projects involving Flutter/Kotlin multi-platform

COURSE OUTCOMES:

On successful completion of this course, the student should be able to

CO1: Design and build simple mobile applications supporting multiple platforms.
CO2: Apply various programming techniques and patterns to build mobile applications.
CO3: Build real-time mobile applications for society/environment
CO4: Build gaming and multimedia based mobile applications
CO5: Build AI based mobile applications for society/environment following ethical practices.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Simone Alessandria, Flutter Projects: A practical project-based guide to building real-world cross-platform mobile applications and games, Packt publishing.
2. 2. Carmine Zaccagnino, Programming Flutter: Native, Cross-Platform Apps the Easy Way (The Pragmatic Programmers), Packt publishing.

REFERENCES:

1. Gergely Orosz, Building Mobile Applications at Scale:39 Engineering Challenges
2. Souvik Biswas & Codemagic, Flutter Libraries we love
3. ED Freitas, Daniel Jebaraj, Flutter Succinctly
4. Antonio Leiva, Kotlin for Android Developers Learn Kotlin the easy way while developing an Android Applications

RAMCO INSTITUTE OF TECHNOLOGY

Department of Information Technology

Academic Year: 2024- 2025 (Even Semester)

Course Objectives and Outcomes

Degree & Branch : **B. TECH -IT**

Semester : **VI**

Course Code & Title : **IT3681 & Mobile Application Development Laboratory**

Course Objectives:

The objective of this course is to enable the students to

- Use Flutter/Kotlin multi-platform environment for building cross-platform mobile applications.
- Demonstrate the knowledge of different programming techniques and patterns for mobile application development.
- Identify the components and structure of mobile application development frameworks.
- Understand the capabilities and limitations of different platforms.
- Design and develop real-time mobile applications.

Course Outcomes:

On successful completion of this course, the student should be able to

CO1: Design and build simple mobile applications supporting multiple platforms.

CO2: Apply various programming techniques and patterns to build mobile applications.

CO3: Build real-time mobile applications for society/environment

CO4: Build gaming and multimedia based mobile applications

CO5: Build AI based mobile applications for society/environment following ethical practices

CO - MAPPING

S No	Name of the Lab Exercises	COs
1	Study and installation of Flutter/Kotlin multi-platform environment	CO1
2	Develop an application that uses Widgets, GUI components, Fonts, and Colors.	CO1
3	Develop a native calculator application	CO2
4	Develop a gaming application that uses 2-D animations and gestures.	CO4
5	Develop a movie rating application (similar to IMDB)	CO3
6	Develop an application to connect to a web service and to retrieve data with HTTP	CO3
7	Develop a simple shopping application.	CO5
8	Design a web server supporting push notifications	CO5
9	Develop an application by integrating Google maps	CO5
10	Mini Projects involving Flutter/Kotlin multi-platform	CO5

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

AIM:

Study and installation of Flutter/Kotlin multi-platform environment

INTRODUCTION:

Flutter is an open-source UI software development toolkit by Google used to build natively compiled applications for mobile, web, and desktop from a single codebase. It enables developers to create high-performance applications with a rich set of pre-designed widgets and tools. Flutter's reactive framework, powered by the Dart programming language, ensures smooth and dynamic UI rendering across multiple platforms.

With its growing popularity in the software development industry, learning Flutter is essential for engineering students aspiring to build modern applications. This guide provides a step-by-step approach to installing Flutter on a Windows system, ensuring students have the necessary setup to start their journey in cross-platform app development.

Step 1: System Requirements

Before installing Flutter, ensure your system meets the minimum requirements:

- **Operating System**: Windows 10 or later (64-bit)
- **Processor**: Intel or AMD processor with 64-bit support
- **RAM**: Minimum 8 GB (recommended 16 GB)
- **Disk Space**: 2.5 GB (excluding space for IDEs and tools)

Development Tools: Git for Windows, Android Studio/VS Code

Step 2: Install Git

Why is Git Needed?

Git is a version control system that helps developers manage code changes efficiently. Flutter relies on Git for downloading dependencies, updating the SDK, and handling source control.

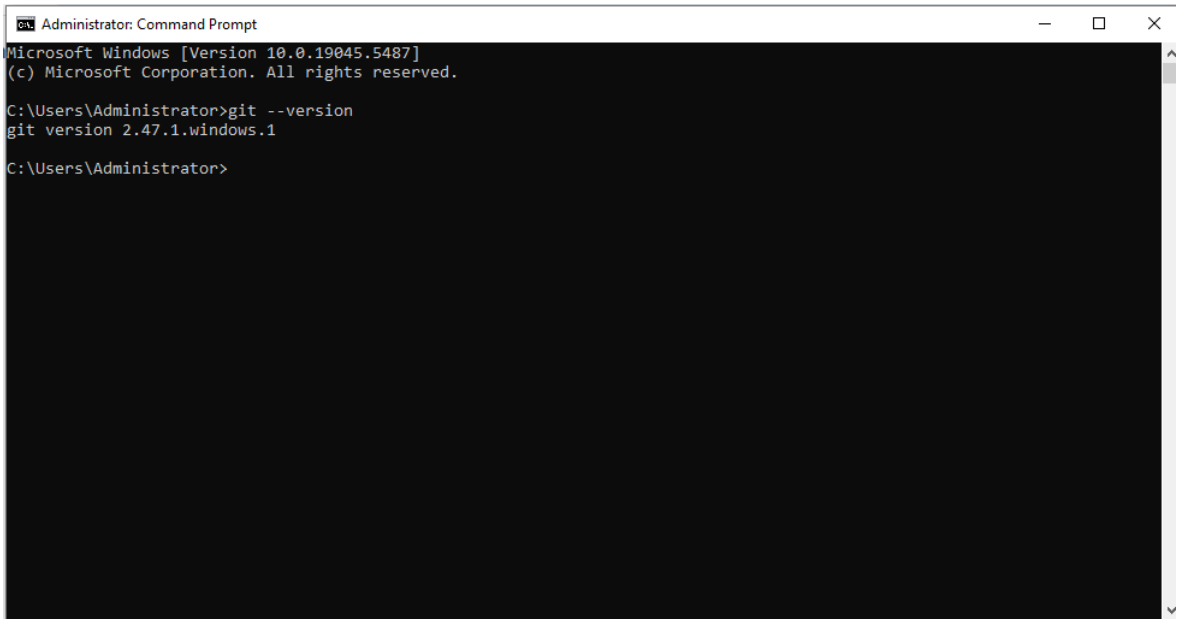
Steps to Install Git

1. Download Git for Windows from https://git-scm.com/downloads. (choose version 2.47.1)
2. Run the installer and follow these steps:
    - Click **Next** on the welcome screen.
    - Choose the default installation path and click **Next**.
    - Select **Git from the command line and also from 3rd-party software**.
    - Choose the default text editor (e.g., VS Code or Notepad++).
    - Use the recommended options for the remaining steps.
3. Click **Install** and wait for the installation to complete.
4. Add Git to System Path:
    - Open **System Environment Variables**:
        - Click **Advanced → Environment Variables**.
    - Under **System Variables**, find Path, select it, and click **Edit**.
    - Click **New**, paste C:\Program Files\Git\bin, and click **OK**.
5. Open **Command Prompt** or **Git Bash** and verify the installation by running:

git --version

This should display the installed Git version.

- Git installation checking:



Step 3: Download Flutter SDK

1. Go to the official Flutter website: https://flutter.dev (choose version: 3.27.1)
2. Click on **Get Started → Windows**.
3. Download the latest **Flutter SDK (Windows .zip file)**.

Step 4: Extract Flutter SDK

1. Navigate to the downloaded ZIP file.
2. Extract it to a suitable location, e.g., C:\development\flutter (avoid extracting inside Program Files).
3. Ensure the extracted folder contains the flutter directory.

Step 5: Add Flutter to System Path

1. Open **File Explorer** and go to C:\flutter\bin.
2. Copy the path (C:\flutter\bin).
3. Open **System Environment Variables**:
    - Click **Advanced → Environment Variables**.
4. Under **System Variables**, find Path, select it, and click **Edit**.

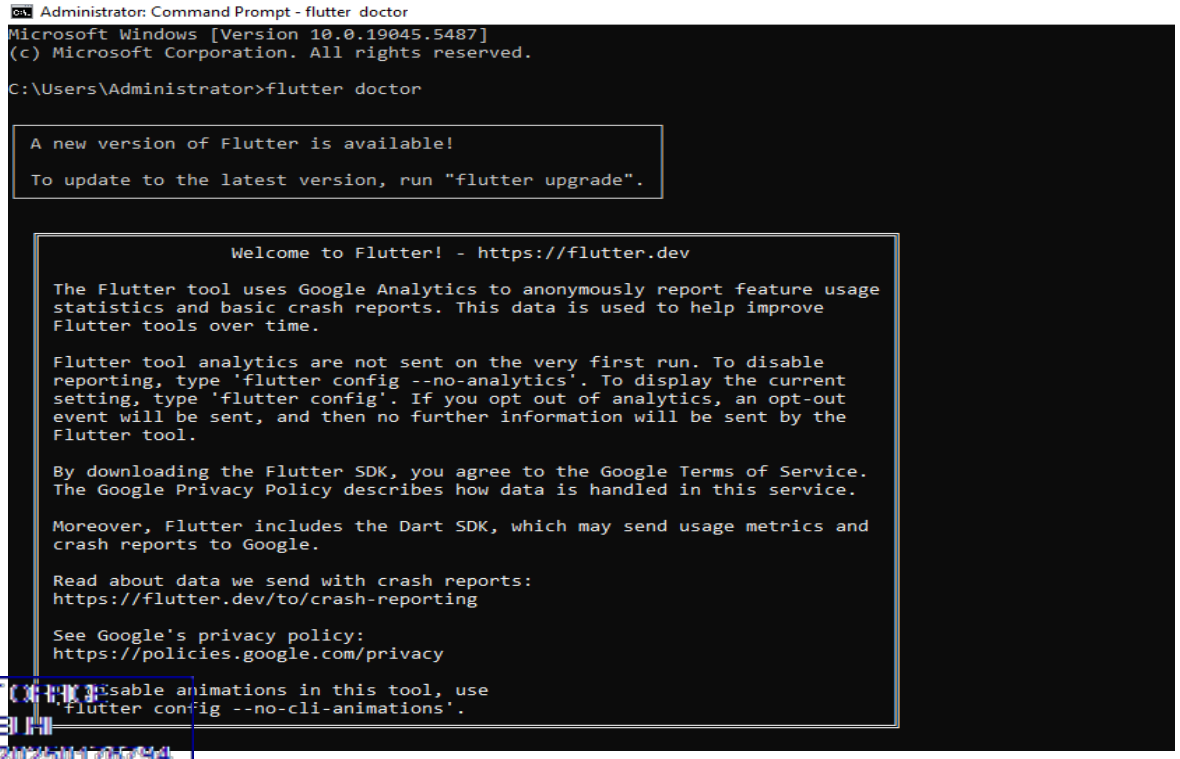
Click **New**, paste C:\development\flutter\bin, and click **OK**.

Step 6: Verify Flutter Installation

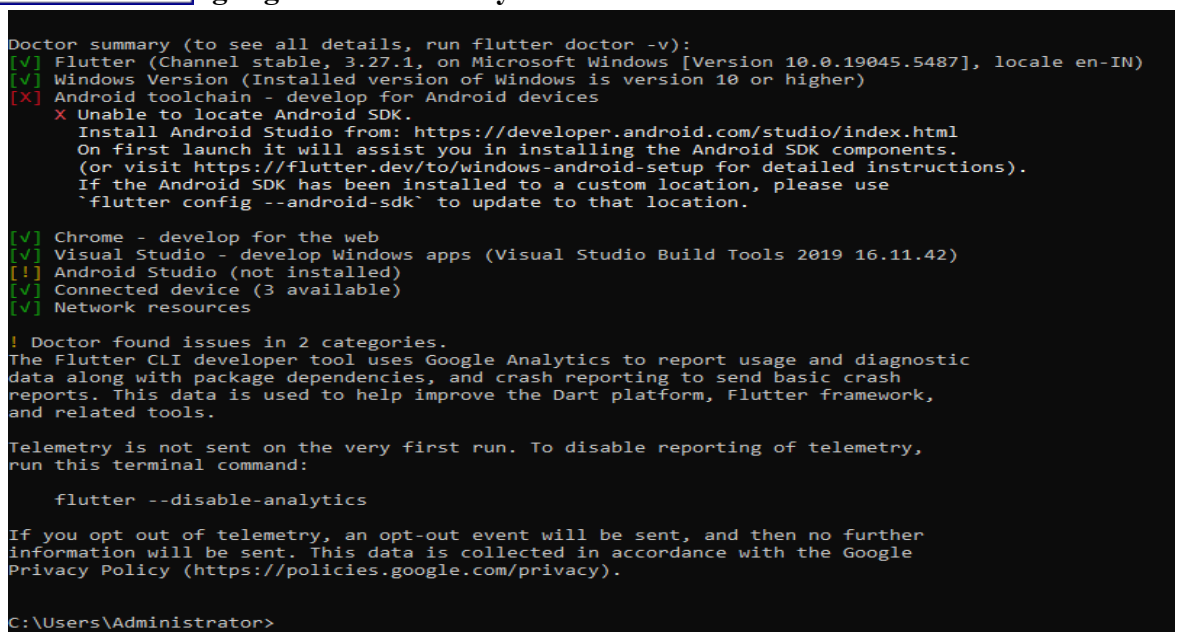
1. Open **Command Prompt (cmd)**.
2. Type the command:

    flutter doctor
3. If Flutter is installed correctly, it will check dependencies and show installation status.

Flutter Installation checking:



Problems are highlighted in red and yellow:

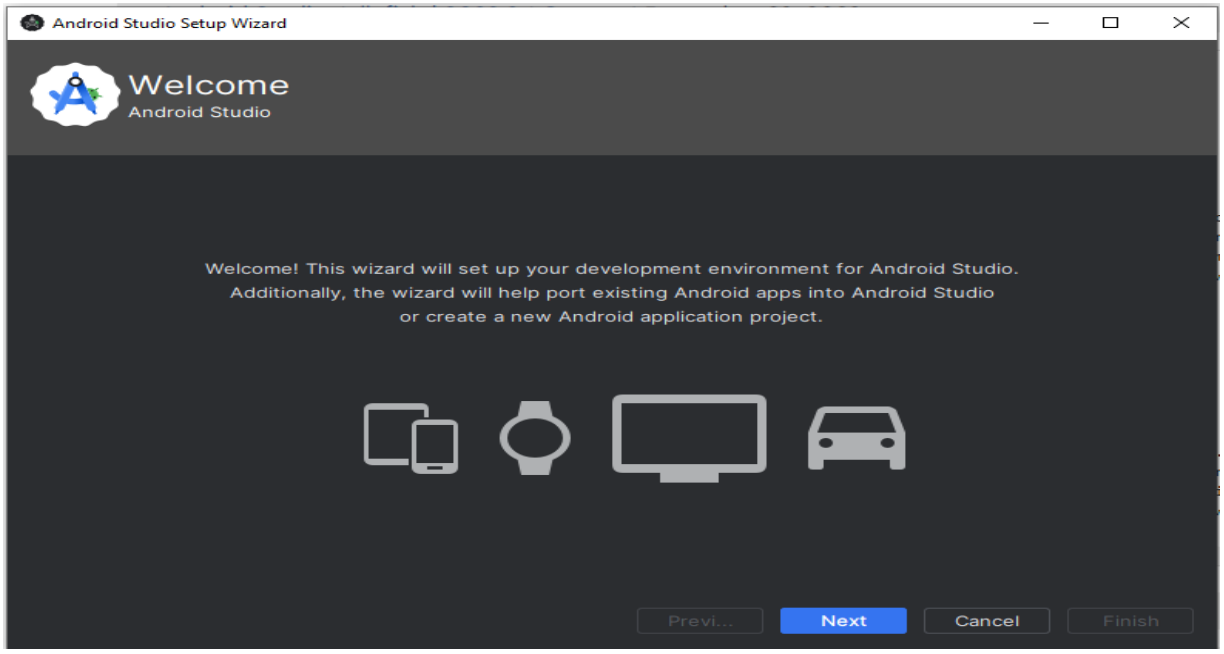


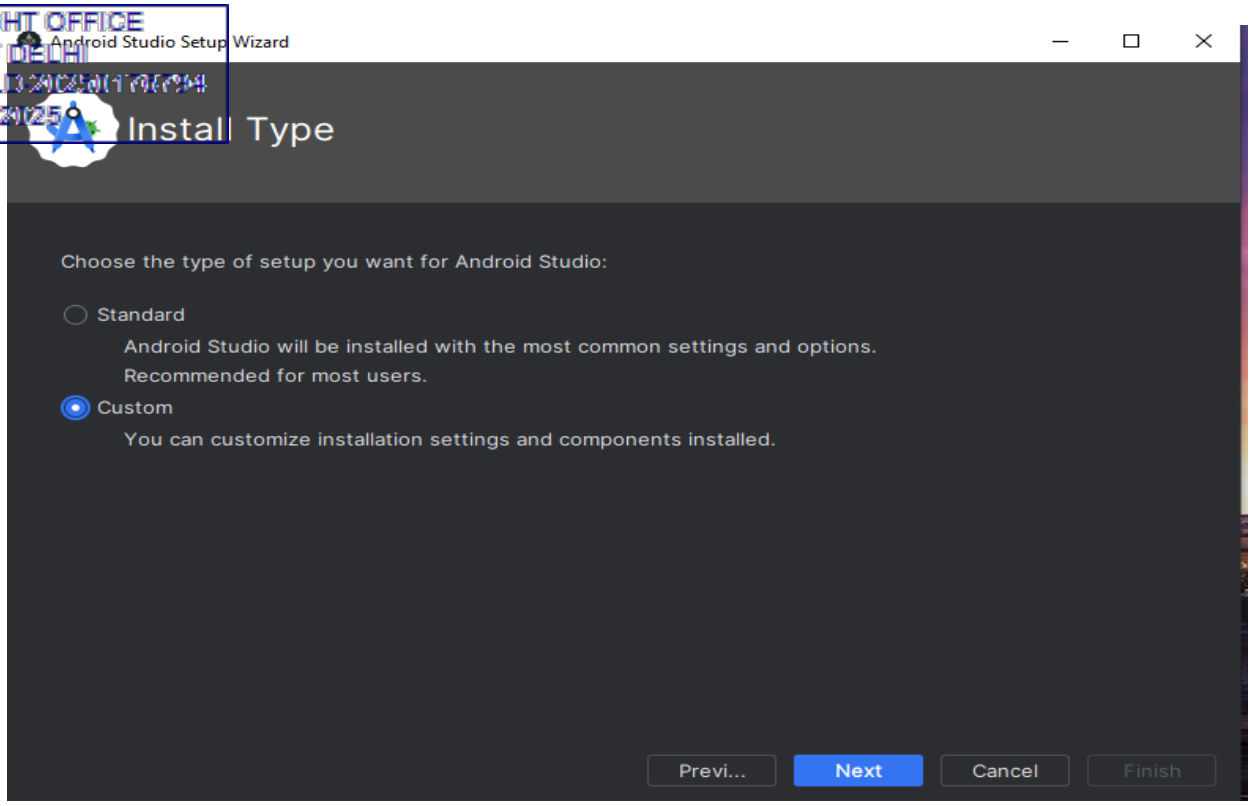
Step 7: Install Android Studio (for Emulator & SDK)

1. Download Android Studio from https://developer.android.com/studio.
2. Run the installer and follow the instructions.
3. Open Android Studio and install:
   - **Flutter Plugin** & **Dart Plugin** (Preferences → Plugins → Search Flutter & Dart → Install).

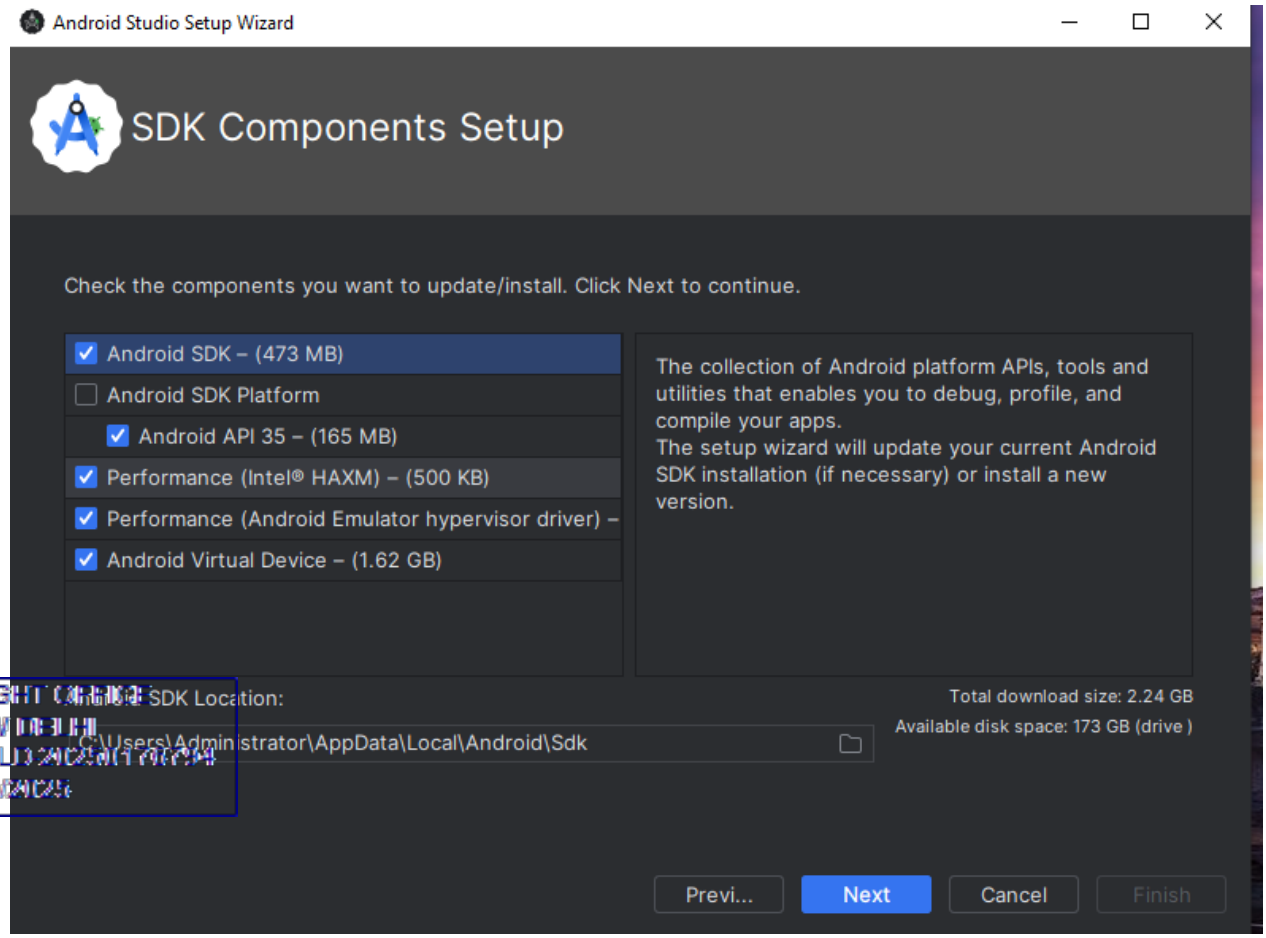
Android SDK (Check under SDK Manager).

Installation of Android after downloading:

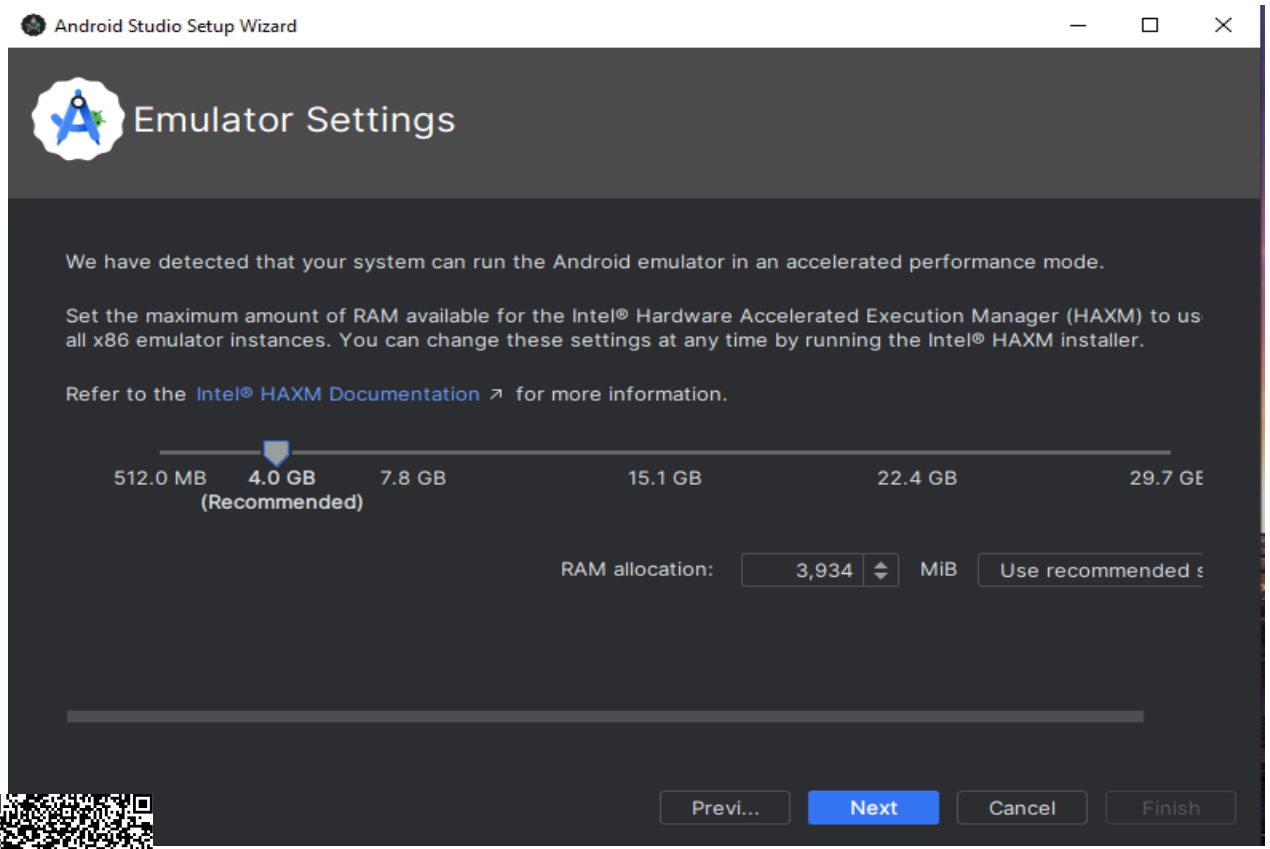




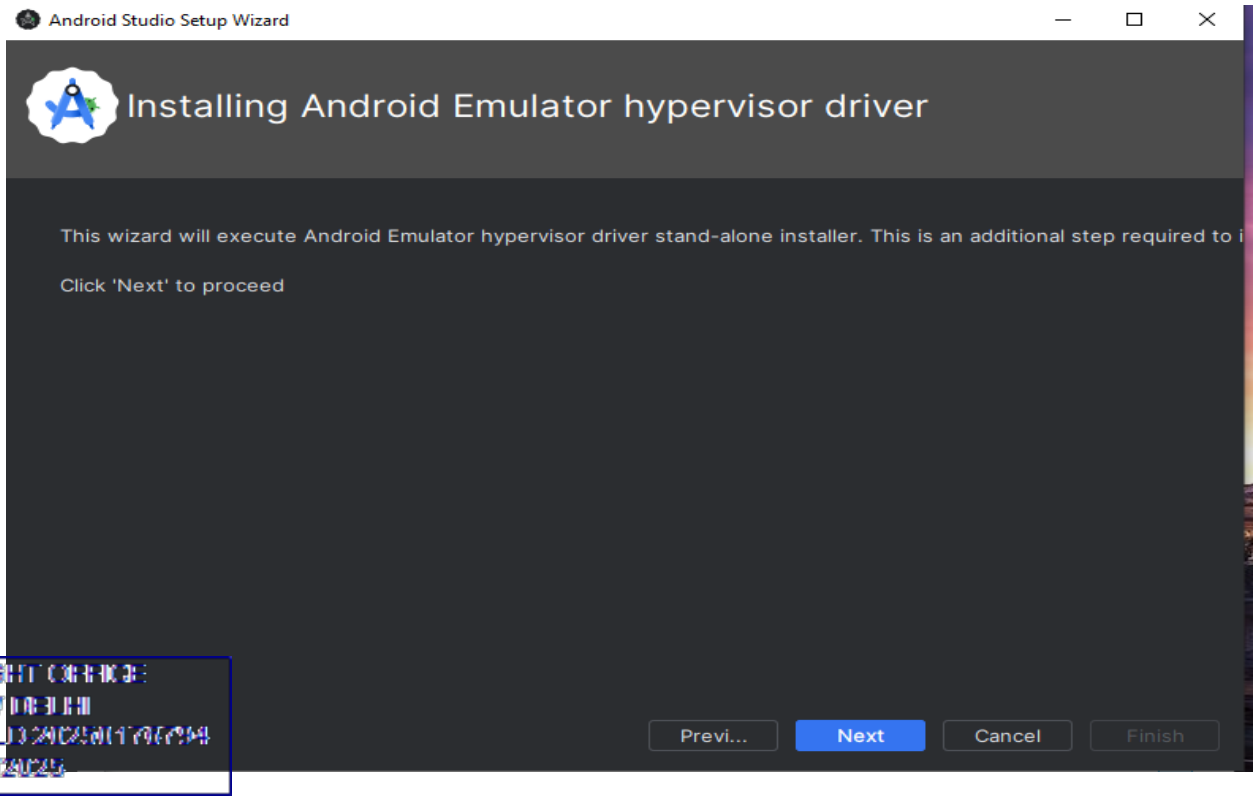
SDK Component location setup:



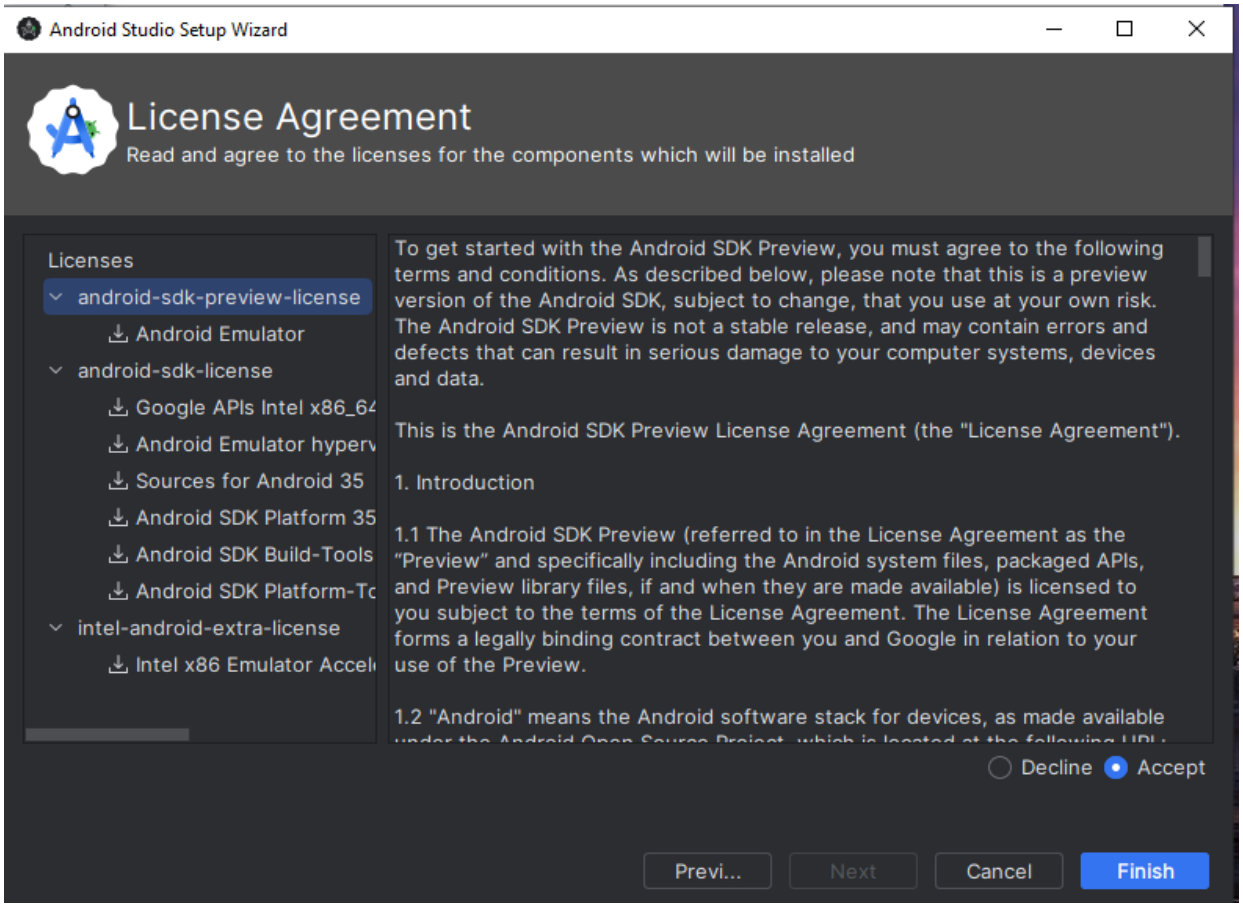
Emulator Settings in the user's system:



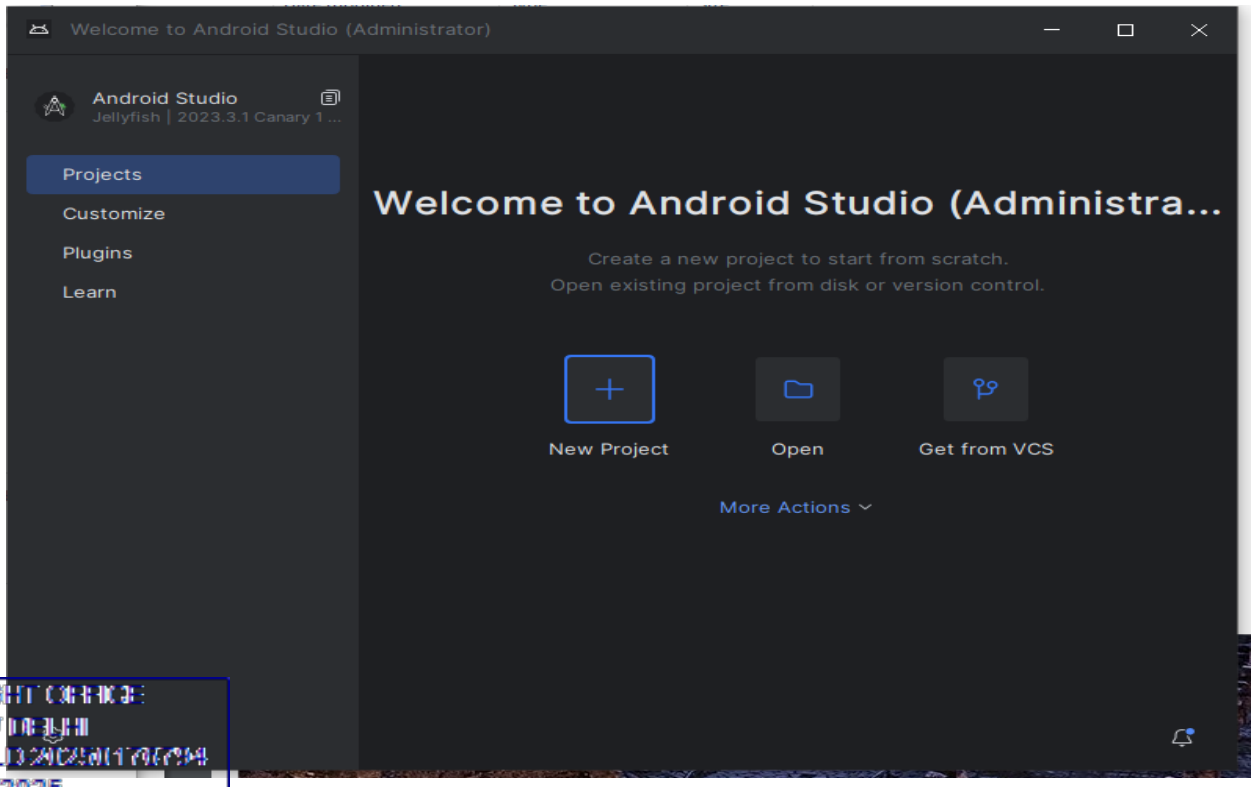
Emulator Installation:



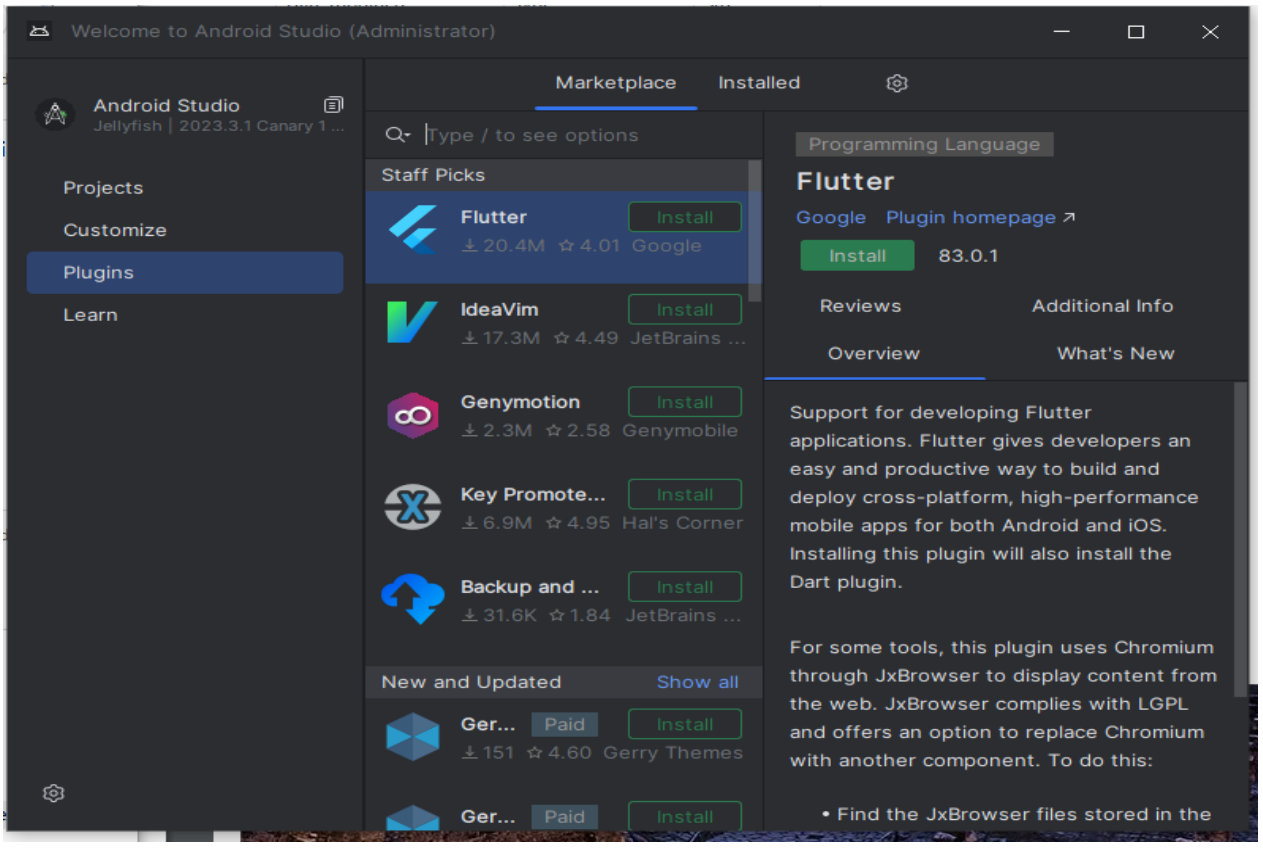
Accept all the Licenses listed



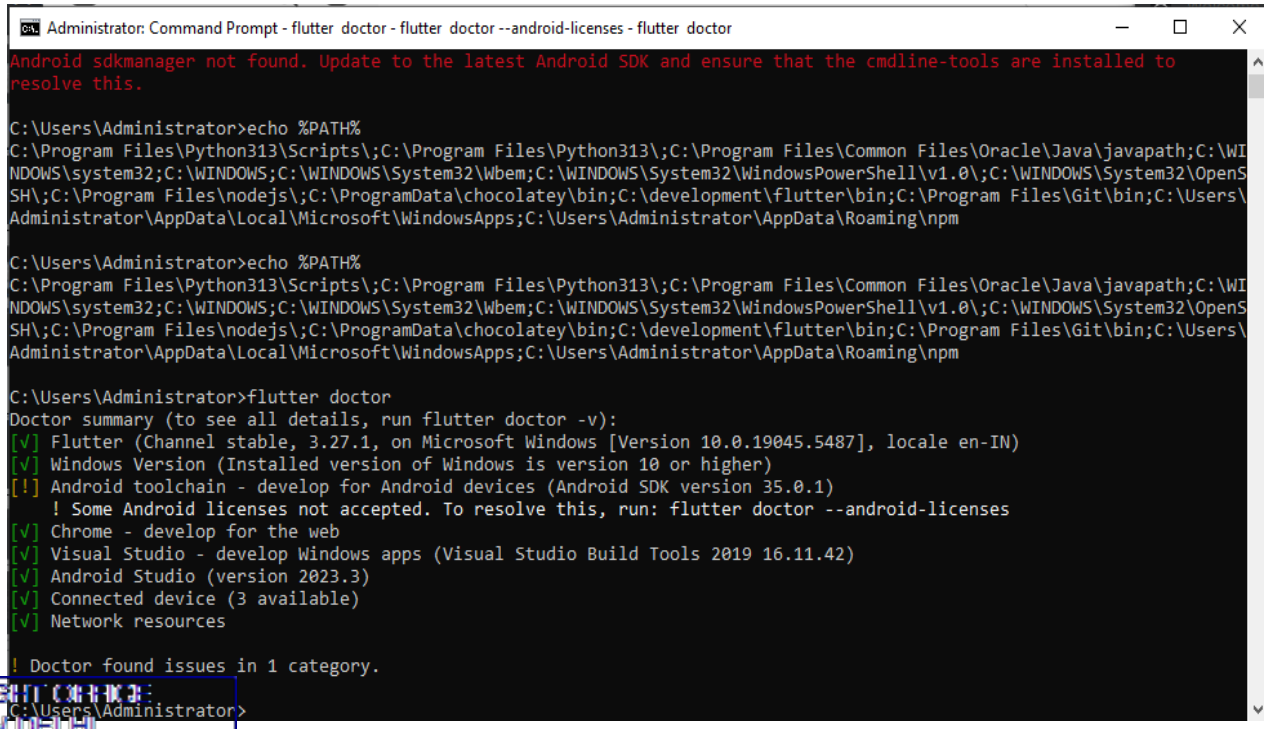
After successful Installation of Android:



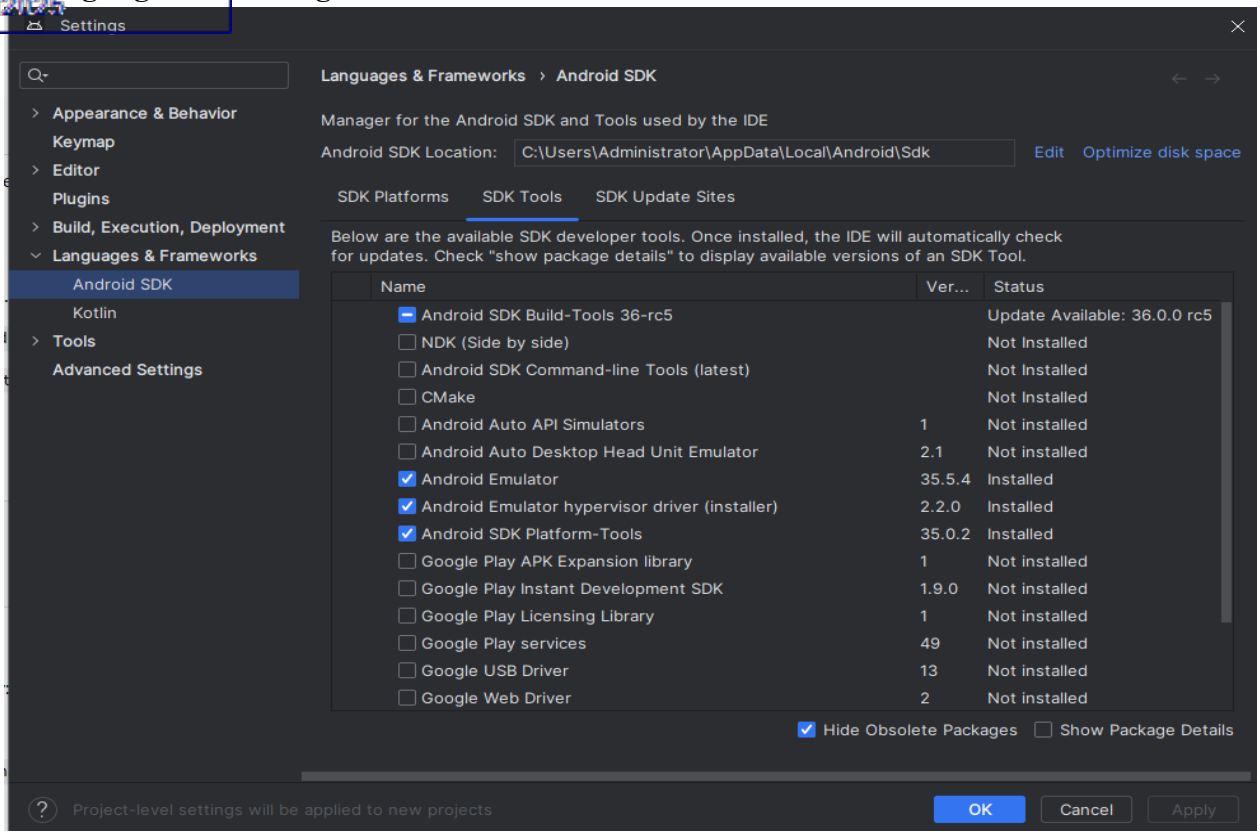
Install Flutter and Dart in Android:



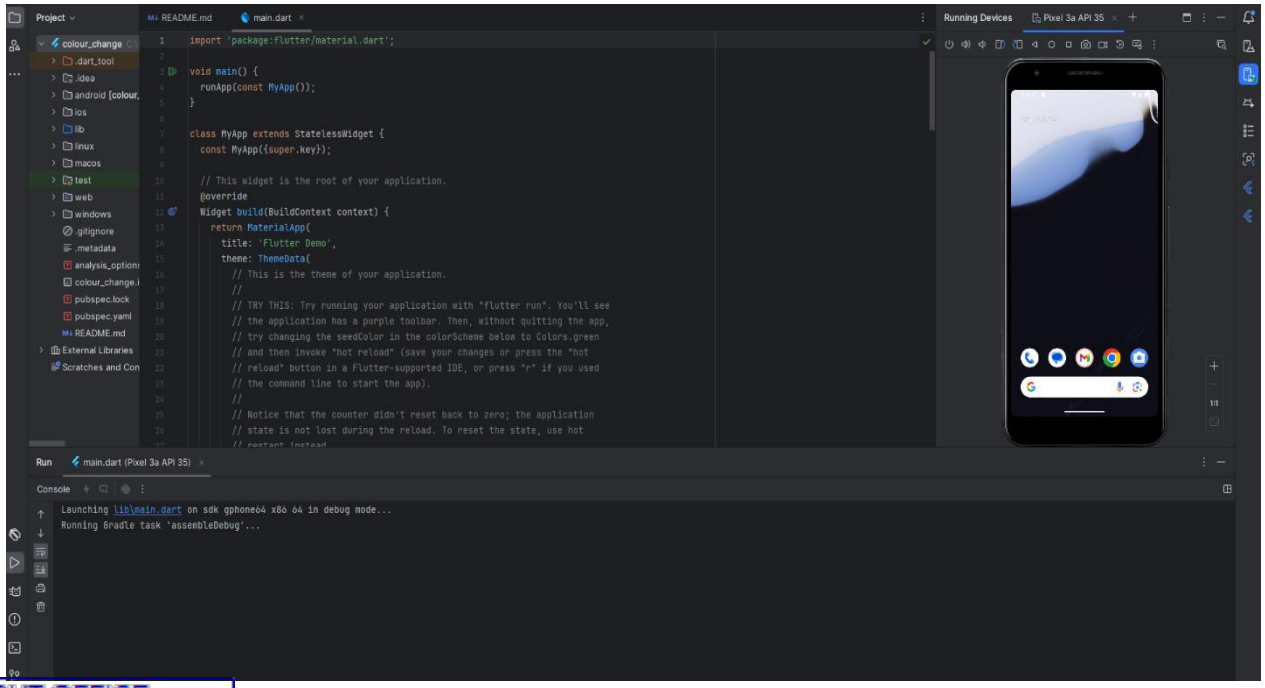
Again, check Flutter is installed properly:



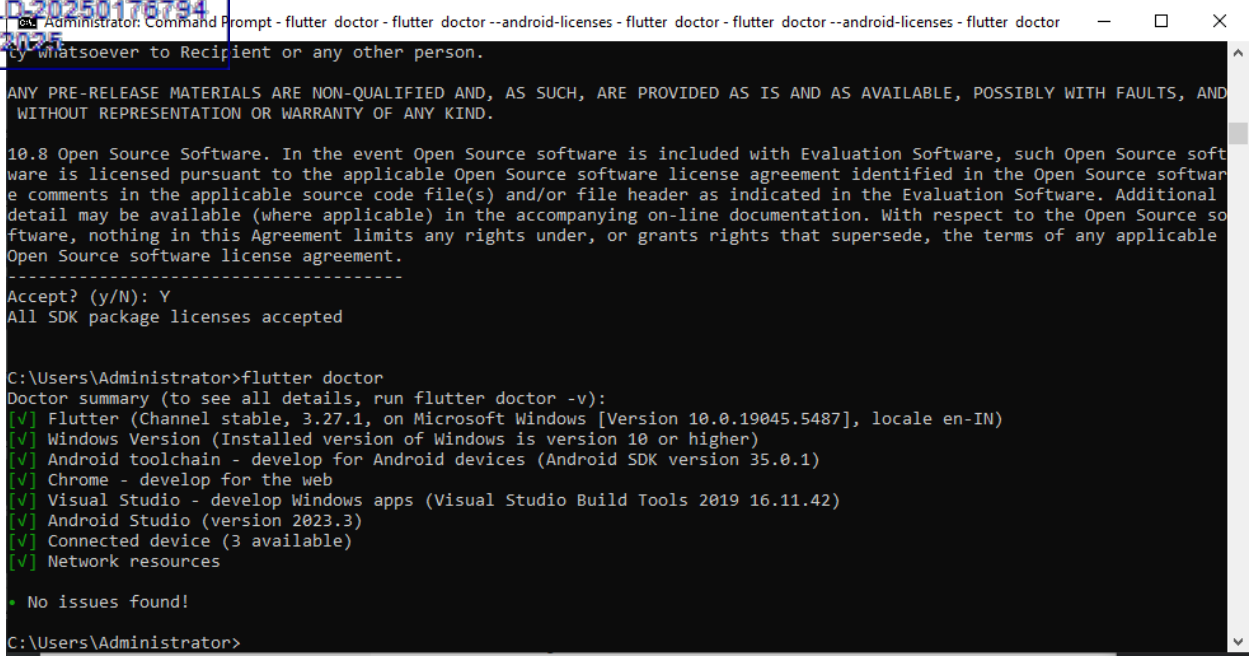
The highlighted warning can be set as below:



Successful Installation of Android Studio with Emulator will be shown as below:



Successful Installation of Flutter with all needed tools will be shown as below:



Step 9: Install VS Code (Optional)

1. Download **Visual Studio Code** from https://code.visualstudio.com.
2. Install the **Flutter Extension**:
   - Open VS Code.
   - Go to **Extensions** (Ctrl + Shift + X).
   - Search for **Flutter** and install it.

Troubleshooting Common Issues

- **Issue: 'flutter' is not recognized as an internal or external command.**
  Solution: Check if C:\flutter\bin is added to the system Path.
- **Emulator not detected.**

- **Solution: Ensure Android Studio is installed and the AVD Manager is properly configured.**

- **Issue: 'git' is not recognized as an internal or external command.**
  - **Solution: Check if C:\Program Files\Git\bin is added to the system Path.**

RESULT:

This, Study and installation of Flutter/Kotlin multi-platform environment installed successfully.

VIVA QUESTIONS

1. What is Flutter, and how does it differ from other mobile app development frameworks?

Flutter is an open-source framework by Google used for building cross-platform mobile applications. Unlike other frameworks, it allows developers to write code once and deploy it on both Android and iOS. Flutter uses the Dart programming language.

2.What is Dart?

Dart is the programming language used in Flutter. It is an object-oriented, class-based, garbage-collected language optimized for building mobile, web, and desktop apps.

3.What is the role of the pubspec.yaml file in Flutter projects?

The pubspec.yaml file is used to manage the dependencies and assets in a Flutter project. It includes information about the app's version, dependencies, fonts, images, etc.

4. What are Widgets in Flutter?

Widgets are the building blocks of a Flutter application. Everything in Flutter is a widget, from the layout (like rows and columns) to UI elements (buttons, text, etc.).

5.What is the difference between Stateful and Stateless Widgets?

A Stateless Widget is immutable and cannot change its state once it's built. A Stateful Widget can change its state over time and rebuild the widget when its state changes.

6.Explain the hot reload feature in Flutter.

Hot reload allows developers to quickly apply code changes to the app without restarting it. It helps speed up the development process by showing changes instantly.

7.How do you manage navigation in a Flutter app?

Flutter provides a Navigator widget to manage app navigation. You can use Navigator.push() to navigate to a new screen, and Navigator.pop() to return to the previous screen.

8.What is Git, and why is it important in software development?

Git is a distributed version control system that helps track changes in code, manage different versions of the software, and collaborate with multiple developers. It is crucial for team-based development and maintaining the history of changes.

9. How do you clone a repository from GitHub and work with Git in your project?

To clone a repository, you use the following command: git clone <repository-url> my-folder

After cloning, you can create branches, make commits, and push changes back to the repository using Git commands like git add, git commit, and git push.

10.How do you debug a Flutter application?

You can use Android Studio or VSCode for debugging Flutter apps. Both offer features like breakpoints, step-through debugging, and inspecting variables. Flutter also provides logs through the console (flutter run).

EX.NO:2	TO DEVELOP A SIMPLE ANDROID APPLICATION THAT USES GUI COMPONENTS, FONT AND COLORS.

AIM:

To develop a Simple Android Application that uses GUI components, Font and Colours.

PROCEDURE:

Step 1: Create a New Project.

a. **Open** Android Studio Application.

b. File-> New-> New Flutter Project

c. Choose **Flutter** and Set the Flutter path: **C:/src/flutter** and press **Next (**src is the folder created by the user)

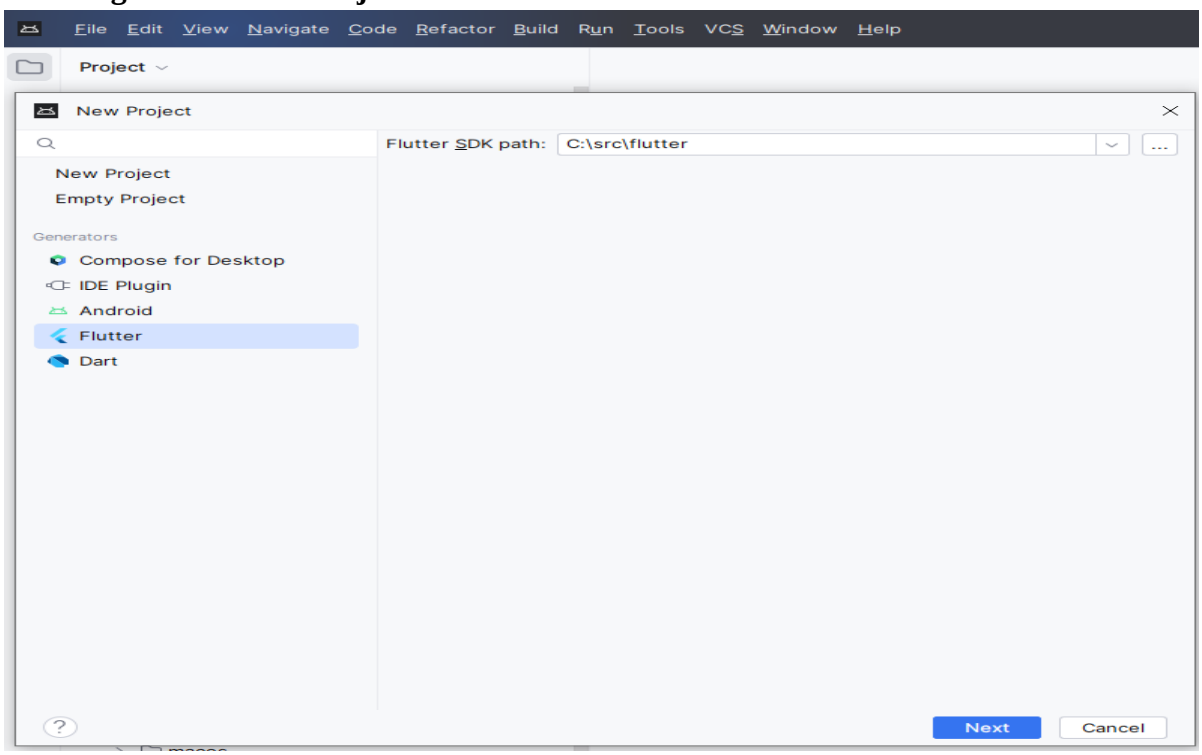
d. **Set** the name for the project and choose the programming language (**java/ Kotlin**) and choose **Create**.

e. New Project will be opened with chosen language.

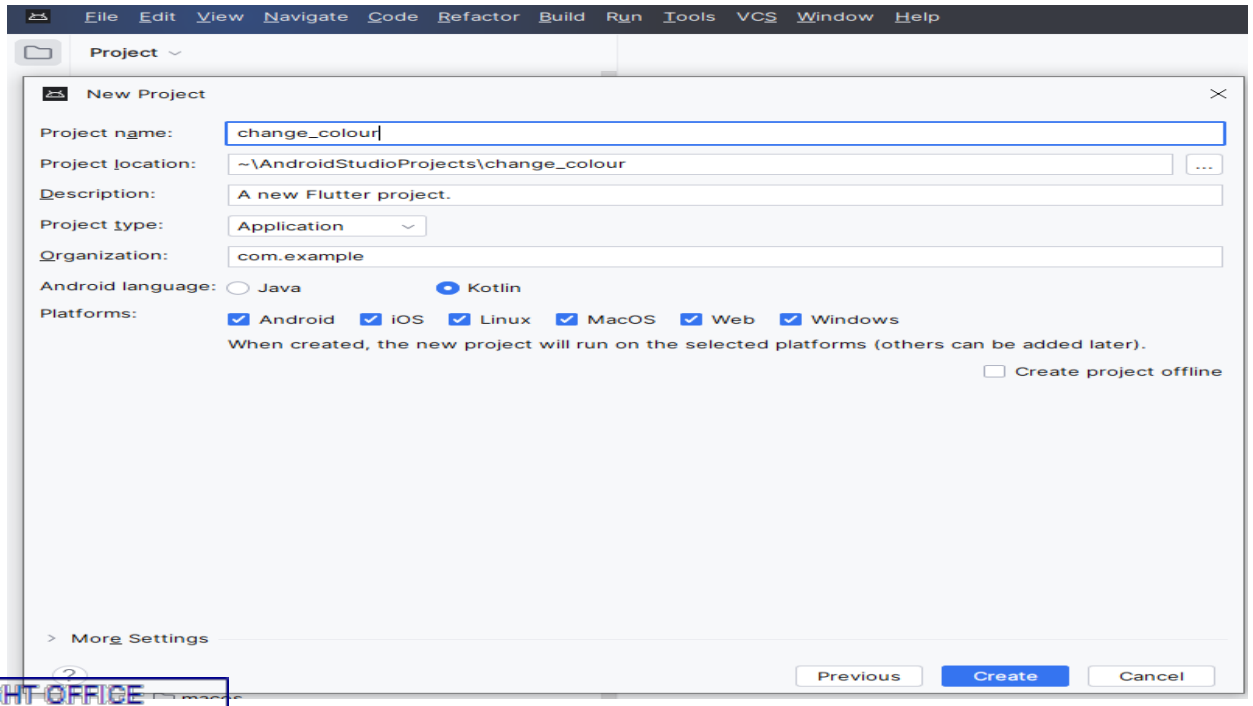
f. Now to add the Emulator refresh the Emulator choice and add the Emulator which we added during the Installation or we can choose the new emulator.

g. Write the coding and run the emulator.

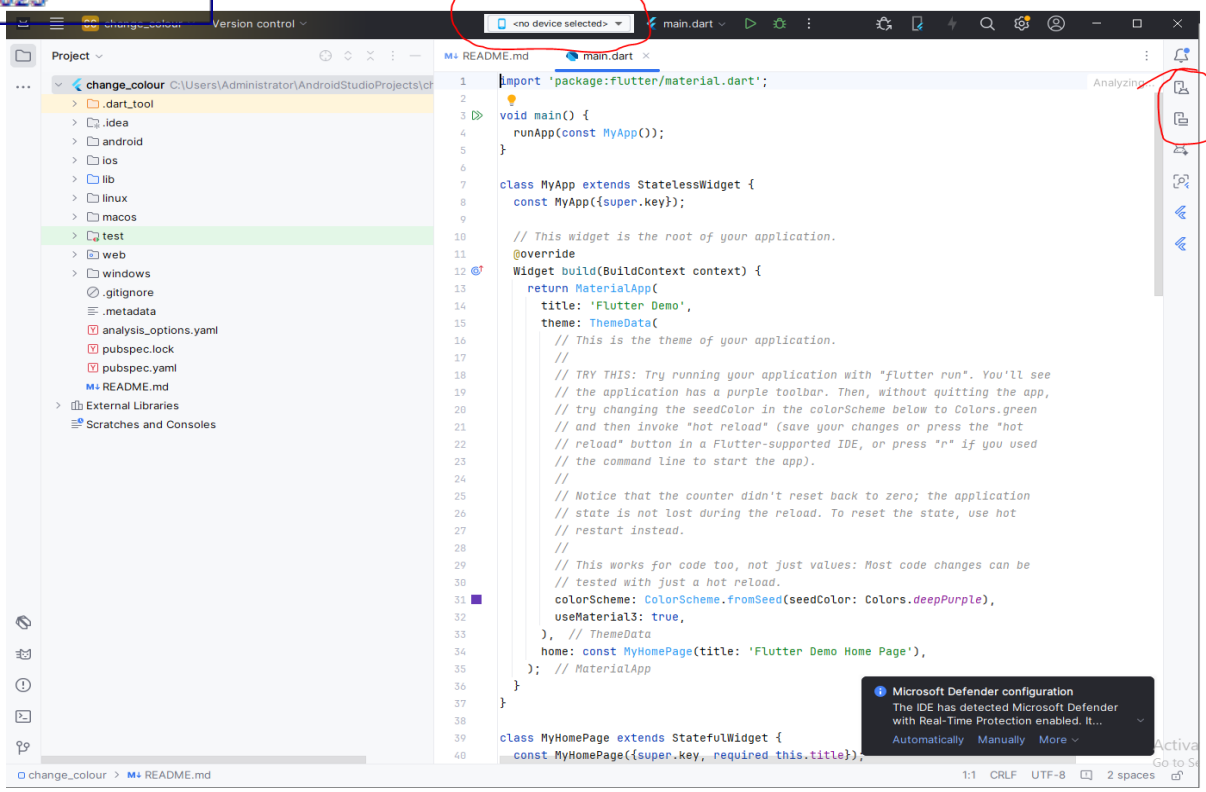
Creating New Flutter Project:



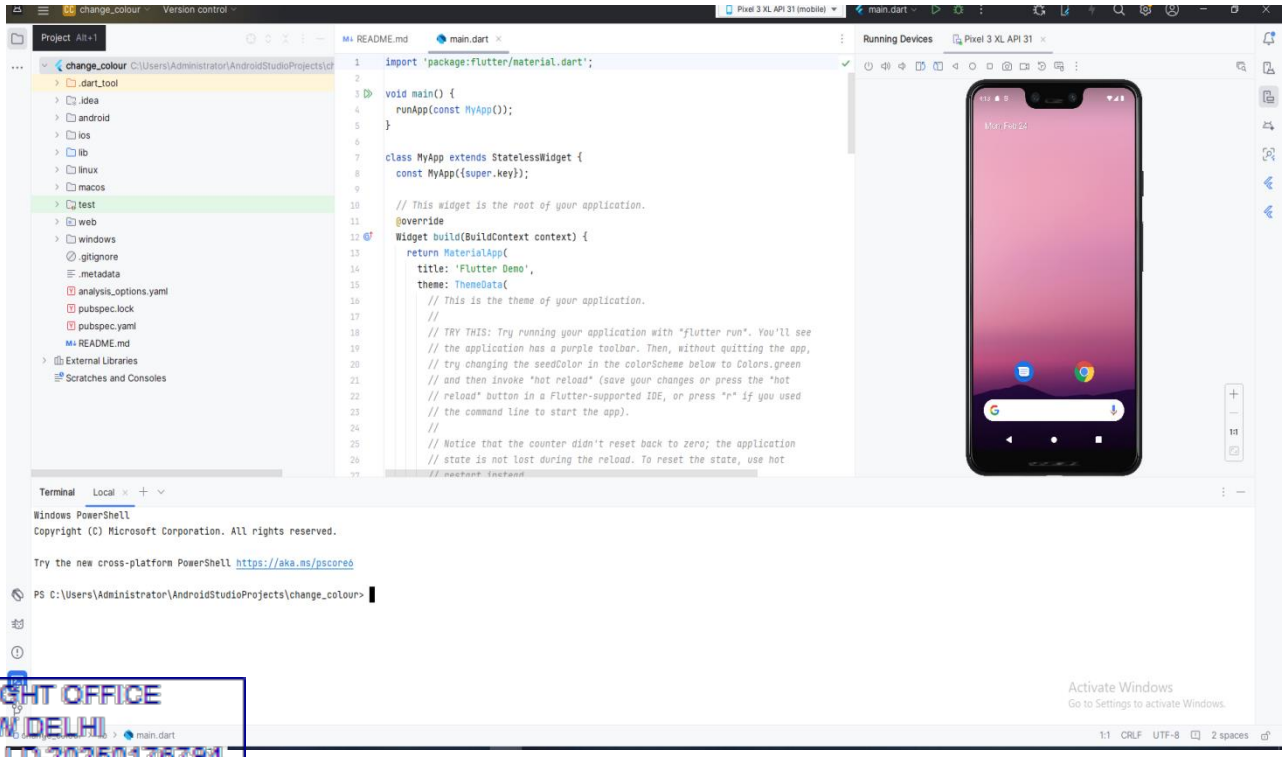
Naming and choosing preferable language for the project:



New Project is created with Kotlin Code without emulator:



choosing the Emulator by refreshing the box we can get this page:



Now we can write coding on main.dart file, which is the main file for Flutter in dart language.

Coding for changing font size and colour:

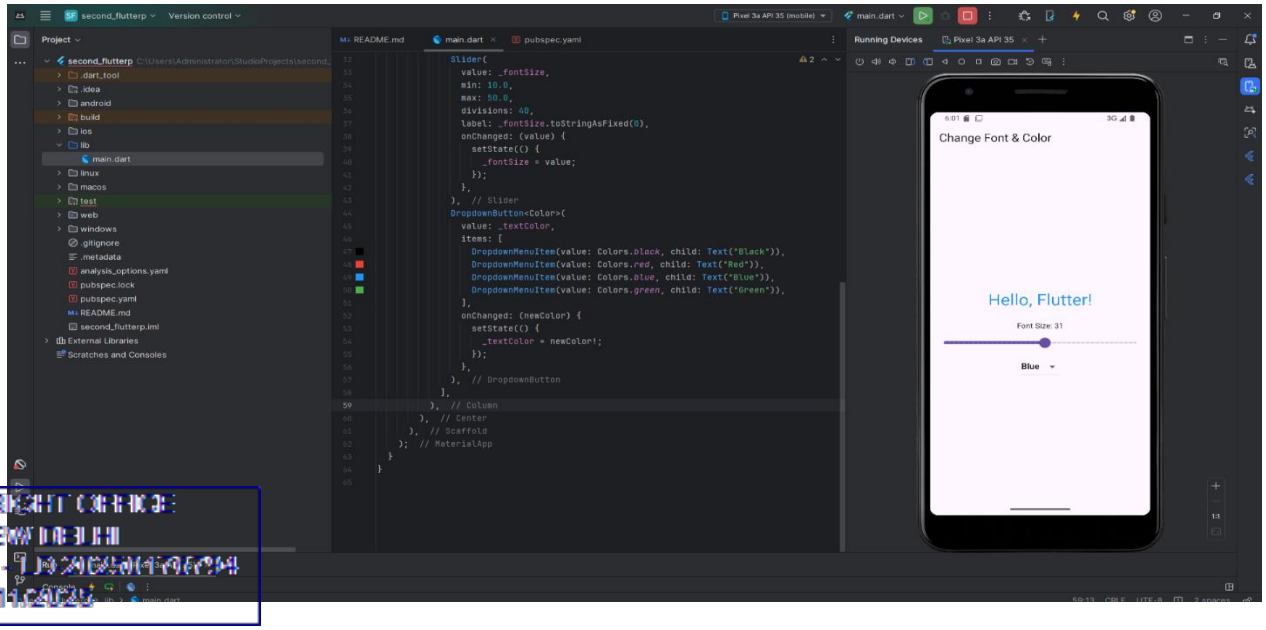
Main.dart

```
import 'package:flutter/material.dart';
void main() {
 runApp(SecondFlutterP());
}class SecondFlutterP extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   home: MyHomePage(),
  );  }}
class MyHomePage extends StatefulWidget {
 @override
 _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
 int _ch = 1;
 double _fontSize = 30.0;
 Color _textColor = Colors.black;
 void _changeFontSize() {
  setState(() {
   _fontSize += 5;
   if (_fontSize == 50) {
    e = 30;
```

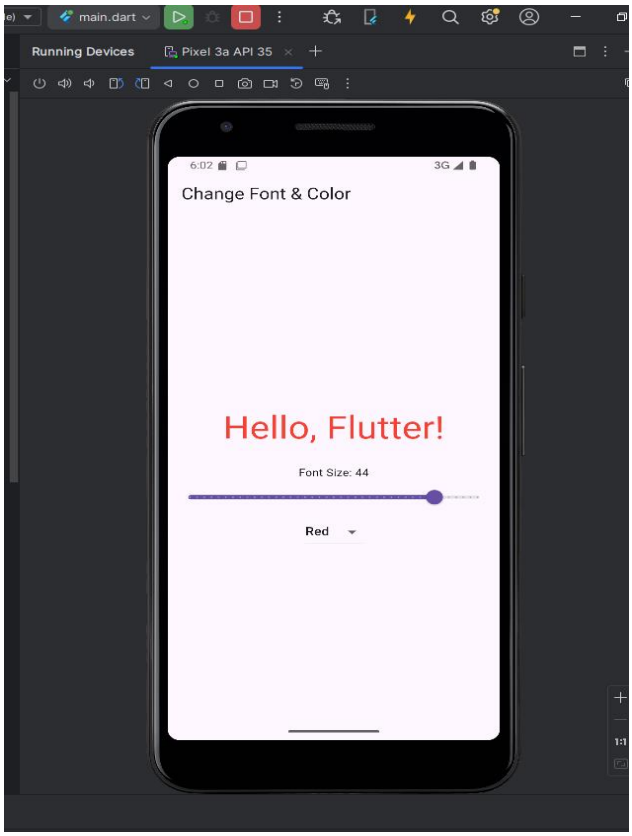
```
void _changeTextColor() {
  setState(() {
    switch (_ch) {
      case 1:
        _textColor = Colors.red;
        break;
      case 2:
        _textColor = Colors.green;
        break;
      case 3:
        _textColor = Colors.blue;
        break;
      case 4:
        _textColor = Colors.cyan;
        break;
      case 5:
        _textColor = Colors.yellow;
        break;
      case 6:
        _textColor = Colors.purple;
        break;
    }     _ch++;
    if (_ch == 7) {
      _ch = 1;
    }   });  }
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Flutter Widgets Example'),
    ),     body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(
            'Hello Flutter!',
            style: TextStyle(
              fontSize: _fontSize,
              color: _textColor,
            ),          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: _changeFontSize,
            child: Text('Change Font Size'),
          ),          SizedBox(height: 20),
          ElevatedButton(
```

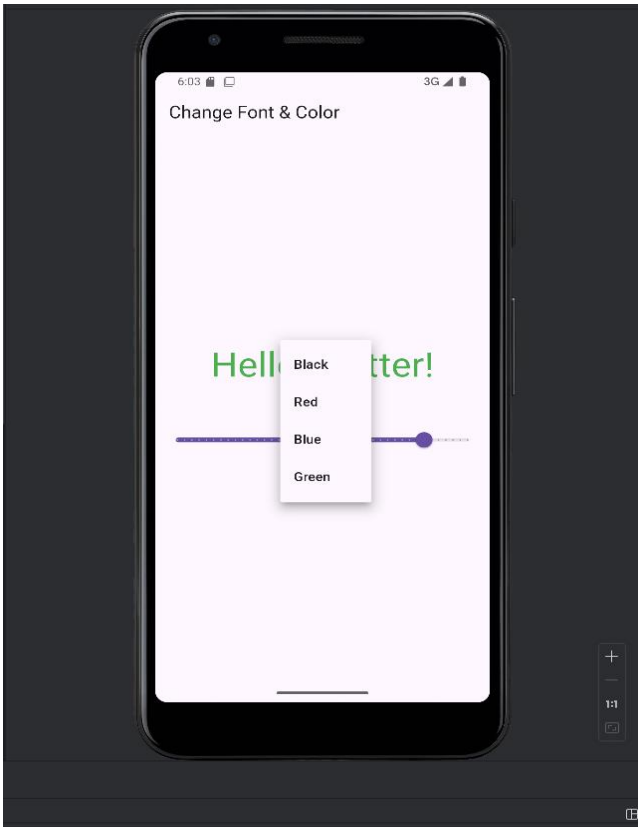
```
onPressed: _changeTextColor,
child: Text('Change Text Color'),
),        ],       ),      ),    ); }}
```

OUTPUT:



Changing colour and font:





RESULT:

Thus, a Simple Android Application that uses GUI components, Font and Colours was executed

VIVA QUESTIONS

1. What is the purpose of the main.dart file in a Flutter application?

The main.dart file is the entry point of a Flutter application. It contains the main() function, which calls runApp(MyApp()) to initialize and start the app.

2. Why do we use the StatefulWidget in this application instead of StatelessWidget?

We use a StatefulWidget because the app's UI changes dynamically when the font size or text color is modified. A StatelessWidget cannot update its state after being built, whereas a StatefulWidget can, using setState().

3. Explain the role of the setState method in Flutter. Why is it necessary?

The setState method tells Flutter to rebuild the widget tree when there is a change in state. In this app, it is used to update the font size and text color dynamically when the slider or dropdown selection changes.

4. How does the Slider widget function in this code? What is its purpose?

The Slider widget allows users to select a value within a range. In this app, it lets users adjust the font size dynamically. The onChanged callback updates _fontSize, and setState() triggers a UI refresh.

5. What are DropdownButton and DropdownMenuItem used for in this application?

DropdownButton<Color> creates a dropdown menu that allows users to select a color. Each DropdownMenuItem<Color> represents a color option.

When a new color is selected,

setState() updates _textColor, changing the text color dynamically.

6. How can you add more color options to the DropdownButton?

To add more color options, include additional
DropdownMenuItem<Color> elements in the items list.
Example:
DropdownMenuItem(value: Colors.purple, child: Text("Purple")),
DropdownMenuItem(value: Colors.orange, child: Text("Orange")),

7. What is the significance of the debugShowCheckedModeBanner: false line in MaterialApp?

This line removes the "Debug" banner in the top right corner when running the app in debug mode. It helps provide a cleaner UI during testing.

8. Explain how font size changes dynamically in this application.

The font size is controlled by _fontSize, which is updated by the Slider.
When the user moves the slider, setState() updates _fontSize, causing the Text widget to rebuild with the new font size.

9. What modifications would you make to allow users to change the font family as well?

You can add a DropdownButton<String> to select different font families and modify the TextStyle in the Text widget:
TextStyle(fontSize: _fontSize, color: _textColor, fontFamily: _selectedFontFamily),
Then, update _selectedFontFamily based on user selection.

10. How does the mainAxisAlignment: MainAxisAlignment.center property affect the layout of widgets?

This property aligns the children of the Column widget along the main axis (vertically) at the center of the screen, ensuring that all elements appear in the middle.

EX.NO:3	DEVELOP A NATIVE CALCULATOR APPLICATION

AIM:

To develop a Simple Android Application for Native Calculator.

PROCEDURE:

Step 1: Open Android Studio and then click on File -> New -> New project.

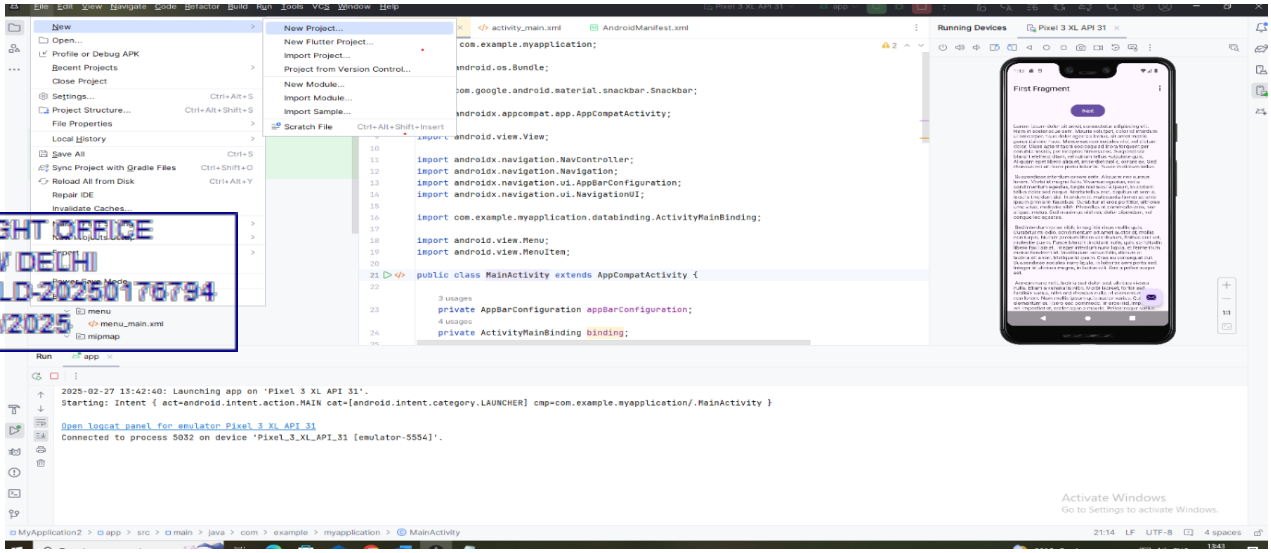
Step 2: Select Basic view Activity and select next.

Step 3: Give name for the project, language in which code has to be done and give finish**.**

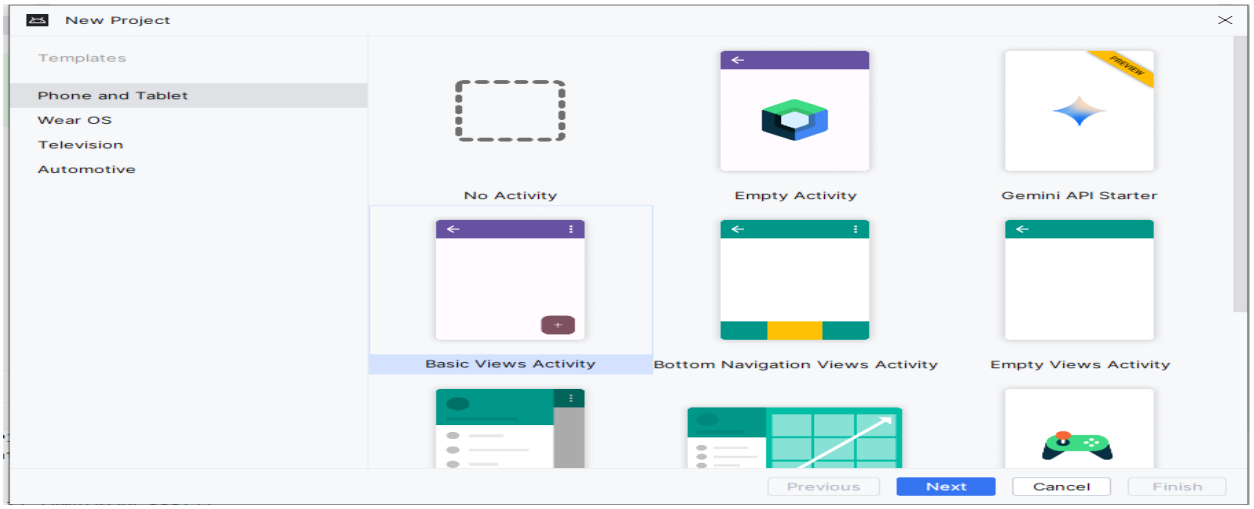
Steo 4: New project with be opened with chosen language pattern**.**

Step 5: Update the coding for calculator.

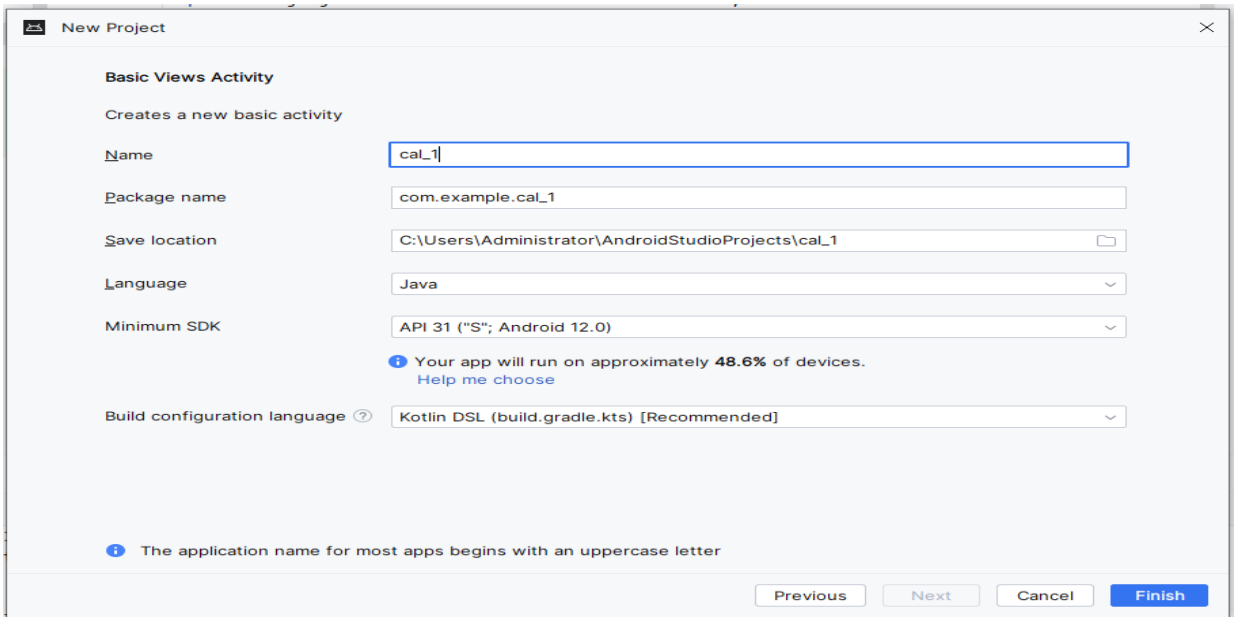
Creating new Android Native Project:



Select Basic Views Activity:



Give name for the Project and choose the language:



Coding:

Structure of the file:

app

------ manifests

----------AndroidManifest.xml

--------java

----------MainActivity.java

--------res

----------layout

----------activity_main.xml (designing file)

********** If you are using the below code for your project do not change the header files and packages already created in your project. Because those automatically created skeleton will have the packages and header for your project name. ******************************

AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        roundIcon="@mipmap/ic_launcher_round"
        supportsRtl="true"
```

```
        android:theme="@style/Theme.Calculator_1"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.Calculator_1">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/num1EditText"
        android:layout_width="0dp"
        android:layout_height="48dp"
        android:layout_marginTop="44dp"
        android:hint="Enter number 1"
        android:inputType="numberDecimal"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/num2EditText"
        android:layout_width="0dp"
        android:layout_height="48dp"
        android:layout_marginTop="12dp"
        android:hint="Enter number 2"
        inputType="numberDecimal"
        ut_constraintEnd_toEndOf="parent"
        ut_constraintHorizontal_bias="0.47"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/num1EditText" />

<Button
    android:id="@+id/addButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="+"
    android:textSize="16sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/num2EditText" />
<Button
    android:id="@+id/subtractButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="-"
    android:textSize="16sp"
    app:layout_constraintEnd_toStartOf="@+id/multiplyButton"
    app:layout_constraintStart_toEndOf="@+id/addButton"
    app:layout_constraintTop_toBottomOf="@+id/num2EditText" />

<Button
    android:id="@+id/multiplyButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="x"
    android:textSize="16sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/num2EditText" />

<Button
    android:id="@+id/divideButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="/"
    android:textSize="16sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/addButton" />

    id="@+id/sqrtButton"
    layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="140dp"
        android:text="Sqrt"
        android:textSize="16sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/subtractButton" />
    <TextView
        android:id="@+id/resultTextView"
        android:layout_width="84dp"
        android:layout_height="41dp"
        android:layout_marginStart="4dp"
        android:layout_marginTop="40dp"
        android:text="Result: "
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/divideButton" />
</androidx.constraintlayout.widget.ConstraintLayout>
```


MainActivity.java

```
package com.example.calculator_1;
import android.os.Bundle;
import com.google.android.material.snackbar.Snackbar;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import java.text.DecimalFormat;
import android.view.View;
import com.example.calculator_1.R;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;
import com.example.calculator_1.databinding.ActivityMainBinding;
import android.view.Menu;
import android.view.MenuItem;
public class MainActivity extends AppCompatActivity {
    // Declare variables to hold references to UI elements
    private EditText num1EditText, num2EditText;
    private TextView resultTextView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Create(savedInstanceState);
        ntView(R.layout.activity_main);
        ize UI elements from the layout
```

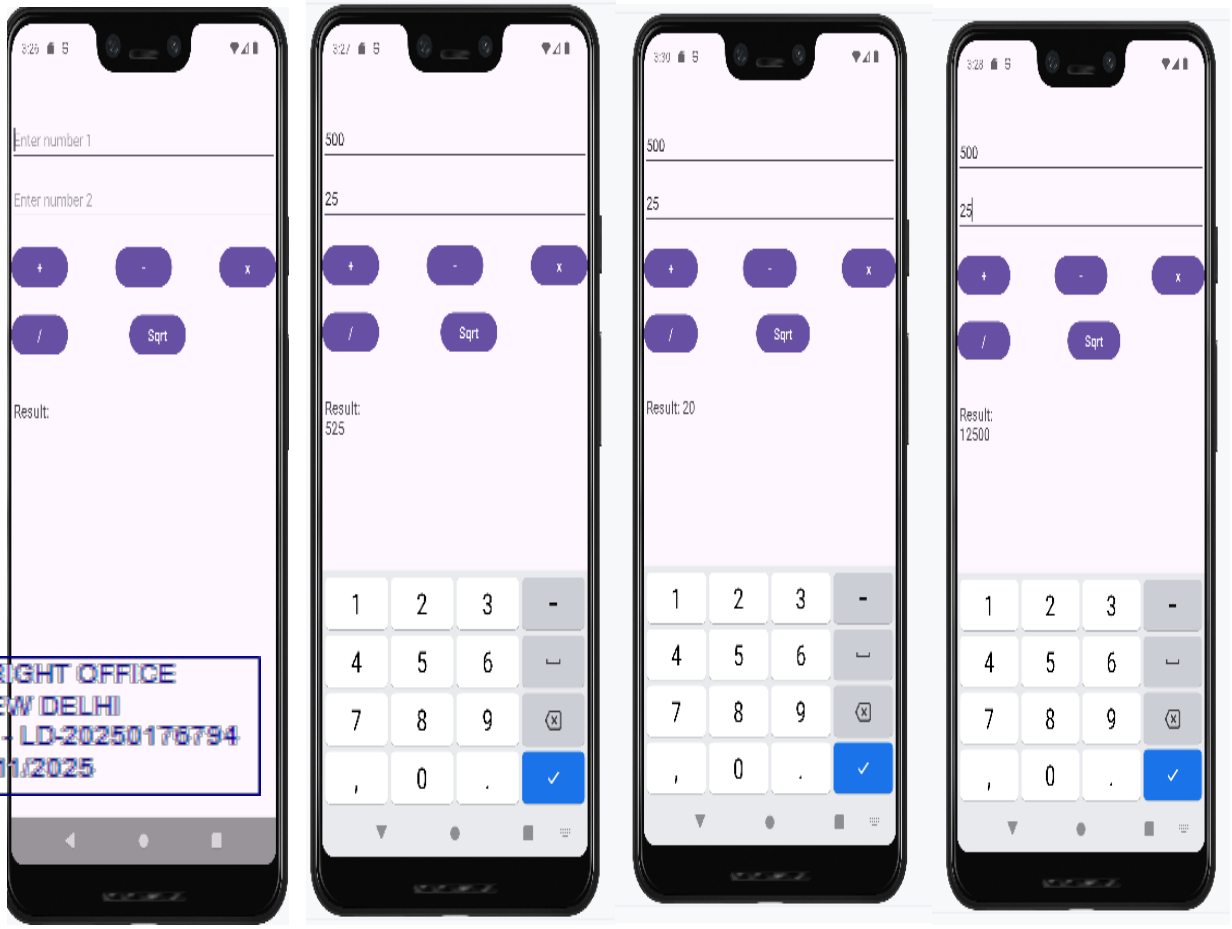
```
        num1EditText = findViewById(R.id.num1EditText);
        num2EditText = findViewById(R.id.num2EditText);
        resultTextView = findViewById(R.id.resultTextView);
        // Set click listeners for arithmetic operation buttons
        Button addButton = findViewById(R.id.addButton);
        addButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                performCalculation('+');
            }       });
        Button subtractButton = findViewById(R.id.subtractButton);
        subtractButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                performCalculation('-');
            }       });
        Button multiplyButton = findViewById(R.id.multiplyButton);
        multiplyButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                performCalculation('*');
            }       });
        Button divideButton = findViewById(R.id.divideButton);
        divideButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                performCalculation('/');
            }       });
        Button sqrtButton = findViewById(R.id.sqrtButton);
        sqrtButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                calculateSquareRoot();
            }       });    }
    private void performCalculation(char operator) {
        // Get the values entered in the input fields
        String num1Str = num1EditText.getText().toString();
        String num2Str = num2EditText.getText().toString();
        // Check if either input field is empty
        if (num1Str.isEmpty() || num2Str.isEmpty()) {
            Toast.makeText(this, "Please enter both numbers", Toast.LENGTH_SHORT).show();
            return; // Exit the method to prevent calculations with empty inputs
        }
        // Convert the input values to numeric format
        double num1 = Double.parseDouble(num1Str);
        double num2 = Double.parseDouble(num2Str);
```

```
        double result = 0;
        // Perform the selected calculation based on the operator
        switch (operator) {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 * num2;
                break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                } else {
                    Toast.makeText(this, "Cannot divide by zero", Toast.LENGTH_SHORT).show();
                    return; // Exit the method if division by zero is attempted
                }
                break;
        }
        // Format and display the calculation result
        DecimalFormat df = new DecimalFormat("#.##");
        resultTextView.setText("Result: " + df.format(result));
    }
    private void calculateSquareRoot() {
        String num1Str = num1EditText.getText().toString();
        // Check if the input field is empty
        if (num1Str.isEmpty()) {
            Toast.makeText(this, "Please enter a number", Toast.LENGTH_SHORT).show();
            return; // Exit the method to prevent calculations with empty inputs
        }

        double num = Double.parseDouble(num1Str);
        double sqrtResult = Math.sqrt(num);
        DecimalFormat df = new DecimalFormat("#.##");
        resultTextView.setText("Square Root: " + df.format(sqrtResult));     }}
```


RESULT:

Thus, the Calculator Home Screen is designed and Addition, Subtraction, Multiplication, Division and Square root operations are checked.

OUTPUT:



RESULT:

Thus, the simple Android native Calculator is created and operations are checked.

VIVA QUESTIONS

1. Why do we use ConstraintLayout in this calculator application?

ConstraintLayout is used in this application because it allows flexible positioning of UI elements using constraints. It helps in creating a responsive and efficient layout without nested LinearLayout or RelativeLayout, improving performance and adaptability across different screen sizes.

2. Which layout is used in the activity_main.xml file?

The application uses ConstraintLayout in activity_main.xml to arrange UI components efficiently with constraints.

3. How do you retrieve user input from an EditText in Android?

User input is retrieved using the getText().toString() method from an EditText component. Example:

String num1Str = num1EditText.getText().toString();

4. What happens if the user tries to divide by zero?

The application checks if the denominator is zero before performing division. If the user tries to divide by zero, a toast message is displayed:

"Cannot divide by zero."

5. How do you set click listeners for buttons in Android?

Click listeners are set using setOnClickListener(), which triggers a method when the button is clicked. Example:

```
addButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        performCalculation('+');
    }
});
```


6. What is the role of DecimalFormat in this application?

DecimalFormat is used to format the result to two decimal places, ensuring that the displayed result is more readable. Example:

```
DecimalFormat df = new DecimalFormat("#.##");
resultTextView.setText("Result: " + df.format(result));
```

7. What changes are required in AndroidManifest.xml to declare the main activity?

The following <activity> tag should be included inside the <application> tag:

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:theme="@style/Theme.Exno3ja">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

This ensures that MainActivity is the entry point of the application.

8. How does the application handle empty input fields?

The app checks whether the input fields are empty before performing any calculation. If empty, it displays a toast message asking the user to enter both numbers. Example:

```
if (num1Str.isEmpty() || num2Str.isEmpty()) {
    Toast.makeText(this, "Please enter both numbers", Toast.LENGTH_SHORT).show();
    r
```

}

9. What is the significance of tools:context=".MainActivity" in activity_main.xml?

It is used by Android Studio to link the XML layout file to the corresponding activity (MainActivity.java). This helps in design previews but does not affect runtime execution.

10. How does the square root function work in this application?

The square root function retrieves the first number entered by the user, calculates the square root using Math.sqrt(), formats the result using DecimalFormat, and displays it in the TextView. Example:

```
double num = Double.parseDouble(num1Str);
double sqrtResult = Math.sqrt(num);
resultTextView.setText("Square Root: " + df.format(sqrtResult));
```


EX.NO:4	DEVELOP A GAMING APPLICATION THAT USES 2-D ANIMATIONS AND GESTURES.

AIM:

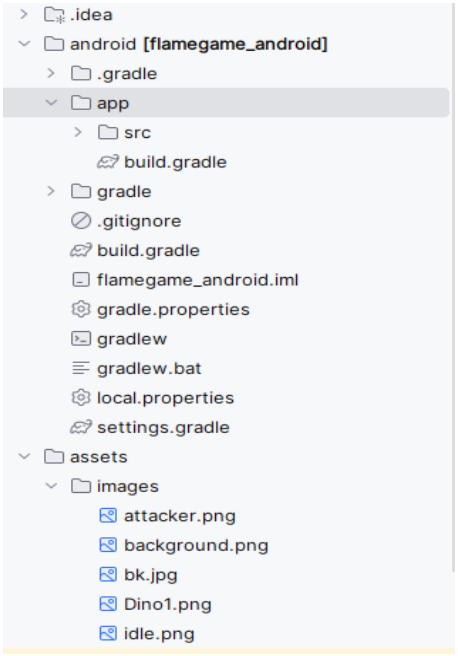
Develop a gaming application that uses 2-D animations and gestures using flutter.

PROCEDURE:

- It is a Flutter Application.
- Folder Structure: (Create app development programs in the below structure)

lib/
├── attacker.dart
├── dino_game.dart
├── dino_player.dart
├── dino_world.dart
├── main.dart
└── directions.dart

- Add images in assets



- **Pubspec.yaml** (Here you have to change only needed dependencies and assets. Do not change this automatic created file for your project. Avoid copy and pasting)

```
name: flamegame
description: "A new Flutter project."
publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1
environment:
  sdk: ^3.6.0
dependencies:
  flutter:
    sdk: flutter
  flame: ^1.1.1
  cupertino_icons: ^1.0.8
```

```
dev_dependencies:
  flutter_test:
    sdk: flutter
flutter_lints: ^5.0.0
flutter:
uses-material-design: true
assets:
     - assets/images/
```

What is Flame?

Flame is a **2D game engine for Flutter** that provides tools and utilities to simplify game development. It is built on top of Flutter's CustomPainter and GameLoop to efficiently render and update game elements. Flame integrates well with Flutter and supports game features like:

- **Sprites & Animations**
- **Collision Detection**
- **Physics Engine** (via Forge2D)
- **Audio & Input Handling**
- **Parallax Backgrounds**
- **Camera & Viewport Handling**

Why is Flame Used in a 2D Dino Game?

A **Dino game** (like the Chrome offline T-Rex game) is a simple 2D infinite runner where the dinosaur jumps over obstacles. **Flame** is used in this game because:

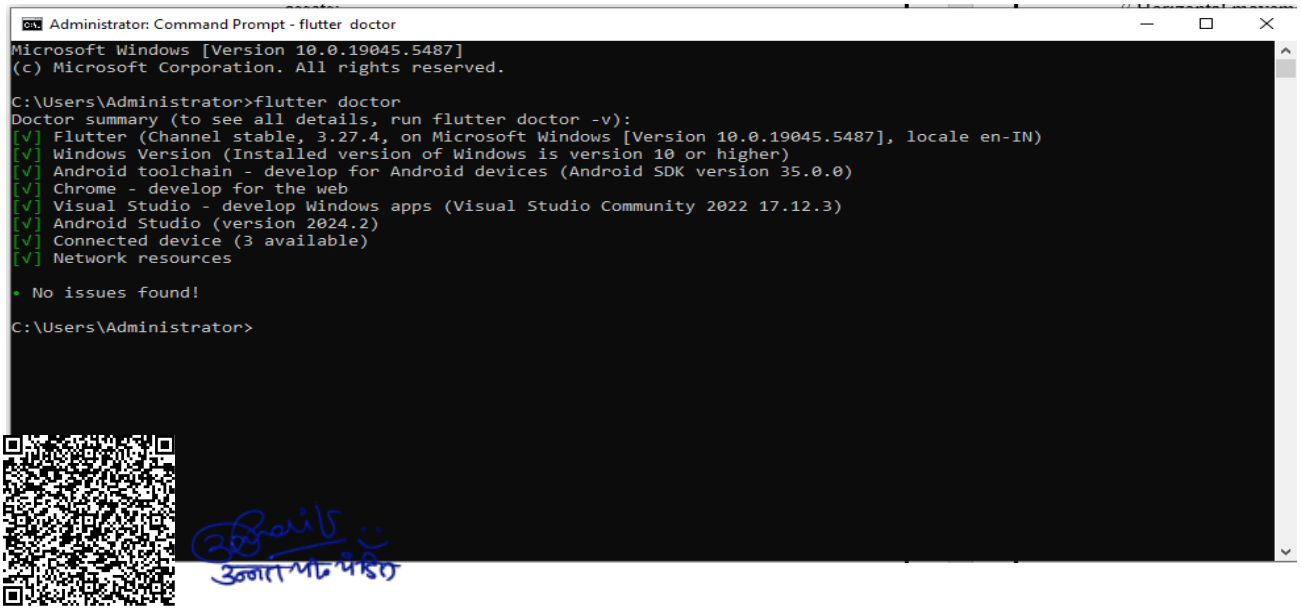
1. **Efficient Rendering** – Handles game loop updates smoothly.
2. **Sprite Animation** – Easily animates the dino character (running, jumping).
3. **Collision Detection** – Detects when the dino hits an obstacle (e.g., cactus).
4. **Parallax Scrolling** – Creates a smooth background effect.
5. **Easy Input Handling** – Detects screen taps for jumping actions.

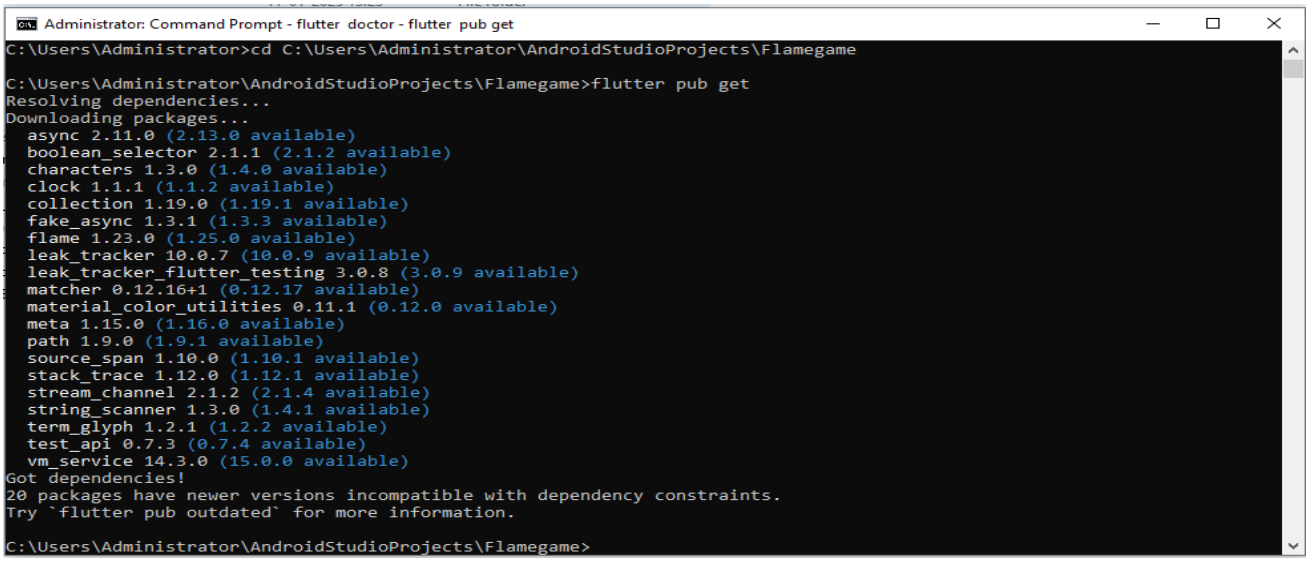
Flame makes building a **Flutter-based 2D Dino game** easier by providing a structured way to manage game objects, physics, and animations.

- **After adding dependencies in the pubspec.yaml or after doing any changes,run the following command to update the file and dependencies. (change the path to the application and then run the command)**

  Command: **flutter doctor** (to check whether flutter is properly installed or not)

  Command: **flutter pub get** (to update the pubspec.yaml)





CODING:

Attacker.dart- This program holds the attackers size , movement and all other characters.


```
import 'package:flame/components.dart';
import 'package:flame/game.dart';
import 'dart:math';
class Attacker extends SpriteComponent with HasGameRef {
  final double speed = 5.0; // Very slow horizontal speed of the attacker
  final double amplitude = 70.0; // Smaller amplitude for vertical movement (up and down)
  double verticalSpeed = 0.5; // Very slow vertical oscillation speed
  double initialY; // Initial Y position
  Attacker(Vector2 position)
      : initialY = position.y,
        super(
        size: Vector2(150.0, 150.0), // Increased size of attackers
        position: position,
      );
  @override
  Future<void> onLoad() async {
    super.onLoad();
    sprite = await gameRef.loadSprite('attacker.png'); // Load attacker sprite
    anchor = Anchor.center;
  }
  @override
  void update(double dt) {
    super.update(dt);
    // Horizontal movement (left to right) at very slow speed
    position.x -= speed * dt; // Move left with very slow speed
    if (position.x < 0) {
      = gameRef.size.x; // Reset position if it goes off-screen
```

```
    // Vertical movement (up and down)
    position.y = initialY + amplitude * sin(verticalSpeed * position.x);
    // Adjust the vertical speed to make the movement slower
    verticalSpeed = 0.5; // Keep vertical speed constant for smooth movement
  }}
```

Dino_game.dart- Which will defines the the gaming page like background, position of attacker and player and other characteristics.

```
---------------------------------------
import 'dart:ui';
import 'package:flame/camera.dart';
import 'package:flame/components.dart';
import 'package:flame/game.dart';
import 'package:flamegame/dino_player.dart';
import 'package:flamegame/dino_world.dart';
import 'package:flamegame/attacker.dart';
import 'package:flamegame/helpers/directions.dart';
class DinoGame extends FlameGame {
  late DinoPlayer _dinoPlayer;
  late DinoWorld _dinoWorld;
  final List<Attacker> attackers = [];
  @override
  Future<void> onLoad() async {
    super.onLoad();
    // Initialize the world and player
    _dinoWorld = DinoWorld();
    _dinoWorld.size = Vector2(2000, size.y); // Set world size wider than the screen
    _dinoWorld.anchor = Anchor.topLeft;
    await add(_dinoWorld);
    _dinoPlayer = DinoPlayer();
    _dinoPlayer.position = Vector2(100, size.y - 150); // Place dino near the bottom
    await add(_dinoPlayer);
    // Add attackers
    spawnAttackers(5);
    // Set up the camera to follow the player
    camera.followComponent(_dinoPlayer,
        worldBounds: Rect.fromLTWH(0, 0, _dinoWorld.size.x, _dinoWorld.size.y));
  }
  @override
  void update(double dt) {
    super.update(dt);
    // Check for collisions with attackers
    for (final attacker in attackers) {
      if (_dinoPlayer.toRect().overlaps(attacker.toRect())) {
rHit();
```

```
    // Update attackers' positions
    for (final attacker in attackers) {
      attacker.update(dt);
    } }
  void onArrowKeyChanged(Direction direction) {
    _dinoPlayer.direction = direction;
  }
  void spawnAttackers(int count) {
    for (int i = 0; i < count; i++) {
      final position = Vector2(
        size.x + (i * 300), // Stagger attackers horizontally
        size.y - 150, // Place near the bottom
      );
      final attacker = Attacker(position);
      attackers.add(attacker);
      add(attacker);
    } }
  void onPlayerHit() {
    print("Player hit by an attacker! Game Over.");
    pauseEngine(); // Stop the game when hit
  }}
extension on CameraComponent {
  void followComponent(DinoPlayer dinoPlayer, {required Rect worldBounds}) {}
}
```


Dino_player.dart -This part of the coding will define the characteristics of dino.

```
import 'package:flame/components.dart';
import 'package:flame/game.dart';
import 'package:flamegame/helpers/directions.dart';
class DinoPlayer extends SpriteComponent with HasGameRef {
  Direction direction = Direction.none;
  final double speed = 200.0;
  DinoPlayer() : super(size: Vector2(100.0, 100.0));
  @override
  Future<void> onLoad() async {
    super.onLoad();
    sprite = await gameRef.loadSprite('Dino1.png');
    anchor = Anchor.center;
  }
  @override
  void update(double dt) {
    super.update(dt);
    // Update position based on direction
    switch (direction) {
      case Direction.up:
        position.y -= speed * dt;
```

```
        break;
      case Direction.down:
        position.y += speed * dt;
        break;
      case Direction.left:
        position.x -= speed * dt;
        break;
      case Direction.right:
        position.x += speed * dt;
        break;
      case Direction.none:
        break;
    }
    // Keep dino within game bounds
    position.clamp(Vector2.zero(), gameRef.size - size);
  }}
```

Dino_world.dart

```
import 'package:flame/components.dart';
class DinoWorld extends SpriteComponent {
  DinoWorld() : super(size: Vector2.all(1000.0)); // Adjust size as needed
  @override
  Future<void> onLoad() async {
    super.onLoad();
    sprite = await Sprite.load('bk.jpg'); // Load the background image
  }}
```

Directions.dart

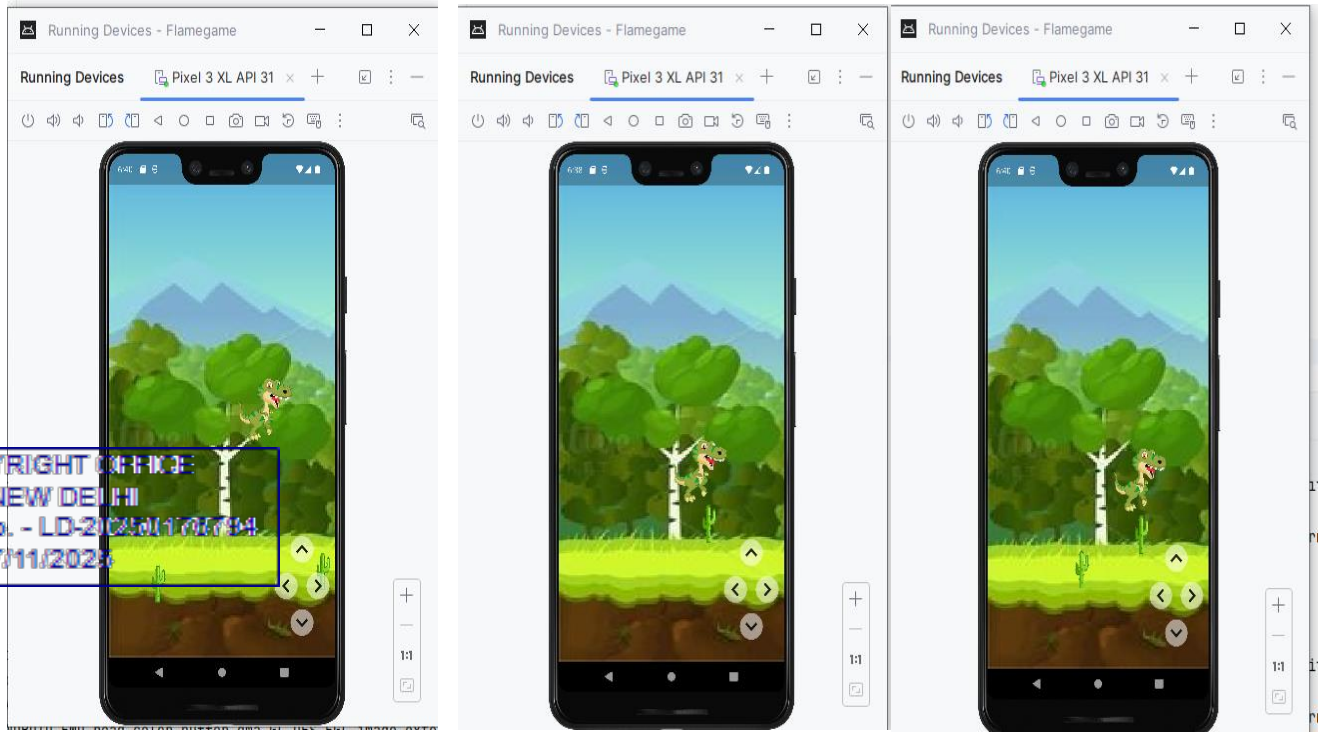
```
// TODO Implement this library.
enum Direction {
  up,  down,  left,  right,  none,}
```

main.dart

```
import 'package:flame/game.dart';
import 'package:flamegame/helpers/navigation_key.dart';
import 'package:flutter/material.dart';
import 'dino_game.dart';
void main() {
  final game = DinoGame();
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        ack(
          n: [
            Widget(
```

```
    game: game,
  ),          Align(
    alignment: Alignment.bottomRight,
    child: NavigationKeys(onDirectionChanged: game.onArrowKeyChanged),
  ),        ],       ),      ),    ),  );}
```

OUTPUT:



RESULT:

This, the 2D Dino game was developed and updated successfully.

VIVA QUESTIONS

1. What is the purpose of the Flame game engine in this project?

Flame is a 2D game engine for Flutter that simplifies game development by providing tools for sprites, animations, collision detection, physics, and user input handling. In this project, it helps manage the game loop, dino character, obstacles, and background rendering efficiently.

2. How does the game handle user gestures to control the dino's movement?

The game uses directions.dart to define movement directions (up, down, left, right). The DinoGame class detects user gestures (e.g., tap or swipe) and updates the DinoPlayer's position accordingly. The onArrowKeyChanged() method processes movement inputs.

3. What is the role of DinoPlayer.dart in this game?

DinoPlayer.dart defines the dino character in the game, including:

- Sprite animation (loading Dino1.png)
- Movement logic (based on user input)
- Game boundary restrictions (preventing the dino from moving off-screen)

4. How is collision detection implemented in this game?

Collision detection is handled in dino_game.dart. The update() method checks if the DinoPlayer's rectangular area overlaps with an Attacker's rectangular area using toRect().overlaps(). If a collision occurs, the game prints "Game Over" and pauses execution using pauseEngine().

e function of Attacker.dart in the game?

defines the enemy obstacles in the game. It manages:

nd position of attackers

- Slow horizontal movement from right to left
- Vertical oscillation (up & down) using sine wave motion
- Sprite loading (attacker.png)

When an attacker reaches the left side of the screen, its position resets.

6. What does the DinoWorld class do in the game?

DinoWorld is responsible for creating the game environment by:

- Loading the background image (bk.jpg)
- Setting the world size and anchor position
- Ensuring the camera follows the dino for a smooth scrolling effect

7. How is the game's camera system handled in DinoGame.dart?

The camera follows the dino using:

camera.followComponent(_dinoPlayer, worldBounds: Rect.fromLTWH(0, 0, _dinoWorld.size.x, _dinoWorld.size.y));

This ensures that as the dino moves forward, the background and attackers appear dynamically, creating an infinite scrolling effect.

8. What is the purpose of the update() method in Flame components?

The update() method runs continuously in the game loop and is used for:

- Moving game objects (dino, attackers)
- Checking for collisions
- Updating the camera position

Each frame, update(dt) ensures smooth animations and interactions between game elements.

9. How does the game generate multiple attackers (obstacles)?

The spawnAttackers(int count) method in DinoGame.dart creates multiple attackers dynamically by:

- Placing them at different x-positions to create staggered obstacles
- Adding them to the game using add(attacker)
- Keeping them moving leftward toward the dino

This ensures a challenging gameplay experience.

10. What steps are taken to update dependencies in pubspec.yaml?

After modifying pubspec.yaml (e.g., adding the flame dependency), the following commands should be run:

```
flutter doctor   # Checks if Flutter is properly installed
flutter pub get  # Fetches and updates dependencies
```

EX.NO:5	DEVELOP A MOVIE RATING APPLICATION

AIM:

Develop a movie rating application using flutter.

PROCEDURE:

- **Flutter Project**
- **Add dependency and Update pubspec.yaml**

Pubspec.yaml

dependecy must be updated

```
---------------------------------------------
name: movie_rating_app
description: "A new Flutter project."
publish_to: 'none'
version: 1.0.0+1
environment:
  sdk: ^3.6.0
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.0.11
cupertino_icons: ^1.0.8
dev_dependencies:
  flutter_test:
    sdk: flutter
 flutter_lints: ^5.0.0
flutter:
    uses-material-design: true
```


shared_preferences is a Flutter package that provides a way to store small amounts of data persistently on a device. It is commonly used for saving user preferences, settings, authentication tokens, and other lightweight data that needs to persist between app sessions.

Key Uses of shared_preferences

1. Storing simple data – Stores key-value pairs (e.g., strings, integers, booleans, lists).
2. Persisting user preferences – Saves theme settings, language preferences, login status, etc.
3. Session management – Stores authentication tokens or user session data.
4. On-device storage – Works like a lightweight local database for simple app data.

open

cmd: // goto flutter path//

cmd: flutter doctor

//goto project path//

cmd:::flutter pub get

- **Write the coding for the application.**

File Format:

lib/

├── helpers/

ng_storage.dart

/

vie.dart

```
├── screens/
│   ├── home_screen.dart
│   └── rating_dialog.dart
├── widgets/
│   └── movie_card.dart
├── main.dart
└── theme.dart (optional)
```

CODING:

Rating_storage.dart

```
-------------------------
import 'package:shared_preferences/shared_preferences.dart';
// Function to save the rating for a movie
Future<void> saveRating(String movieName, double rating) async {
  final prefs = await SharedPreferences.getInstance();
  prefs.setDouble(movieName, rating); // Save the rating with the movie name as key
}

// Function to load the rating for a movie
Future<double?> loadRating(String movieName) async {
  final prefs = await SharedPreferences.getInstance();
  return prefs.getDouble(movieName); // Get the saved rating using the movie name as key
}
```


movie.dart

```
-------------------------
class Movie {
  final String name;
  final String description;
  double rating;
  Movie({
    required this.name,
    required this.description,
    required this.rating,
  });
}
```

home_screen.dart

```
-------------------------
import 'package:flutter/material.dart';
import 'rating_dialog.dart'; // Import RatingDialog
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}
ScreenState extends State<HomeScreen> {
  ovie data with initial ratings
  String, dynamic>> movies = [
```

```
  {
   'title': 'Inception',
   'description': 'A mind-bending thriller.',
   'rating': 4.5,
  },
  {
   'title': 'The Dark Knight',
   'description': 'A superhero film with a dark twist.',
   'rating': 4.8,
  },
  {
   'title': 'Interstellar',
   'description': 'A journey beyond the stars.',
   'rating': 4.7,
  },
];
// Function to handle rating update
void updateRating(BuildContext context, int index, double newRating) {
  setState(() {
   movies[index]['rating'] = newRating; // Update the rating in the list
  });
  print('Updated Rating for ${movies[index]['title']}: $newRating');
}
// Function to show RatingDialog
void showRatingDialog(BuildContext context, int index) {
  showDialog(
    context: context,
    builder: (context) {
      return RatingDialog(
        movieName: movies[index]['title'],
        currentRating: movies[index]['rating'],
        onRatingChanged: (rating) {
          updateRating(context, index, rating); // Update rating when changed
        },       );     },    );  }
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.deepPurple[100], // Colorful background for the page
    appBar: AppBar(
      title: Text('Movie Rating App'),
      backgroundColor: Colors.deepPurpleAccent, // Vibrant app bar color
      centerTitle: true,
    ),
    tView.builder(
      nt: movies.length,
      lder: (context, index) {
```

```
final movie = movies[index];
final movieTitle = movie['title'];
final movieDescription = movie['description'];
final movieRating = movie['rating'];
return Container(
  margin: EdgeInsets.symmetric(vertical: 10, horizontal: 15),
  padding: EdgeInsets.all(10),
  decoration: BoxDecoration(
    color: Colors.white, // Movie card color
    borderRadius: BorderRadius.circular(10),
    boxShadow: [
      BoxShadow(
        color: Colors.black.withOpacity(0.1),
        blurRadius: 5,
        spreadRadius: 3,
      ),
    ],
  ),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        movieTitle,
        style: TextStyle(
          fontSize: 22,
          fontWeight: FontWeight.bold,
          color: Colors.deepPurpleAccent, // Movie title color
        ),
      ),
      SizedBox(height: 8),
      Text(
        movieDescription,
        style: TextStyle(fontSize: 16, color: Colors.grey[700]), // Movie description color
      ),
      SizedBox(height: 8),
      Row(
        children: [
          Text(
            'Rating: ${movieRating.toStringAsFixed(1)}/5',
            style: TextStyle(fontSize: 18, color: Colors.amber), // Rating text color
          ),
          Spacer(),
          GestureDetector(
            onTap: () => showRatingDialog(context, index), // Show dialog on tap
            child: Icon(
              Icons.star,
```

```
                color: Colors.amber,
                size: 30,
              ),              ),              ],              ),              ],          ),          );      },    ),
  );  }}
```

rating_dialog.dart

```
import 'package:flutter/material.dart';
class RatingDialog extends StatefulWidget {
  final String movieName;
  final double currentRating;
  final ValueChanged<double> onRatingChanged;
  RatingDialog({
    required this.movieName,
    required this.currentRating,
    required this.onRatingChanged,
  });
    @override
    RatingDialogState createState() => _RatingDialogState();
}
class _RatingDialogState extends State<RatingDialog> {
  late double rating;
  @override
  void initState() {
    super.initState();
    rating = widget.currentRating; // Initialize rating with the current rating passed
  }
  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      backgroundColor: Colors.deepPurpleAccent, // Set background color for dialog
      title: Text(
        'Rate ${widget.movieName}',
        style: TextStyle(color: Colors.white, fontSize: 24, fontWeight: FontWeight.bold),
      ),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Container(
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(10.0),
              gradient: LinearGradient(
                colors: [Colors.pink, Colors.orangeAccent], // Color gradient for slider track
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
              ),
```

```
            child: Slider(
              value: rating,
              min: 0.0,
              max: 5.0,
              divisions: 5,
              activeColor: Colors.amber, // Slider active color
              inactiveColor: Colors.grey, // Slider inactive color
              label: rating.toStringAsFixed(1),
              onChanged: (newRating) {
                setState(() {
                  rating = newRating; // Update rating dynamically as slider moves
                });
              },
            ),
          ),
          SizedBox(height: 20),
          Text(
            'Current Rating: ${rating.toStringAsFixed(1)}',
            style: TextStyle(
              color: Colors.white,
              fontSize: 18,
              fontWeight: FontWeight.bold,
            ),
          ),
        ],
      ),
      actions: [
        TextButton(
          onPressed: () {
            widget.onRatingChanged(rating); // Pass the new rating back
            Navigator.pop(context); // Close the dialog
          },
          style: TextButton.styleFrom(
            foregroundColor: Colors.white, backgroundColor: Colors.amber, // Button text color
            shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8.0)),
          ),
          child: Text('Submit', style: TextStyle(fontSize: 18)),
        ),      ],    );  }}
```


movie_card.dart

```
-------------------------
import 'package:flutter/material.dart';
import '../models/movie.dart';
ens/rating_dialog.dart';
ard extends StatelessWidget {
 movie;
```

```
  final ValueChanged<double> onRate;
  final double rating;
  MovieCard({
    required this.movie,
    required this.onRate,
    required this.rating,
  });
  @override  Widget build(BuildContext context) {
    return Card(
      margin: EdgeInsets.all(10.0),
      child: ListTile(
        title: Text(movie.name),
        subtitle: Text(movie.description),
        trailing: Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text('Rating: ${rating.toStringAsFixed(1)}/5'),
            IconButton(
              icon: Icon(Icons.star),
              onPressed: () {
                showDialog(
                  context: context,
                  builder: (context) {
                    return RatingDialog(
                      movieName: movie.name,
                      currentRating: rating,
                      onRatingChanged: onRate,
                    );          },          );        },        ),      ,    ),   ),  ); }}
```

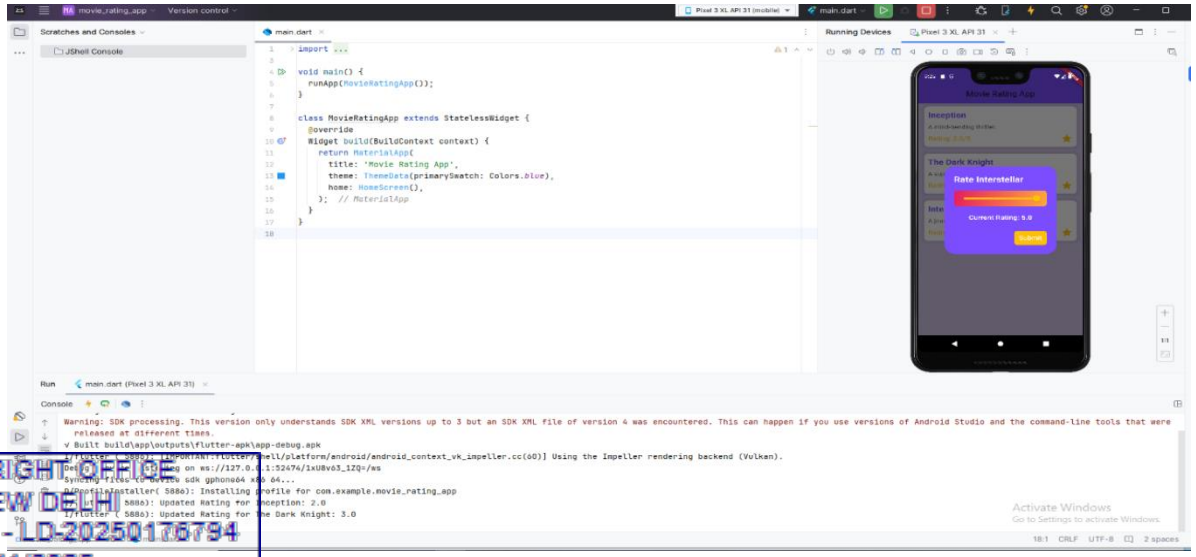
main.dart

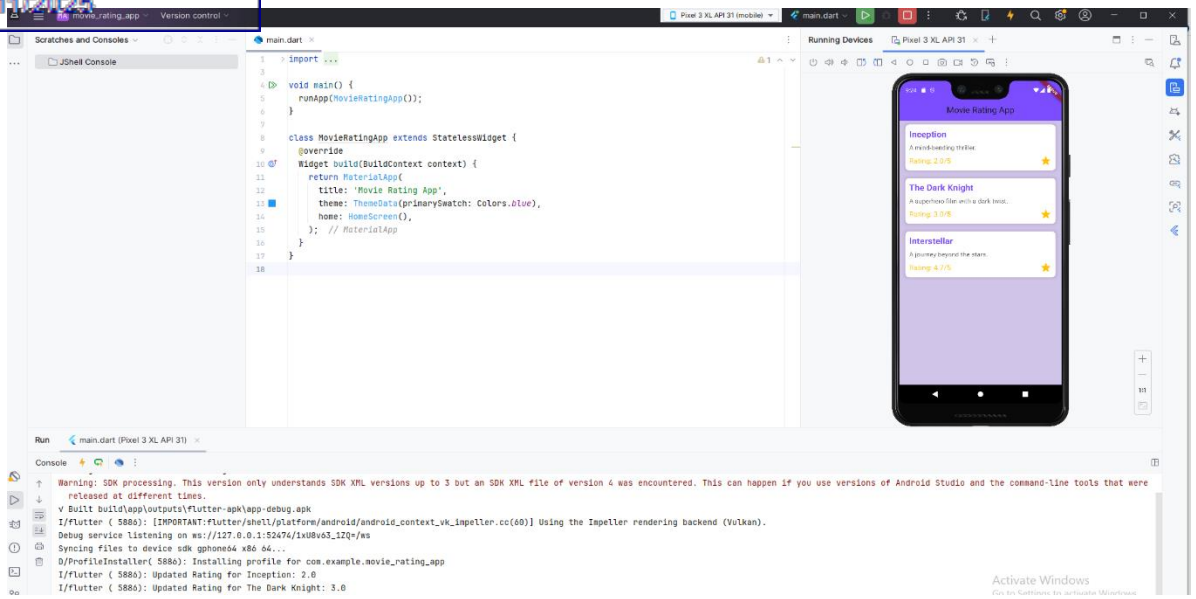
```
import 'package:flutter/material.dart';
import 'screens/home_screen.dart';
void main() {
  runApp(MovieRatingApp());
}
class MovieRatingApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Movie Rating App',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: HomeScreen(),
    );  }}
```

...ge:flutter/material.dart';

```
final ThemeData appTheme = ThemeData(
  primarySwatch: Colors.blue,
  visualDensity: VisualDensity.adaptivePlatformDensity,
);
```

OUTPUT:





RESULT:

Thus, the movie ratting app is created successfully and rating is updated.

VIVA QUESTIONS

1. What is the purpose of the shared_preferences package in this project?

The shared_preferences package is used for persistent storage of small amounts of data on a device. In this project, it is used to store and retrieve movie ratings so that they are saved even after the app is closed and reopened.

2. How does the saveRating function work in rating_storage.dart?

The saveRating function uses the SharedPreferences instance to store a movie rating as a key-value ...vie name is used as the key, and the rating is stored as a double value.

```
... saveRating(String movieName, double rating) async {
```

```
 final prefs = await SharedPreferences.getInstance();
 prefs.setDouble(movieName, rating);}
```

3. How is the movie rating updated dynamically in home_screen.dart?

When a user submits a new rating using the rating dialog, the updateRating function updates the corresponding movie's rating in the movies list and calls setState(), which triggers a UI rebuild.

CopyEdit

```
void updateRating(BuildContext context, int index, double newRating) {
 setState(() {
  movies[index]['rating'] = newRating;
 });}
```

4. What is the role of showRatingDialog in home_screen.dart?

The showRatingDialog function displays a dialog box where users can select a new rating for a movie. It passes the current rating and updates the value when the user submits a new rating.

```
void showRatingDialog(BuildContext context, int index) {
 showDialog(
  context: context,
  builder: (context) {
   return RatingDialog(
    movieName: movies[index]['title'],
    currentRating: movies[index]['rating'],
    onRatingChanged: (rating) {
     updateRating(context, index, rating);
    },
   );   },  );}
```


5. How does rating_dialog.dart enable users to rate movies?

The RatingDialog widget presents a slider that allows users to select a rating between 0.0 and 5.0. When the "Submit" button is pressed, the new rating is passed to the callback function onRatingChanged, which updates the rating in the main movie list.

6. Explain the role of the Movie model in movie.dart.

The Movie model is a class that represents a movie entity with attributes such as name, description, and rating. It helps in structuring movie data efficiently.

```
class Movie {
 final String name;
 final String description;
 double rating;
 Movie({
  required this.name,
  required this.description,
  required this.rating,
 });}
```

7. What is the function of MovieCard in movie_card.dart?

The MovieCard widget is a reusable component that displays movie details, including the title, [illegible]nd rating. It also includes a star button that triggers the rating dialog when clicked.

```
[illegible]ard extends StatelessWidget {
 movie;
```

```
  final ValueChanged<double> onRate;
  final double rating;}
```

8. What is the significance of main.dart in this Flutter project?

The main.dart file serves as the entry point of the Flutter application. It initializes the app and sets HomeScreen as the first screen to be displayed.

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

9. How does the Slider widget in rating_dialog.dart function?

The Slider widget allows users to select a rating value between 0 and 5. The onChanged callback updates the rating variable dynamically as the user moves the slider.

```
Slider(
  value: rating,
  min: 0.0,
  max: 5.0,
  divisions: 5,
  onChanged: (newRating) {
    setState(() {
      rating = newRating;
    });},)
```


10. How does the theme.dart file contribute to the application?

The theme.dart file defines a consistent theme for the application, including the primary color and visual density settings. It helps in maintaining a uniform appearance across the app.

```
final ThemeData appTheme = ThemeData(
  primarySwatch: Colors.blue,
  visualDensity: VisualDensity.adaptivePlatformDensity,
);
```

EX.NO:6	DEVELOP AN APPLICATION TO CONNECT TO A WEB SERVICE AND TO RETRIEVE DATA WITH HTTP

AIM:

To develop an application to connect to a web service and to retrieve data with http

PROCEDURE:

This Flutter application demonstrates how to fetch data from an API using the http package. It retrieves a list of posts from a public API (https://jsonplaceholder.typicode.com/posts) and displays them in a ListView.

Step 1: Create a New Flutter Project

Step 2: Add Dependencies

Step 3: Create api_service.dart to Handle API Requests

Create a new file lib/api_service.dart. This file contains a class ApiService that is responsible for making HTTP requests.

Step 4: Modify main.dart to Display API Data

Step 5: Run the Application

CODING:

Pubspec.yaml


```
----------
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.4
```

api_service.dart

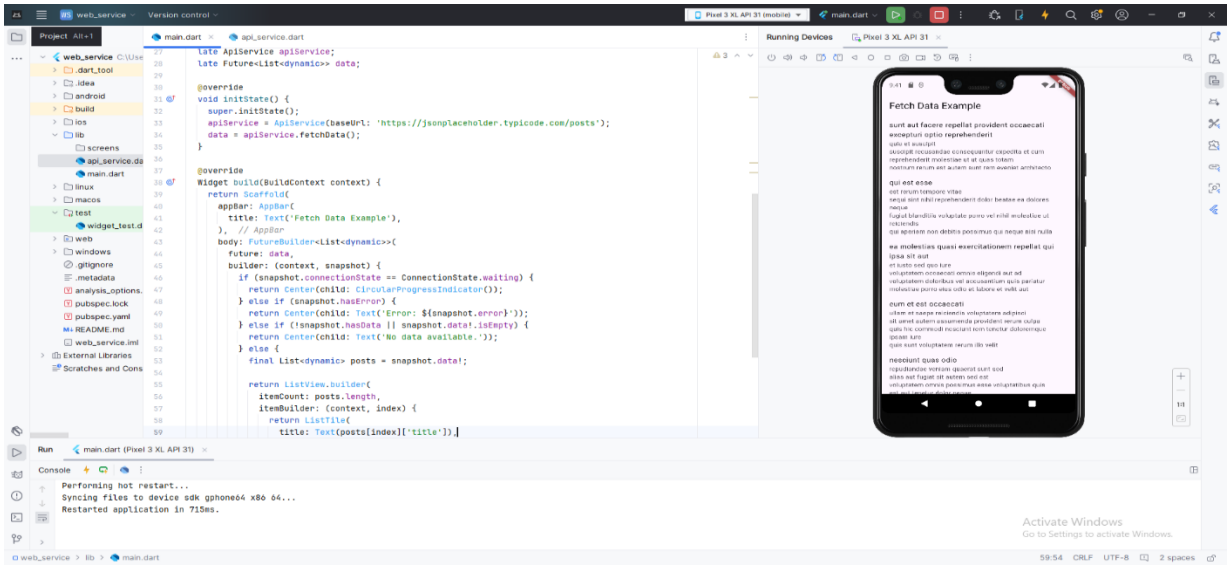
```
---------------------------
import 'dart:convert';
import 'package:http/http.dart' as http;
class ApiService {
final String baseUrl;
ApiService({required this.baseUrl});
// Method to fetch data from the API
Future<List<dynamic>> fetchData() async {
final response = await http.get(Uri.parse(baseUrl));
if (response.statusCode == 200) {
// If the server returns a successful response, parse the JSON
  return json.decode(response.body);
 } else {   // If the server does not return a successful response, throw an error
  throw Exception('Failed to load data');
 } }}
```

main.dart

```
-----------------------
import 'package:flutter/material.dart';
import 'api_service.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter HTTP Example',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),      home: HomePage(),
    );  }}class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}
class _HomePageState extends State<HomePage> {
  late ApiService apiService;
  late Future<List<dynamic>> data;
  @override
  void initState() {
    super.initState();
    apiService = ApiService(baseUrl: 'https://jsonplaceholder.typicode.com/posts');
    data = apiService.fetchData();
  }
  @override
  Widget build(BuildContext context) {    return Scaffold(
      appBar: AppBar(
        title: Text('Fetch Data Example'),
      ),      body: FutureBuilder<List<dynamic>>(
        future: data,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
            return Center(child: Text('No data available.'));
          } else {            final List<dynamic> posts = snapshot.data!;
            return ListView.builder(
              itemCount: posts.length,
              itemBuilder: (context, index) {                return ListTile(
                  title: Text(posts[index]['title']),
                  subtitle: Text(posts[index]['body']),
                );            },          );        }      },    );  }}
```

OUTPUT:



https://jsonplaceholder.typicode.com/posts

RESULT:

Thus, an application developed to connect to a web service and to retrieve data with HTTP executed successfully.

VIVA QUESTIONS

1. What is the purpose of the http package in this Flutter application?

The http package is used to make HTTP requests to fetch data from a web API. In this application, it sends a GET request to https://jsonplaceholder.typicode.com/posts and retrieves JSON data.

2. What does the FutureBuilder widget do in this program?

The FutureBuilder widget is used to asynchronously fetch and display data from a future. It checks the state of the future and updates the UI accordingly—showing a loading indicator, an error message, or the fetched data.

3.Why is initState() used in the HomePage widget?

initState() is used to initialize the ApiService and call fetchData() when the HomePage is first created. This ensures that the API call happens only once when the widget is

4. How does the program handle API errors?

The program handles errors using a try-catch mechanism in the fetchData() method. If the API response status code is not 200, an exception is thrown. The FutureBuilder widget then displays an error message if the future encounters an error.

5. What is the role of the ListView.builder widget?

The ListView.builder widget is used to create a scrollable list of items dynamically. It takes an itemCount (number of posts) and an itemBuilder function to generate each list item efficiently.

6. What would happen if the API URL is incorrect or unreachable?

If the API URL is incorrect or the server is unreachable, the http.get() request will fail, and an exception will be thrown. The FutureBuilder will then display an error message in the UI.

7.Why do we use late for ApiService apiService and Future<List<dynamic>> data?

The late keyword ensures that the variables are initialized before use but after the object is created. It is necessary here because apiService and data are assigned values inside initState() rather than at declaration.

8.What is the purpose of using json.decode(response.body) in fetchData()?

json.decode(response.body) converts the JSON response from the API into a Dart List<dynamic>. This allows the data to be processed and displayed in the Flutter app.

9. How can you modify the program to fetch data from a different API?

To fetch data from a different API:

1. Change the baseUrl in ApiService to the new API endpoint.
2. Update the data model if the JSON structure differs.
3. Modify the UI to match the new data format

10. What improvements can be made to this program?

Some possible improvements include:

- Implementing error handling using try-catch inside fetchData().
- Adding pagination for large datasets.
- Caching API responses to reduce network requests.
- Displaying images if available in the API response.
- Using ChangeNotifier and Provider for better state management.

EX.NO:7	DEVELOP A SIMPLE SHOPPING APPLICATION.

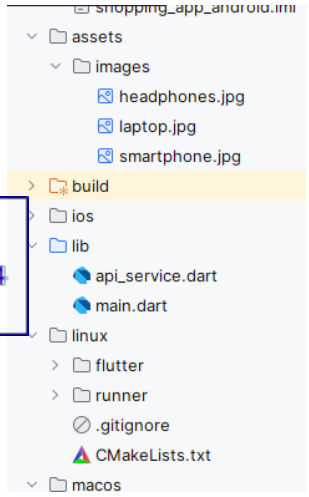
AIM:

To develop a simple shopping application

PROCEDURE:

Step 1: Set Up a New Flutter Project
Step 2: Install Dependencies
Step 3: Add index.html (for Web Version Only)
Step 4: Add Assets (Images for Products)
Step 5: Create api_service.dart for API Handling
Step 6: Create main.dart for the Shopping App
Step 7: Run the Application

File Format:



create index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shopping App</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header>
    <h1>Welcome to Our Shop!</h1>
  </header>
  <div class="products-container">
    <div class="product-card">
      <img src="images/smartphone.jpg" alt="Smartphone">
      <h2>Smartphone</h2>
      <p>$299.99</p>
Add to Cart</button>
="product-card">
```

```
    <img src="images/laptop.jpg" alt="Laptop">
    <h2>Laptop</h2>
    <p>$899.99</p>
    <button>Add to Cart</button>
   </div>
   <div class="product-card">
    <img src="images/headphones.jpg" alt="Headphones">
    <h2>Headphones</h2>
    <p>$49.99</p>
    <button>Add to Cart</button>
   </div>
  </div>
</body>
</html>
```

Adding Images:

---

assets

images

Pubspec.yaml

```
name: shopping_app
description: "A new Flutter project."
publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1
environment:
  sdk: ^3.6.0
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3
  cupertino_icons: ^1.0.8
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
flutter:
  assets:
    - assets/images/
  uses-material-design: true
```

Main.Dart

```
ge:flutter/material.dart';

App());
```

```
}
class Product {
  final String name;
  final String description;
  final String imagePath;
  final double price;
  Product({
    required this.name,
    required this.description,
    required this.imagePath,
    required this.price,
  });}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Shopping App',
      theme: ThemeData(
        primarySwatch: Colors.orange,
        hintColor: Colors.blueAccent,
      ),
      home: HomePage(),
    );  }}
class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}
class _HomePageState extends State<HomePage> {
  final List<Product> products = [
    Product(
      name: "Laptop",
      description: "High-quality sound for an immersive experience.",
      imagePath: "assets/images/headphones.jpg",
      price: 99.99,
    ),
    Product(
      name: "Speakers",
      description: "Powerful laptop with high performance for work and gaming.",
      imagePath: "assets/images/laptop.jpg",
      price: 1299.99,
    ),
    Product(
      name: "Headphones",
      n: "Latest speakers with advanced features.",
      n: "assets/images/smartphone.jpg",
      .99,
```

```
  ),  ];
List<Product> cart = [];
void addToCart(Product product) {
  setState(() {
    cart.add(product);
  });  }
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Shopping App'),
      backgroundColor: Colors.teal,
      actions: [
        IconButton(
          icon: Icon(Icons.shopping_cart),
          onPressed: () {
            // Navigate to the cart page
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => CartPage(cart: cart),
              ),              );            },          ),        ],      ),
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [Colors.purple, Colors.blueAccent],
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
        ),        ),
      child: ListView.builder(
        itemCount: products.length,
        itemBuilder: (context, index) {
          return Card(
            color: Colors.white.withOpacity(0.8),
            margin: EdgeInsets.all(10),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),
            elevation: 5,
            child: ListTile(
              leading: ClipRRect(
                borderRadius: BorderRadius.circular(8),
                child: Image.asset(
                  products[index].imagePath,
                  width: 80,
                  height: 80,
```

```
            fit: BoxFit.cover,
          ),          ),
        title: Text(
          products[index].name,
          style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold),
        ),
        subtitle: Text(
          products[index].description,
          style: TextStyle(color: Colors.black54),
        ),
        trailing: Text(
          '\$${products[index].price}',
          style: TextStyle(color: Colors.green, fontWeight: FontWeight.bold),
        ),
        onTap: () {
          addToCart(products[index]); // Add product to the cart
        },        ),        );      },      ),    );  }}
```



```
class CartPage extends StatefulWidget {
  final List<Product> cart;
  CartPage({required this.cart});
  @override
  _CartPageState createState() => _CartPageState();
}
class _CartPageState extends State<CartPage> {
  late List<Product> cart;
  @override
  void initState() {
    super.initState();
    cart = widget.cart;
  }
  double getTotalPrice() {
    return cart.fold(0, (total, product) => total + product.price);
  }
  void removeItem(int index) {
    setState(() {
      cart.removeAt(index); // Remove the item at the specified index
    });  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Shopping Cart'),
        backgroundColor: Colors.teal,
```

```
t.isEmpty
```

```
        ? Center(child: Text("No items in cart", style: TextStyle(fontSize: 20)))
        : Column(
       children: [
        Expanded(
         child: ListView.builder(
          itemCount: cart.length,
          itemBuilder: (context, index) {
           return Card(
            margin: EdgeInsets.all(10),
            child: ListTile(
             leading: ClipRRect(
              borderRadius: BorderRadius.circular(8),
              child: Image.asset(
               cart[index].imagePath,
               width: 50,
               height: 50,
               fit: BoxFit.cover,
              ),                ),
             title: Text(cart[index].name),
             subtitle: Text("\$${cart[index].price}"),
             trailing: IconButton(
              icon: Icon(Icons.remove_circle),
              color: Colors.red,
              onPressed: () {
               removeItem(index); // Remove item from cart
              },              ),              ),              );          },          ),          ),
        Padding(
         padding: EdgeInsets.all(20),
         child: Text(
          "Total: \$${getTotalPrice().toStringAsFixed(2)}",
          style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
        ),        ),        ],      ),    );  }}
```

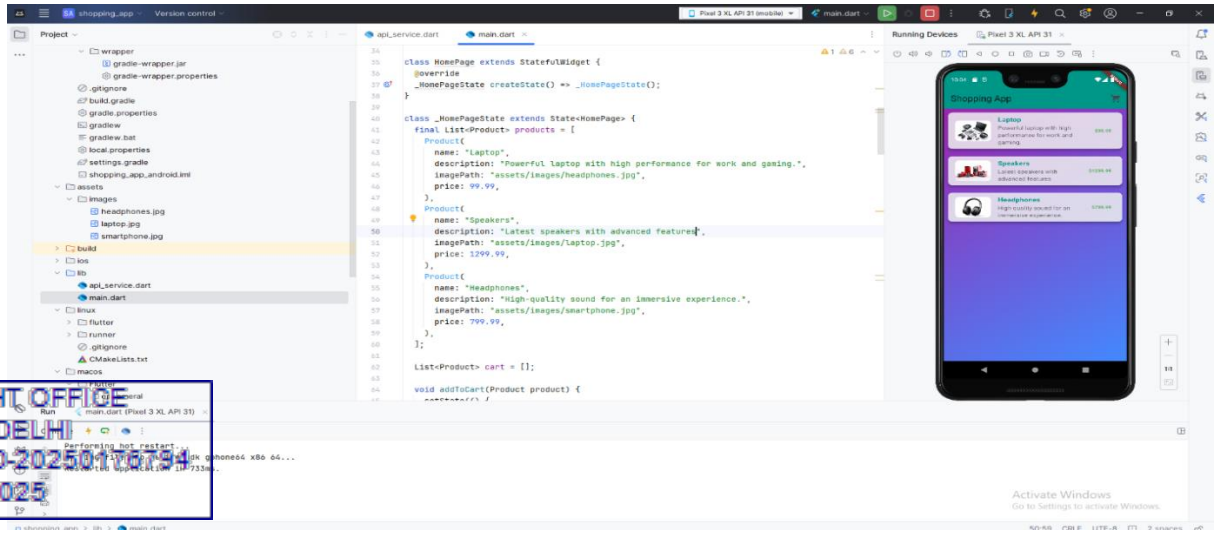

api_services.dart

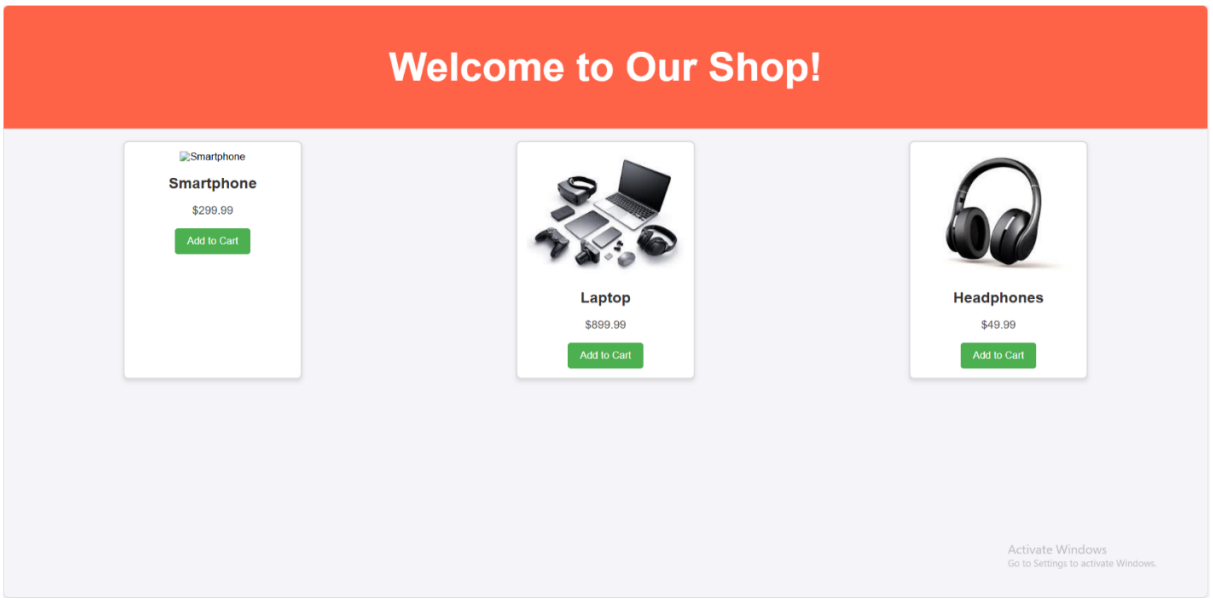
```
---------------------------------------------------------------------
import 'dart:convert';
import 'package:http/http.dart' as http;
// The API service class
class ApiService {
 final String baseUrl;
 // Constructor to initialize the base URL for API requests
 ApiService({required this.baseUrl});
 // Method to fetch data from the API
 Future<List<dynamic>> fetchData() async {
nse = await http.get(Uri.parse(baseUrl));
```

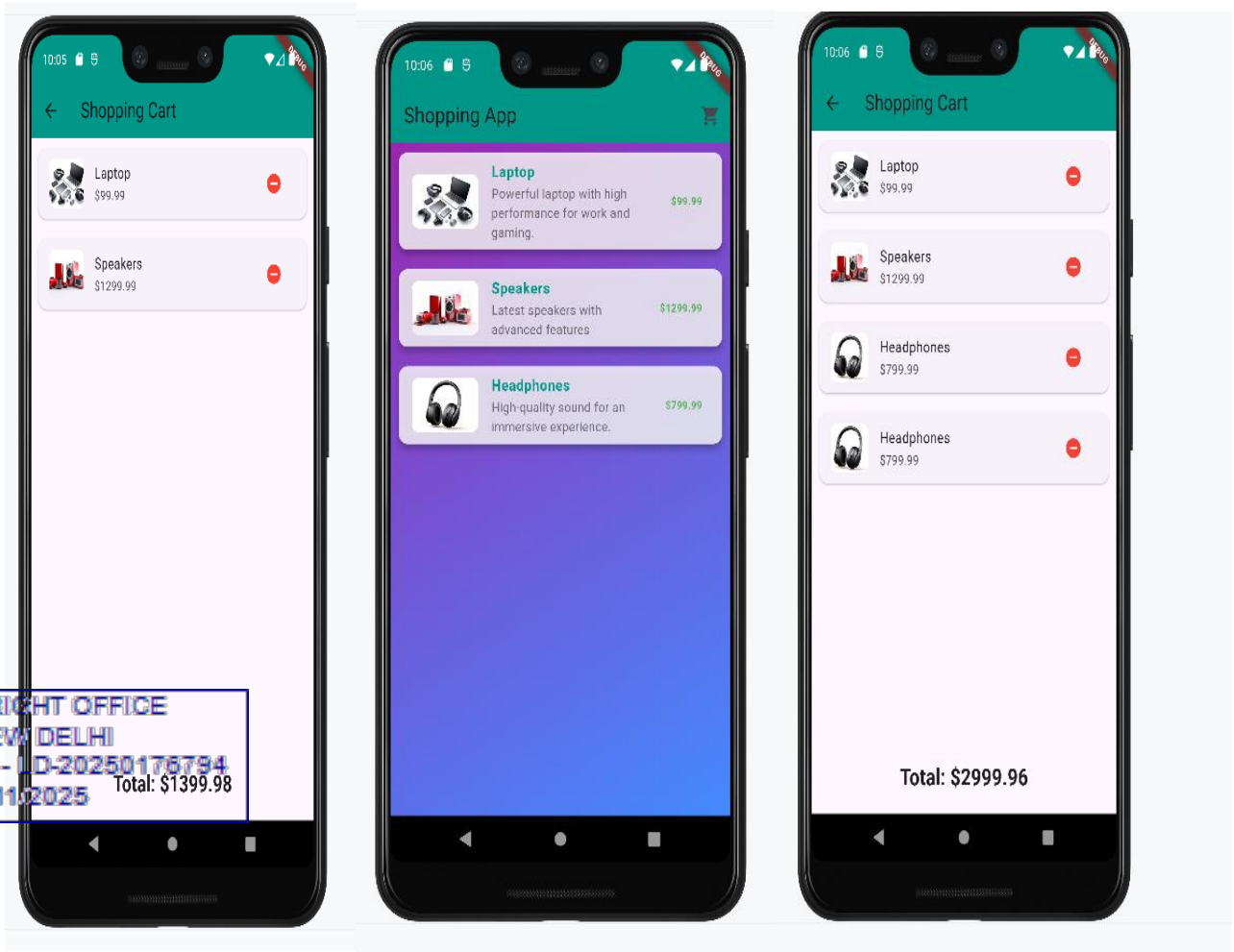
the response status is successful (HTTP status 200)

```
  if (response.statusCode == 200) {
    // If the server returns a successful response, parse the JSON
    return json.decode(response.body);
  } else {
    // If the server does not return a successful response, throw an error
    throw Exception('Failed to load data');
  } }}
```

OUTPUT:







RESULT:

Thus. The shopping application is developed and product information were updated.

VIVA QUESTIONS

1. What is the purpose of this Flutter shopping app?
   This Flutter shopping app allows users to browse products, add them to a shopping cart, and view the total price. It also integrates a simple UI with a product list, cart functionality, and image assets.
2. How does the app store product information?
   The app uses a Product class that defines product attributes like name, description, imagePath, and price. The products are stored in a list inside the HomePage class.
3. How does the app handle adding products to the cart?
   The app maintains a List<Product> named cart. When a user taps on a product, it triggers the addToCart(Product product) function, which adds the product to the list.
4. What is the role of setState() in the app?
   setState() is used to update the UI dynamically when the cart's contents change. It ensures that the UI reflects the latest state of the cart.
5. How are images added and displayed in the app?
   Images are stored in the assets/images/ directory and declared in pubspec.yaml. They are displayed using Image.asset('assets/images/product.jpg') inside ListTile widgets.
6. What happens when a user clicks the shopping cart icon in the app bar?
   When the shopping cart icon is clicked, the app navigates to the CartPage, passing the current cart list as an argument. The CartPage then displays all added products.
7. How is the total price of cart items calculated?
   The getTotalPrice() function uses Dart's fold() method to sum up the prices of all products in the cart.
8. How does the app handle removing items from the cart?
   The removeItem(int index) function is called when the remove button is pressed, using cart.removeAt(index), followed by setState() to update the UI.
9. What is the purpose of ApiService in this project?
   The ApiService class is responsible for fetching product data from an external API using an HTTP GET request. It processes JSON responses and returns a list of data.
10. How can this app be improved further?
    The app can be improved by adding features like user authentication, backend integration for real-time product updates, checkout/payment functionality, and product filtering.

EX.NO:8	DESIGN A WEB SERVER SUPPORTING PUSH NOTIFICATIONS

AIM:

To develop Design a web server supporting push notifications.

PROCEDURE:

Step 1: Create a New Flutter Project

Step 2: Add Dependencies in pubspec.yaml

Step 3: Add Assets (Icons for Notifications)

1.Create an assets/ folder inside the project root.

2.Place the following image files inside assets/:

amazon_icon.png

amazon_notification_icon.png

icon.png

3.Register the assets in pubspec.yaml:

Step 4: Implement WebSocket and Notification Service

Step 5: Run the Application

Step 6: Test WebSocket Notifications

- Click **Go to Sender** and enter a message.
- Click **Send Notification**, and it will be sent via WebSocket.
- Open the **Receiver Screen**, and a local push notification should appear.

CODING:

Folder Format:

assets

-amazon_icon.png

-amazon_notofication_icon.png

-icon.png

lib

-main.dart

-notification_service.dart

-receiver_screen.dart

-sender_screen.dart

---

pubsec.yaml

```
name: flutter_websocket_push
description: "A new Flutter project."
publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1
environment:
  sdk: ^3.6.0
dependencies:
  flutter:
    sdk: flutter
  web_socket_channel: ^3.0.2
  flutter_launcher_icons: ^1.0.8
  flutter_local_notifications: ^18.0.1
```

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
flutter:
  uses-material-design: true
  assets:
    - assets/amazon_icon.png
    - assets/amazon_notification_icon.png
  ------------------------------------------------------------
```

main.dart

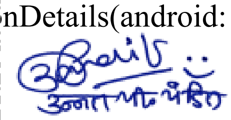
```
------------------------------------------------------------
import 'package:flutter/material.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
import 'notification_service.dart';
import 'sender_screen.dart';
import 'receiver_screen.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await NotificationService.init();
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  final WebSocketChannel channel =
  WebSocketChannel.connect(Uri.parse("ws://echo.websocket.org"));

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
        scaffoldBackgroundColor: Colors.purple.shade50,
      ),
      home: HomeScreen(channel: channel),
    ); }}
class HomeScreen extends StatelessWidget {
  final WebSocketChannel channel;
  HomeScreen({required this.channel});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Push Notification Demo")),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

```
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => SenderScreen(channel: channel)),
                );            },
              child: Text("Go to Sender"),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => ReceiverScreen(channel: channel)),
                );            },            child: Text("Go to Receiver"),
            ),      ],      ),     ),    );  }}
```


----

notification_service.dart

```
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
class NotificationService {
  static final FlutterLocalNotificationsPlugin _notificationsPlugin =
  FlutterLocalNotificationsPlugin();
  static Future<void> init() async {
    const AndroidInitializationSettings androidSettings =
    AndroidInitializationSettings('@mipmap/ic_launcher');
    const InitializationSettings settings =
    InitializationSettings(android: androidSettings);
    await _notificationsPlugin.initialize(settings);
  }
  static Future<void> showNotification(String message) async {
    const AndroidNotificationDetails androidDetails =
    AndroidNotificationDetails(
      'channel_id',
      'Push Notifications',
      importance: Importance.max,
      priority: Priority.high,
      showWhen: true,
      largeIcon: DrawableResourceAndroidBitmap("amazon_notification_icon"), // Use Amazon icon
      icon: 'amazon_notification_icon', // Amazon icon for push notifications
    );
                ficationDetails platformDetails =
                nDetails(android: androidDetails);
```



```
    await _notificationsPlugin.show(
        0, 'Amazon Update', message, platformDetails);
  }}
```

---

receiver_screen.dart

```
import 'package:flutter/material.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
import 'notification_service.dart';
class ReceiverScreen extends StatefulWidget {
  final WebSocketChannel channel;
  ReceiverScreen({required this.channel});
  @override
  _ReceiverScreenState createState() => _ReceiverScreenState();
}
class _ReceiverScreenState extends State<ReceiverScreen> {
  @override
  void initState() {
    super.initState();
    widget.channel.stream.listen((message) {
      NotificationService.showNotification(message);
    });  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Receive Notifications")),
      body: Center(
        child: Text(
          "Waiting for push notification...",
          style: TextStyle(fontSize: 18, color: Colors.deepPurple),
        ),      ),    );  }}
```

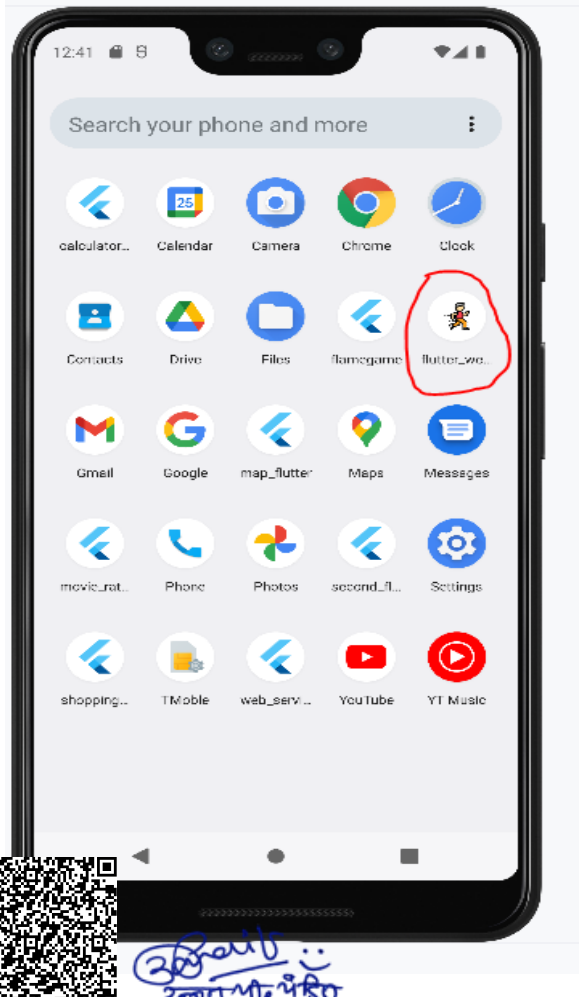

---

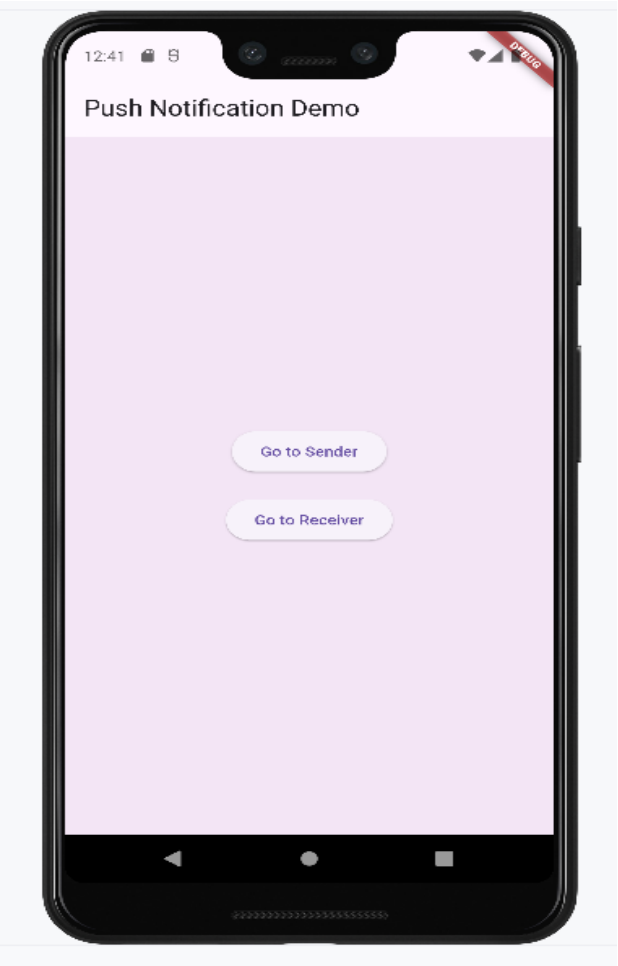
sender_screen.dart

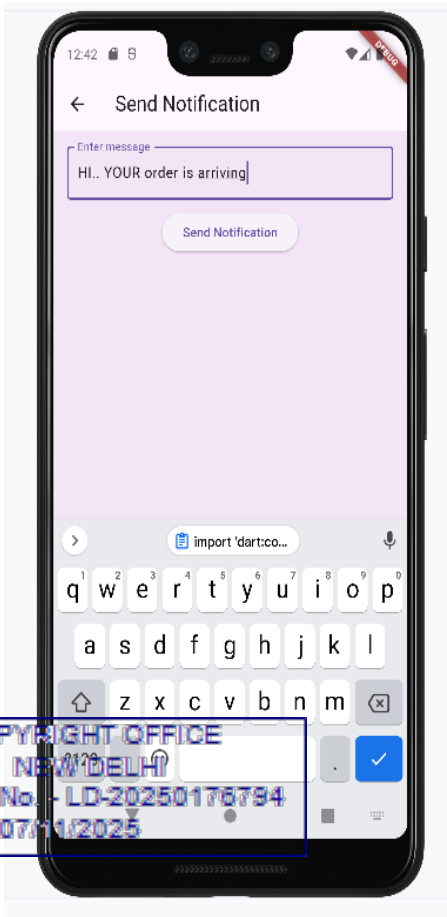
```
import 'package:flutter/material.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
class SenderScreen extends StatefulWidget {
  final WebSocketChannel channel;
  SenderScreen({required this.channel});
  @override
  _SenderScreenState createState() => _SenderScreenState();
}
class _SenderScreenState extends State<SenderScreen> {
  TextEditingController _controller = TextEditingController();
  void _sendMessage() {
    if (_controller.text.isNotEmpty) {
```

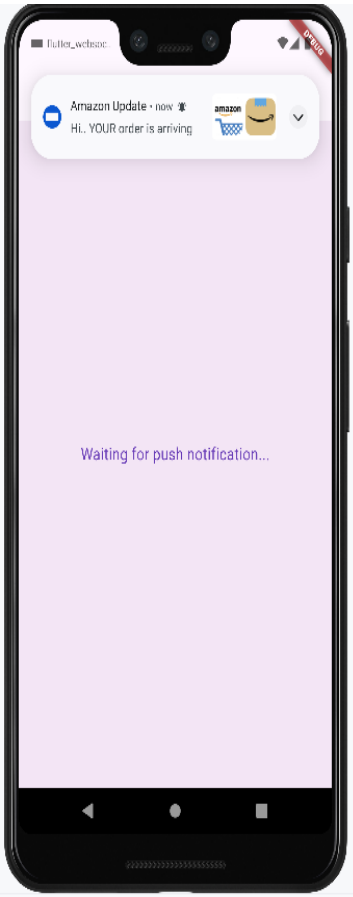
```
    widget.channel.sink.add(_controller.text);
    _controller.clear();
  } }
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("Send Notification")),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          TextField(
            controller: _controller,
            decoration: InputDecoration(
              labelText: "Enter message",
              border: OutlineInputBorder(),
            ),          ),
          SizedBox(height: 10),
          ElevatedButton(
            onPressed: _sendMessage,
            child: Text("Send Notification"),
          ),        ],       ),     ),    ); }}
```

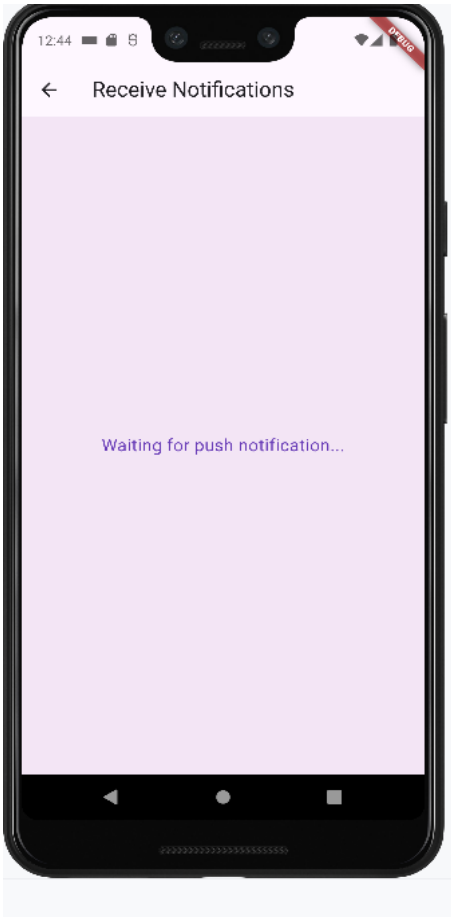

OUTPUT:











RESULT:

Thus, the push notification is checked from server side to client side.

VIVA QUESTIONS

1.What is the purpose of this project?

This project demonstrates real-time push notifications in a Flutter app using WebSockets. A sender sends messages, and a receiver listens for messages and displays notifications.

2.What is WebSocket, and why is it used here?

WebSocket is a communication protocol that enables real-time, bidirectional communication between a client and a server. It is used here to send and receive push notifications instantly.

3.Which dependencies are used in this project?

The project uses:
web_socket_channel: For WebSocket communication
flutter_local_notifications: For handling push notifications
cupertino_icons: For UI enhancements

4.What is the role of notification_service.dart**?**

It initializes the notification service and provides a method to display push notifications when a message is received.

5.How does receiver_screen.dart **handle incoming messages?**

It listens to the WebSocket stream and triggers a local notification using NotificationService.showNotification(message) when a message is received.

6.How do you establish a WebSocket connection in Flutter?

Using WebSocketChannel.connect(Uri.parse("ws://echo.websocket.org")), which creates a WebSocket channel.

7.How do you send a message using WebSocket?

In sender_screen.dart, the _sendMessage() function sends a message using widget.channel.sink.add(_controller.text);.

8.What happens if the WebSocket connection fails?

If the connection fails, the app will not be able to send or receive messages. To handle reconnections, error handling and retries should be implemented.

9.What is the function of flutter_local_notifications **in this project?**

It is used to display local notifications when a message is received via WebSocket.

10.What is the role of AndroidInitializationSettings **in the notification service?**

It initializes the local notifications on Android, specifying an icon (@mipmap/ic_launcher) to be used.

11.Why is WidgetsFlutterBinding.ensureInitialized() used in main.dart?

It ensures that Flutter is fully initialized before running asynchronous operations like NotificationService.init();.

12.What modifications are required to make this work on iOS?

Additional setup in Info.plist is needed, along with specific iOS notification permissions and configurations.

13.How can you change the WebSocket URL to connect to a real server?

Update WebSocketChannel.connect(Uri.parse("ws://your_server_address")) with the actual WebSocket server URL.

14.How does the receiver differentiate between different types of notifications?Currently, all have the same title ("Amazon Update"). To differentiate, custom logic can be added to ent message types.

EXP.NO: 9	DEVELOP AN APPLICATION BY INTEGRATING GOOGLE MAPS

AIM:

To develop an application by integrating google maps.

PROCEDURE:

Step 1: Create a New Flutter Project
Step 2: Add Dependencies
Step 3: Modify main.dart
Step 4: Implement map_screen.dart
Step 5: Modify Android Permissions
Step 6: Update Android Minimum SDK Version
Step 7: Modify iOS Permissions
Step 8: Run the Application

Folder Format:

lib
screens-map_screen.dart
main.dart

pubsec.yaml
----------name: map_flutter
description: "A new Flutter project."

publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1
environment:
  sdk: ^3.6.2
dependencies:
  flutter:
    sdk: flutter
  flutter_map: ^6.1.0 # OSM Map package
  latlong2: ^0.9.0 # For handling coordinates
  geolocator: ^11.0.0 # Get user location
  permission_handler: ^11.0.1 # Handle location permissions

  cupertino_icons: ^1.0.8
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0
flutter:
  uses-material-design: true

```
import 'package:flutter/material.dart';
import 'package:map_flutter/screens/map_screen.dart';
import 'map_screen.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   home: MapScreen(),
  );  }}
```

---

map_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_map/flutter_map.dart';
import 'package:latlong2/latlong.dart';
import 'package:geolocator/geolocator.dart';
import 'package:permission_handler/permission_handler.dart';

class MapScreen extends StatefulWidget {
 @override
 _MapScreenState createState() => _MapScreenState();
}
class _MapScreenState extends State<MapScreen> {
 LatLng _currentPosition = LatLng(0, 0);
 bool _isLocationFetched = false;
 @override
 void initState() {
  super.initState();
  _getUserLocation();
 }
 // Function to get user's location
 Future<void> _getUserLocation() async {
  var status = await Permission.location.request();
  if (status.isGranted) {
   Position position = await Geolocator.getCurrentPosition(
    desiredAccuracy: LocationAccuracy.best,  // Highest accuracy
   );
   print("Latitude: ${position.latitude}, Longitude: ${position.longitude}"); // Debugging
   setState(() {
    Position = LatLng(position.latitude, position.longitude);
    ionFetched = true;
```

```
  } else {
    print("Location permission denied.");
  } }
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("My Location")),
    body: _isLocationFetched
        ? FlutterMap(
      options: MapOptions(
        initialCenter: _currentPosition,
        initialZoom: 15,
      ),
      children: [
        TileLayer(
          urlTemplate: "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
          subdomains: ['a', 'b', 'c'],
        ),
        MarkerLayer(
          markers: [
            Marker(
              point: _currentPosition,
              width: 40,
              height: 40,
              child: Icon(Icons.location_pin, color: Colors.red, size: 40),
            ),          ],          ),       ],                          )
        : Center(child: CircularProgressIndicator()),
  );  }}
```


4. AndroidManifest.xml (Android Permissions)

Modify android/app/src/main/AndroidManifest.xml:

Add these permissions before <application>:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

Inside the <application> tag, add:

```
<service
   android:name="com.baseflow.geolocator.GeolocatorLocationService"
   android:permission="android.permission.BIND_JOB_SERVICE"
   android:foregroundServiceType="location" />
```

5. android/app/build.gradle (Min SDK Version)

Modify android/app/build.gradle to ensure compatibility:

{

```
    minSdkVersion 21
}
```

If targeting Android 12+, add this in android/gradle.properties:

ini

```
android.useAndroidX=true
android.enableJetifier=true
```

6. ios/Runner/Info.plist (iOS Permissions)

If you're targeting iOS, modify ios/Runner/Info.plist:

xml

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>We need your location to show your position on the map</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>We need your location to improve the map experience</string>
```

Final Steps

Run:

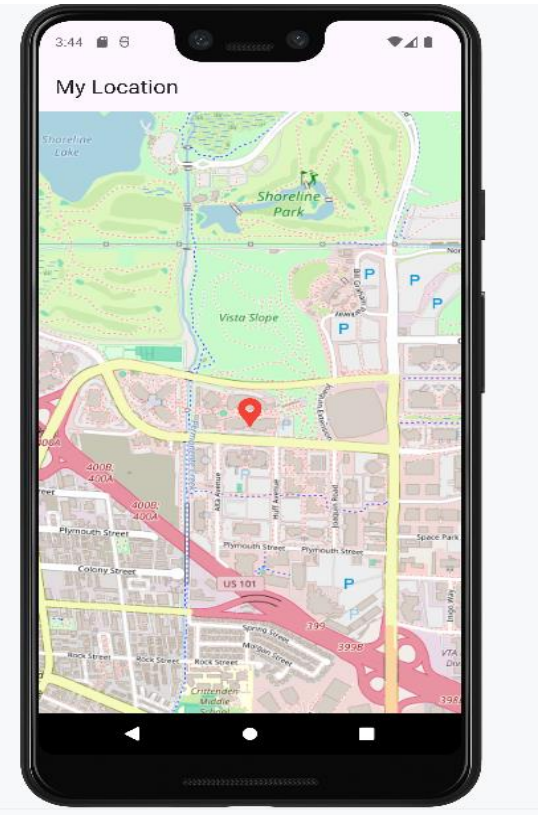
sh

```
flutter clean
flutter pub get
flutter run
```

If testing on an emulator, enable location manually.

If testing on a real device, ensure location services are enabled.

OUTPUT:



RESULT:

Google map integrated with the Emulator.

VIVA QUESTIONS

1. What is the purpose of the flutter_map **package in this project?**

The flutter_map package is used to integrate OpenStreetMap (OSM) into the Flutter application. It provides a customizable map widget and supports features like tile layers, markers, and interactivity.

2. Why do we use the latlong2 **package in this project?**

The latlong2 package is used to handle geographical coordinates, such as latitude and longitude. It provides the LatLng class, which is essential for defining positions on the map

3. How does the app request and handle location permissions?

The app uses the permission_handler package to request location permissions. It calls Permission.location.request(), and if the permission is granted, it fetches the user's current location using the geolocator package

4. How does the app fetch the user's current location?

The app uses the Geolocator.getCurrentPosition() method with LocationAccuracy.best to get the precise location of the user. The fetched latitude and longitude are then stored in the _currentPosition variable.

5. Why do we use setState() **inside** _getUserLocation()**?**

setState() is used to update the UI when the user's location is fetched. It ensures that the new coordinates are reflected in the map widget by triggering a rebuild of the MapScreen.

6. What happens if the user denies location permission?

If the user denies location permission, the app prints "Location permission denied." in the console and does not fetch the user's location. The map will not be centered on the user's position.

7. What is the role of TileLayer **in** FlutterMap**?**

TileLayer is responsible for rendering the map tiles from a URL source. In this project, it fetches tiles from OpenStreetMap using the template "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png".

8. How does the app display the user's current location on the map?

The app uses the MarkerLayer to place a Marker at _currentPosition. The marker is represented by a red location pin icon (Icons.location_pin).

9. What changes are required in AndroidManifest.xml **to enable location access?**

The following permissions must be added before the <application> tag:

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/><uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/><uses-permission android:name="android.permission.FOREGROUND_SERVICE"/><uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>

Additionally, a service must be declared inside <application>:

```
<service
    android:name="com.baseflow.geolocator.GeolocatorLocationService"
    android:permission="android.permission.BIND_JOB_SERVICE"
    android:foregroundServiceType="location" />
```

10. What steps should be taken to test this application on a real device?

Ensure location services are enabled on the device.
...l the app on the device using flutter run.
... location permission when prompted.
...ve the map displaying the user's current location with a marker.

EXP.NO: 10	LIST OF MINI PROJECTS

S.NO	PROBLEM STATEMENTS
1.	Personal Finance Tracker-Develop a mobile application to track users' income, expenses, and savings with detailed analytics and category-wise expenditure insights. Users should be able to set monthly budgets and receive alerts when they exceed their limits. The app should support exporting reports in PDF and CSV formats.
2.	**To-Do List App** – Create a task manager where users can add, edit, and delete tasks. Implement categories and due dates for better organization. Provide notifications to remind users of pending tasks.
3.	**Unit Converter** – Develop an app that converts units for length, weight, and temperature. Users should be able to select input and output units. Provide an easy-to-use UI with instant conversion.
4	**Currency Converter** – Fetch real-time exchange rates and convert currencies. Allow users to select different currencies from a dropdown menu. Display historical exchange rate trends for reference.
5.	**Weather App** – Fetch real-time weather updates using an API. Display temperature, humidity, and wind speed based on the user's location. Include a five-day forecast feature with icons and animations.
6.	**Calculator** – Implement a calculator with basic arithmetic and scientific operations. Support history tracking to store previous calculations. Ensure a clean and responsive user interface.
7.	**BMI Calculator** – Let users enter weight and height to calculate BMI. Show results with categorization (underweight, normal, overweight, obese). Provide health recommendations based on the BMI range.
8.	**Flashlight App** – Use the device's flashlight for a simple torch feature. Provide an interface to switch between normal and strobe light modes. Include a brightness adjustment option.
9.	**QR Code Scanner** – Implement a QR scanner using the phone's camera. Allow users to scan QR codes and extract the information. Provide a history log of scanned QR codes for future reference.
10.	**Stopwatch & Timer** – Create an app that features a countdown timer and a stopwatch. Allow users to record lap times in the stopwatch mode. Enable alarm notifications when the countdown timer reaches zero.
11.	**Expense Tracker** – Develop an app where users can log their daily expenses. Categorize expenses (food, travel, shopping) and generate reports. Display monthly spending trends with a graphical interface.
12.	**Chat App** – Build a real-time messaging app using Firebase. Enable user authentication and display online/offline status. Support sending text messages, images, and emojis.
	Voice Recorder – Record audio and save it in the local storage. Allow playback, pause, and delete functionalities. Provide options for different audio formats and quality settings.

14.	**Event Reminder** – Let users create event reminders with date and time. Send push notifications as reminders for upcoming events. Allow recurring event creation for daily, weekly, or monthly reminders.
15.	**Emergency Contact App** – Store emergency contacts with quick dial functionality. Allow sending an SOS message with the user's live location. Provide an option to activate emergency mode with one tap.
16.	**Journal App** – A digital diary where users can write and save daily entries. Secure entries with a password or biometric authentication. Allow users to attach images or voice notes to entries.
17.	**Flashcards App** – Help users memorize concepts using digital flashcards. Allow adding text, images, and audio to flashcards. Implement a spaced repetition system for better learning.
18.	**Quiz App** – A quiz application with multiple-choice questions. Provide instant feedback and a scoring system. Allow users to review correct and incorrect answers after completion.
19.	**Dictionary App** – Allow users to search for word meanings and pronunciations. Fetch definitions from an API or local database. Include a "word of the day" feature for learning new words.
20.	**Language Learning App** – Offer lessons for learning basic phrases in different languages. Include pronunciation assistance with audio playback. Track user progress and unlock new levels upon completion
21.	**Step Counter** – Track the number of steps taken using phone sensors. Display daily, weekly, and monthly step count statistics. Provide calorie burn estimates based on user activity.
22.	**Water Reminder** – Notify users to drink water at regular intervals. Customize notifications based on the user's weight and activity level. Log daily water intake and provide hydration insights.
23.	**Sleep Tracker** – Log sleep duration and analyze sleep patterns. Allow users to set sleep goals and bedtime reminders. Generate sleep quality reports based on movement and sound analysis.
24.	**Meditation App** – Provide guided meditation sessions with voice instructions. Include soothing background sounds for relaxation. Allow users to set daily meditation goals and track progress.
25.	**Yoga Pose Guide** – Display different yoga poses with step-by-step instructions. Include pose difficulty levels and health benefits. Allow users to create personalized yoga routines.
26.	**Music Player** – Build an offline music player supporting different formats. Implement a playlist feature and an equalizer. Allow users to change playback speed and set sleep timers.
	me Generator – Let users create memes using predefined templates. Allow om text input and font selection. Provide social media sharing options for erated memes.

28.	**Photo Editor** – Implement basic image editing features like crop, filter, and rotate. Support adding text, stickers, and drawing tools. Allow users to save and share edited images.
29.	**Wallpaper App** – Let users browse and download high-quality wallpapers stored locally. Categorize wallpapers based on themes and resolutions. Enable the option to set wallpapers directly from the app.
30.	**Grocery List App** – Allow users to create and manage shopping lists. Provide a checklist feature for marking purchased items. Include a barcode scanner for quick item addition.
31.	**Invoice Generator** – Generate and store invoices for small businesses. Allow users to input customer details, items, and tax rates. Provide PDF export and email-sharing options.
32.	**Digital Visiting Card** – Create and share digital business cards as images. Allow customization with logos, social links, and contact details. Store received business cards in the app for easy access.
33.	**Notebook App** – A simple note-taking app that saves text notes locally. Allow users to categorize notes and search for them easily. Provide a dark mode option for a better user experience.
34.	**Tic-Tac-Toe** – Create a simple two-player Tic-Tac-Toe game. Implement AI-based gameplay for single-player mode. Ensure an engaging UI with animations.
35.	**Sudoku Solver** – Generate and solve Sudoku puzzles. Allow users to enter their own puzzles for solving. Include different difficulty levels and hint functionality.
36.	**Guess the Word** – Display hints and let users guess the correct word. Track scores and provide feedback for wrong answers. Implement difficulty levels to increase engagement.
37.	**Rock-Paper-Scissors** – Implement the classic game against an AI opponent. Allow users to play against the computer with animated gestures. Display a scoreboard tracking user and AI wins.
38.	**Math Puzzle Game** – Provide math problems for players to solve within a time limit. Include addition, subtraction, multiplication, and division questions. Track high scores and time taken for solving puzzles.
39.	**Fake Call App** – Simulate an incoming call for escaping awkward situations. Allow users to set caller name and ringtone. Provide scheduling options for the fake call.
40.	**Name Generator** – Generate random names for characters, pets, or projects. Allow customization based on gender and culture. Provide an option to save favorite names.
41.	**Daily Motivation App** – Show motivational quotes every morning. Allow users to save and share their favorite quotes. Implement a daily notification feature.
42.	**Sound Level Meter** – Measure ambient noise levels using the phone's microphone. Display noise levels in decibels with a graphical meter. Provide warnings if noise levels are too high.
43.	**Book Tracker** – Keep track of books read, with ratings and notes. Allow users to create a wishlist of books to read. Generate reading statistics like books completed per month.
	Calculator – Calculate tips based on the total bill amount and percentage. Provide split bill feature to divide the total among multiple people. Ensure a simple and easy-to-use UI.

45.	**Interest Calculator** – Calculate simple and compound interest based on user input. Allow users to select different interest rates and time periods. Display a detailed breakdown of interest calculations.
46.	**Password Generator** – Generate strong passwords with customizable options. Allow users to select password length, symbols, and numbers. Provide an option to copy the generated password to the clipboard.
47.	**Pomodoro Timer** – Implement a Pomodoro productivity timer. Allow users to set work and break intervals. Display session statistics and notify users when time is up.
48.	**Voice Notes App** – Record voice notes and store them locally. Allow users to play, rename, and delete recordings. Provide an option to share audio files via messaging apps.
49.	**Healthy Recipe App** – Provide a list of healthy food recipes with ingredients and steps. Allow users to filter recipes based on diet preferences (vegetarian, keto, high protein). Provide a shopping list feature.
50.	**Catch the Falling Object Game** – A simple game where the player moves a basket to catch falling objects. Provide different difficulty levels and increasing speeds. Track high scores.
51.	**Loan EMI Calculator** – Calculate monthly loan payments based on the loan amount, interest rate, and tenure. Display the total interest paid over time. Provide a graphical representation of loan repayment.
52.	**Stock Portfolio Tracker** – Allow users to manually enter stock details and track portfolio performance. Display gains and losses with a simple chart. Provide a history log for past transactions.
53.	**Ringtone Maker** – Let users trim and cut audio files to create custom ringtones. Provide a waveform visualization for better precision. Allow saving and playing edited ringtones.
54.	**Horoscope App** – Show daily horoscopes for different zodiac signs. Allow users to select their zodiac sign from a list. Display predictions related to career, love, and health.
55.	**Random Joke Generator** – Display a new joke every time the user taps a button. Store jokes in a local database or JSON file. Provide a "Copy" or "Share" button for easy sharing.
56.	**Color Palette Generator** – Generate random color palettes for designers. Allow users to save their favorite palettes. Display color codes in HEX and RGB formats.
57.	**Video Streaming App** – Stream online videos using YouTube or other APIs. Provide search functionality to find specific videos. Enable full screen mode and video quality selection.
58.	**Day Counter** – Let users enter two dates and calculate the difference in days, months, and years. Useful for event countdowns and project deadlines. Provide an option to save important date calculations.