

Experiment 1

Aim : Hello World Application: Create a basic "Hello World" application for a mobile platform of your choice (Android or iOS) using the respective development environment.

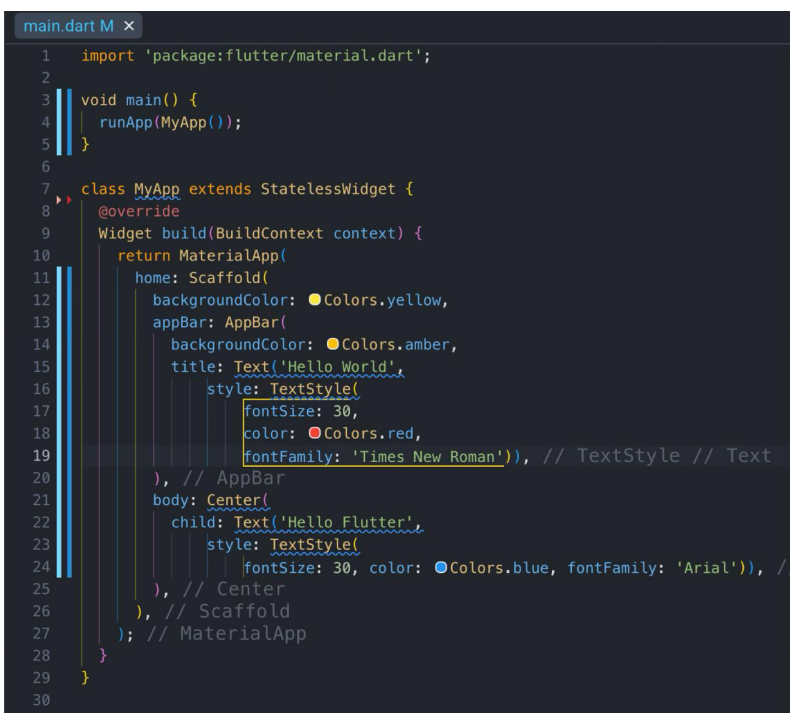
Theory : Android Studio is the official integrated development environment (IDE) for Android application development. It provides tools for designing, coding, debugging, and testing Android apps. It is based on JetBrains IntelliJ IDEA and supports Kotlin, Java, and C++.

Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and provides a rich set of pre-designed widgets for a smooth, customizable UI. Flutter is widely used for cross-platform development, enabling apps to run seamlessly on Android, iOS, Web, and Desktop with a single codebase.

The AndroidManifest.xml file is a crucial component of every Android application. It acts as a configuration file that defines essential app components, permissions, hardware requirements, and intent filters. The file contains details such as:

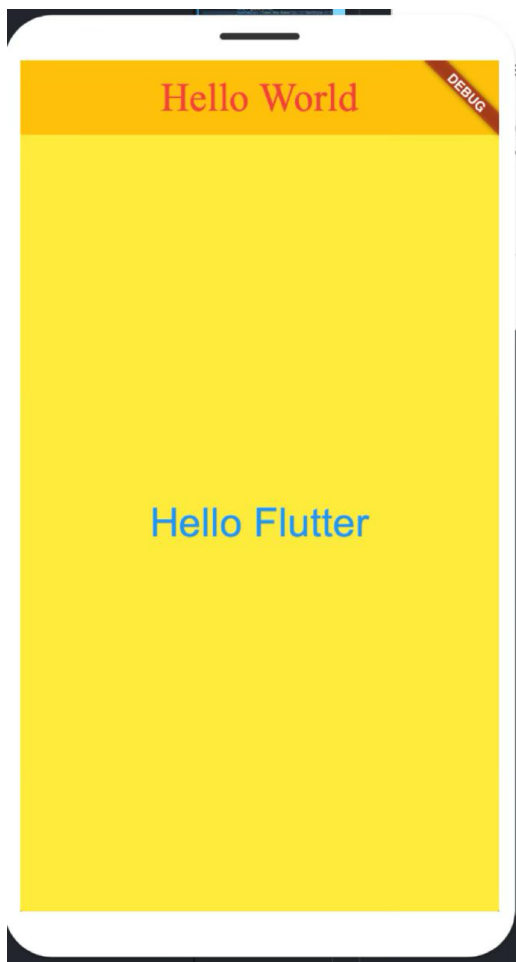
- Application components (Activities, Services, Broadcast Receivers, and Content Providers).
- Permissions required for app functionality (e.g., internet, camera, location).
- Hardware and software features needed for compatibility.
- App metadata, including package name, version code, and app theme.
-

Code :



```
main.dart M x
1  import 'package:flutter/material.dart';
2
3  void main() {
4    runApp(MyApp());
5  }
6
7  class MyApp extends StatelessWidget {
8    @override
9    Widget build(BuildContext context) {
10     return MaterialApp(
11       home: Scaffold(
12         backgroundColor: Colors.yellow,
13         appBar: AppBar(
14           backgroundColor: Colors.amber,
15           title: Text('Hello World',
16             style: TextStyle(
17               fontSize: 30,
18               color: Colors.red,
19               fontFamily: 'Times New Roman')), // TextStyle // Text
20         ), // AppBar
21         body: Center(
22           child: Text('Hello Flutter',
23             style: TextStyle(
24               fontSize: 30, color: Colors.blue, fontFamily: 'Arial')), //
25         ), // Center
26       ), // Scaffold
27     ); // MaterialApp
28   }
29 }
30
```

Output :



Learning Outcomes :

Experiment 2

Aim : Design a simple user interface for a mobile application using a design tool or framework like Sketch, Adobe XD, or Flutter.

Theory : A User Interface (UI) is the space where interactions between humans and machines occur. In mobile apps, UI includes visual components like buttons, text fields, images, navigation bars, and interactive layouts that enable the user to interact with the app efficiently and pleasantly.

Why Use Flutter in Android Studio?

Flutter is Google's open-source UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase. Android Studio provides a robust IDE with built-in Flutter and Dart support.

Key Features of Flutter :

- Single codebase for Android and iOS
- Fast development with hot reload
- Rich set of pre-built widgets
- Highly customisable UI
- Native performance

Best Practices in UI Design :

- Maintain consistency in colors, fonts, and layout
- Follow Material Design guidelines
- Optimize for various screen sizes and resolutions
- Ensure accessibility for all users
- Use clean code and modular widget structure

By using Flutter in Android Studio, we can rapidly design and implement visually appealing and functional UIs for mobile applications. This project serves as a foundational piece in building a strong developer portfolio, demonstrating both design sense and technical implementation.

Code :

Main.dart Screen code :

```
import 'package:flutter/material.dart';
import 'home_screen.dart';
import 'portfolio_screen.dart';
import 'contact_screen.dart';
void main() {
  runApp(PortfolioApp());
}
class PortfolioApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => HomeScreen(),
        '/portfolio_screen': (context) => PortfolioItemScreen(),
        '/contact_screen': (context) => ContactScreen(),
      },
    );
  }
}
```

home_Screen.dart screen code :

```
import 'package:flutter/material.dart';
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      body: Stack(
        children: [
          // Background image
          Positioned.fill(
            child: Image.asset(
              'lib/assets/wallpaper.jpg', // Change this to your image path
              fit: BoxFit.cover,
            ),
          ),
          Positioned.fill(
            child: Container(
              color: Colors.black.withOpacity(0.3), // Adjust opacity for readability
            ),
          ),
        ],
      ),
    );
  }
}
```

```

// Content
Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Text(
        'Welcome to My Portfolio!',
        style: TextStyle(fontSize: 24.0, fontWeight: FontWeight.bold, color: Colors.purple),
      ),
      SizedBox(height: 20.0),
      ElevatedButton(
        onPressed: () {
          Navigator.pushNamed(context, '/portfolio_screen');
        },
        child: Text('View Portfolio'),
      ),
      SizedBox(height: 10.0),
      ElevatedButton(
        onPressed: () {
          Navigator.pushNamed(context, '/contact_screen');
        },
        child: Text('Contact Me'),
      ),
    ],
  ),
),
),
],
),
);
}
}

```

portfolio_screen.dart screen code:

```

import 'package:flutter/material.dart';
class PortfolioItemScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Portfolio'),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: ListView(
          children: [
            // Profile Section
            Card(

```

```

elevation: 4,
shape: RoundedRectangleBorder(
  borderRadius: BorderRadius.circular(10),
),
child: Padding(
  padding: EdgeInsets.all(16.0),
  child: Column(
    children: [
      CircleAvatar(
        radius: 50,
        backgroundImage: AssetImage('lib/assets/photo.jpg'),
      ),
      SizedBox(height: 10),
      Text(
        'Lavanya Pundir', // Change to your name
        style: TextStyle(fontSize: 22, fontWeight: FontWeight.bold),
      ),
      Text(
        'Flutter Developer', // Change to your profession
        style: TextStyle(fontSize: 16, color: Colors.grey),
      ),
    ],
  ),
),
),
SizedBox(height: 20),
// Skills Section
_buildSectionTitle('Skills'),
_buildSkillCard([
  'Flutter & Dart',
  'Android & iOS Development',
  'Firebase & Cloud Firestore',
]),
SizedBox(height: 20),
// Education Section
_buildSectionTitle('Education'),
_buildEducationCard(
  degree: 'Bachelor of Technology',
  institution: 'VIPS-TC',
  year: '2022 - 2026',
),
SizedBox(height: 20),
// Back Button
Center(
  child: ElevatedButton(
    onPressed: () {
      Navigator.pop(context); // Navigate back
    },
    child: Text('BACK'),
  ),
),

```

```

        ),
    ),
],
),
),
);
}
// Section Title Widget
Widget _buildSectionTitle(String title) {
    return Text(
        title,
        style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
    );
}
// Skills Card Widget
Widget _buildSkillCard(List<String> skills) {
    return Card(
        elevation: 3,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: Padding(
            padding: EdgeInsets.all(16.0),
            child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: skills.map((skill) => Text('• $skill', style: TextStyle(fontSize: 16))).toList(),
            ),
        ),
    );
}
// Education Card Widget
Widget _buildEducationCard({required String degree, required String institution, required String
year}) {
    return Card(
        elevation: 3,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: Padding(
            padding: EdgeInsets.all(16.0),
            child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                    Text(degree, style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
                    SizedBox(height: 5),
                    Text(institution, style: TextStyle(fontSize: 16, color: Colors.grey[700])),
                    SizedBox(height: 5),
                    Text(year, style: TextStyle(fontSize: 14, color: Colors.grey[600])),
                ],
            ),
        ),
    );
}

```

```
}  
}
```

Contact_screen.dart screen code:

```
import 'package:flutter/material.dart';  
class ContactScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Contact'),  
      ),  
      body: Column(  
        crossAxisAlignment: CrossAxisAlignment.start,  
        children: [  
          Text(  
            'Contact Information',  
            style: TextStyle(  
              fontSize: 24,  
              fontWeight: FontWeight.bold,  
              color: Colors.teal,  
            ),  
          ),  
          SizedBox(height: 20),  
          Row(  
            children: [  
              Icon(Icons.person, color: Colors.teal),  
              SizedBox(width: 10),  
              Text(  
                'Lavanya Pundir',  
                style: TextStyle(fontSize: 16),  
              ),  
            ],  
          ),  
          SizedBox(height: 10),  
          Row(  
            children: [  
              Icon(Icons.email, color: Colors.teal),  
              SizedBox(width: 10),  
              Text(  
                'lavanya.pundir99@gmail.com',  
                style: TextStyle(fontSize: 16),  
              ),  
            ],  
          ),  
          SizedBox(height: 10),  
          Row(  

```

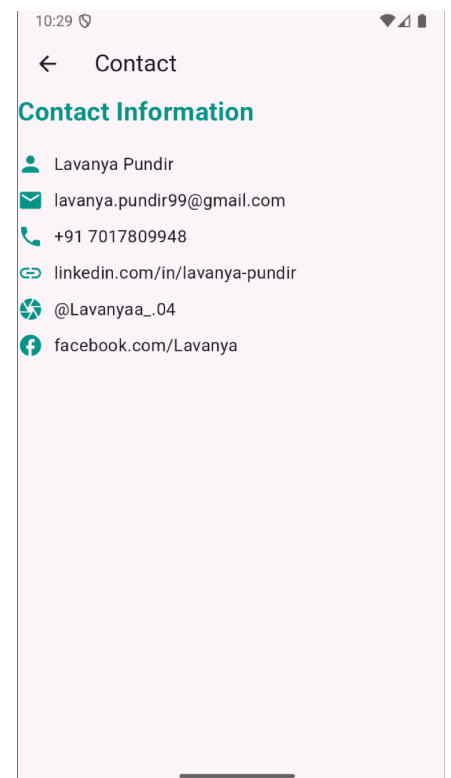
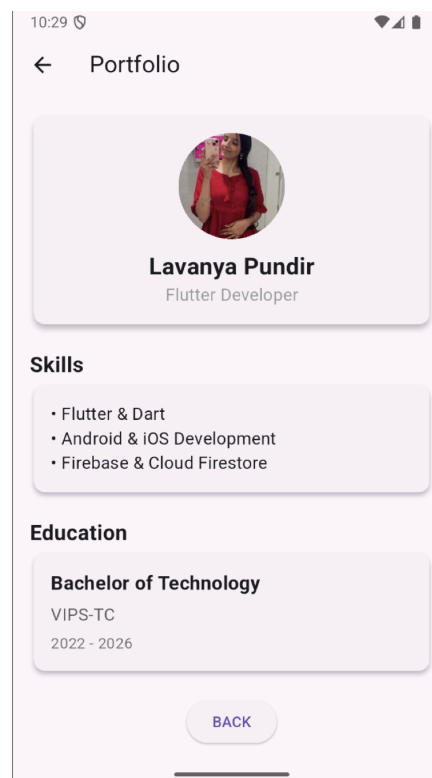
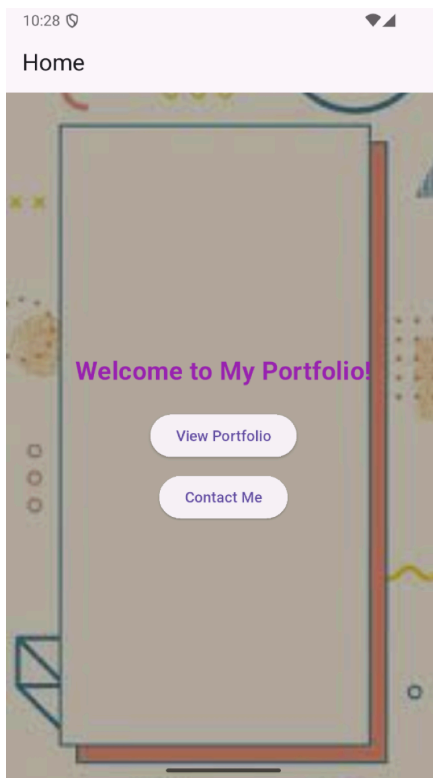


```

children: [
  Icon(Icons.phone, color: Colors.teal),
  SizedBox(width: 10),
  Text(
    '+91 7017809948',
    style: TextStyle(fontSize: 16),
  ),
],
),
SizedBox(height: 10),
Row(
  children: [
    Icon(Icons.link, color: Colors.teal),
    SizedBox(width: 10),
    Text(
      'linkedin.com/in/lavanya-pundir',
      style: TextStyle(fontSize: 16),
    ),
  ],
),
SizedBox(height: 10),
Row(
  children: [
    Icon(Icons.camera, color: Colors.teal),
    SizedBox(width: 10),
    Text(
      '@Lavanyaa_.04', // Instagram ID
      style: TextStyle(fontSize: 16),
    ),
  ],
),
SizedBox(height: 10),
Row(
  children: [
    Icon(Icons.facebook, color: Colors.teal),
    SizedBox(width: 10),
    Text(
      'facebook.com/Lavanya',
      style: TextStyle(fontSize: 16),
    ),
  ],
),
],
),
);
}
}

```

Output:



Learning Outcomes :

Experiment 3

Aim : Implement data storage functionality in your mobile application using local storage options like SQLite database or shared preferences.

Theory : Mobile applications often require saving data locally to maintain state, personalise user experience, or support offline functionality. Flutter, a popular cross-platform framework, provides several plugins and tools to manage local storage efficiently.

Local storage refers to saving data directly on the user's device, without needing an internet connection. It is essential for:

- User Preferences (e.g., theme, language)
- Session Persistence (e.g., remembering login state)
- Offline Access (e.g., storing notes, contacts, or cached data)
- Small Databases (e.g., storing tasks, messages)
-

Local Storage Options in Flutter:

1. Shared Preferences – Key-Value Pair Storage : Shared Preferences is a simple and lightweight storage system used for storing primitive data types such as:

- Booleans
- Integers
- Strings
- Doubles
- String Lists

It stores data as key-value pairs in XML files on the device.

2. SQLite Database – Structured Relational Database : SQLite is an embedded SQL database engine widely used in mobile apps. It supports full CRUD operations:

- Create (insert new records)
- Read (query existing data)
- Update (modify existing data)
- Delete (remove data)

Local data storage is a critical aspect of mobile app development. Using Flutter and Android Studio, developers can efficiently implement Shared Preferences for quick key-value storage and SQLite for structured, queryable data. Understanding and implementing these storage solutions enhances the app's functionality, offline usability, and user experience, making it a valuable addition to any developer's portfolio.

Code :

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DataStorageScreen(),
    );
  }
}
class DataStorageScreen extends StatefulWidget {
  @override
  _DataStorageScreenState createState() => _DataStorageScreenState();
}
class _DataStorageScreenState extends State<DataStorageScreen> {
  TextEditingController enrollmentController = TextEditingController();
  TextEditingController nameController = TextEditingController();
  TextEditingController projectController = TextEditingController();
  String studentDetails = "";
  @override
  void initState() {
    super.initState();
  }

  // Save student data to shared preferences
  Future<void> saveStudentData(
    String enrollment, String name, String project) async {
    try {
      SharedPreferences prefs = await SharedPreferences.getInstance();
      // Create a map to store student details using enrollment number as key
      Map<String, String> studentData = {
        'name': name,
        'project': project,
      };

      // Save the data in shared preferences, using the enrollment number as the key
      await prefs.setString(enrollment, studentData.toString());
    } catch (e) {
      print('Error saving data: $e');
    }
  }

  // Retrieve student data based on enrollment number
  Future<void> getStudentData(String enrollment) async {
    try {
      SharedPreferences prefs = await SharedPreferences.getInstance();
      String? data = prefs.getString(enrollment);
    }
  }
}
```

```

if (data != null) {
  setState(() {
    studentDetails = data;
  });
} else {
  setState(() {
    studentDetails = 'No data found for this enrollment number.';
  });
}
} catch (e) {
  print('Error retrieving data: $e');
}
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Student Data Storage'),
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: <Widget>[
          TextField(
            controller: enrollmentController,
            decoration: InputDecoration(labelText: 'Enter Enrollment Number'),
          ),
          TextField(
            controller: nameController,
            decoration: InputDecoration(labelText: 'Enter Student Name'),
          ),
          TextField(
            controller: projectController,
            decoration: InputDecoration(labelText: 'Enter Project Title'),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              String enrollment = enrollmentController.text;
              String name = nameController.text;
              String project = projectController.text;
              if (enrollment.isNotEmpty &&
                  name.isNotEmpty &&
                  project.isNotEmpty) {
                saveStudentData(enrollment, name, project);
              }
            },
            child: Text('Save Student Data'),
          ),
          SizedBox(height: 20),
          TextField(
            controller: enrollmentController,

```

Output :



Experiment 4

Aim : Develop a mobile application that interacts with a RESTful API to fetch and display data from a remote server.

Theory : Modern mobile applications often rely on remote servers to retrieve dynamic data. Whether it's a social media feed, weather updates, or product listings in an e-commerce app, the client (mobile app) and the server communicate over the internet using RESTful APIs.

Flutter provides powerful tools and packages, such as `http`, to seamlessly connect and interact with these web services.

A RESTful API (Representational State Transfer) is a web service that follows REST architecture principles to allow communication between client and server using standard HTTP methods.

* Key HTTP Methods:		
Method	Usage Example	Description
GET	<code>/users</code>	Fetches data from the server
POST	<code>/users</code>	Sends new data to the server
PUT	<code>/users/1</code>	Updates an existing resource
DELETE	<code>/users/1</code>	Deletes a specific resource

Why Use RESTful APIs in Mobile Apps?

- Fetch **live or remote data** dynamically (e.g., user profiles, weather, news)
- Perform **CRUD** operations on a backend
- Support real-time interactions and updates
- Separate front-end and back-end development
- Allow scalability and reuse of services

Interacting with RESTful APIs in a mobile application is an essential skill for modern developers. It allows apps to fetch dynamic content from remote servers, enabling a rich, connected, and personalized user experience. In Flutter, the `http` package simplifies HTTP communications, allowing developers to send requests, handle responses, and display the data efficiently. By building a Flutter app that fetches and displays data from a REST API, developers demonstrate key skills in asynchronous programming, JSON parsing, UI rendering, and API integration — crucial components of full-stack mobile development.

Code :

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter REST API Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
  List<dynamic> _data = [];
  @override
  void initState() {
    super.initState();
    fetchData();
  }
  Future<void> fetchData() async {
    final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));
    if (response.statusCode == 200) {
      setState(() {
        _data = json.decode(response.body);
      });
    } else {
      throw Exception('Failed to load data');
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Flutter REST API Demo'),
      ),
    ),
  }
}
```

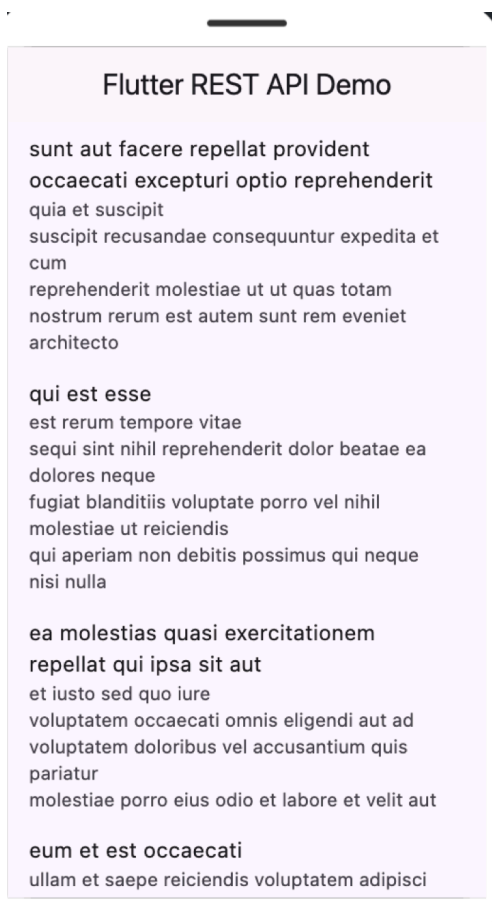


```

body: _data.isEmpty
    ? Center(child: CircularProgressIndicator())
    : ListView.builder(
  itemCount: _data.length,
  itemBuilder: (BuildContext context, int index) {
    return ListTile(
      title: Text(_data[index]['title']),
      subtitle: Text(_data[index]['body']),
    );
  },
);
}
}

```

Output :



Learning Outcomes :

Experiment 5

Aim : Integrate sensors such as accelerometer, gyroscope, or GPS into your mobile application to capture and utilize sensor data.

Theory : Modern smartphones come equipped with a wide range of hardware sensors that provide valuable data about the device's environment and movement. By accessing these sensors through mobile applications, developers can create context-aware and interactive experiences, such as fitness tracking, navigation apps, gaming, or augmented reality.

Flutter provides access to these sensors via plugins such as:

- `sensors_plus` for accelerometer & gyroscope
- `geolocator` or `location` for GPS

Types of Sensors :

1. GPS (Global Positioning System)

- **Type:** Location Sensor
- **Purpose:** Determines the device's geographic location using satellite data.
- **Common Use Cases:** Maps, navigation, ride-hailing apps, geo-fencing, fitness apps.

2. Accelerometer

- **Type:** Motion Sensor
- **Purpose:** Measures the acceleration force acting on the device in all three physical axes (X, Y, Z).
- **Common Use Cases:** Step counting, shake detection, tilt detection.

3. Gyroscope

- **Type:** Rotation Sensor
- **Purpose:** Measures the device's rate of rotation around each axis.
- **Common Use Cases:** Gaming, VR/AR apps, orientation-aware apps.

Advantages of Using Sensors in Mobile Apps :

- **Context Awareness:** Apps can respond to physical movement and environmental changes.
- **Real-time Tracking:** Enables live updates (e.g., user location or movement).
- **Enhanced User Experience:** Adds interactivity and gamification.
- **Automation:** Actions can be triggered automatically based on motion/location (e.g., auto-rotate, step counter, geo-notifications).

Integrating mobile device sensors such as the GPS, accelerometer, and gyroscope adds immense value to mobile applications by enabling real-time data capture and contextual awareness. Through plugins like `sensors`, `geolocator`, developers can efficiently access and use these sensor inputs to create dynamic and intelligent apps.

Code :

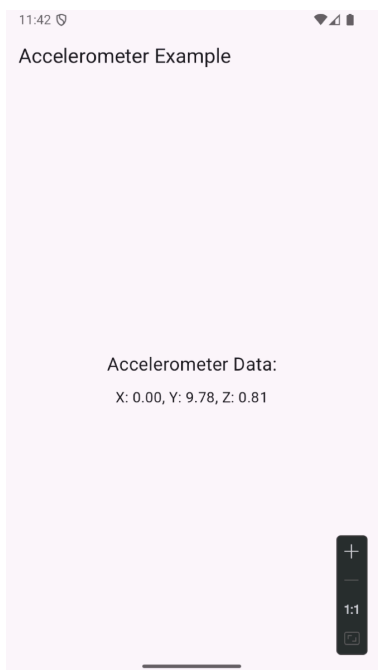
```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:sensors_plus/sensors_plus.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.green, // Set the app's primary theme color
      ),
      debugShowCheckedModeBanner: false,
      home: AccelerometerExample(),
    );
  }
}
class AccelerometerExample extends StatefulWidget {
  const AccelerometerExample({super.key});
  @override
  State<AccelerometerExample> createState() => _AccelerometerExampleState();
}
class _AccelerometerExampleState extends State<AccelerometerExample> {
  // List to store accelerometer data
  List<AccelerometerEvent> _accelerometerValues = [];
  // StreamSubscription for accelerometer events
  late StreamSubscription<AccelerometerEvent> _accelerometerSubscription;
  @override
  void initState() {
    super.initState();
    // Subscribe to accelerometer events
    _accelerometerSubscription = accelerometerEvents.listen((event) {
      setState(() {
        // Update the _accelerometerValues list with the latest event
        _accelerometerValues = [event];
      });
    });
  }
  @override
  void dispose() {
    // Cancel the accelerometer event subscription to prevent memory leaks
    _accelerometerSubscription.cancel();
    super.dispose();
  }
  @override
```

```

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Accelerometer Example'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'Accelerometer Data:',
            style: TextStyle(fontSize: 20),
          ),
          SizedBox(height: 10),
          if (_accelerometerValues.isNotEmpty)
            Text(
              'X: ${_accelerometerValues[0].x.toStringAsFixed(2)}, '
              'Y: ${_accelerometerValues[0].y.toStringAsFixed(2)}, '
              'Z: ${_accelerometerValues[0].z.toStringAsFixed(2)}',
              style: TextStyle(fontSize: 16),
            )
          else
            Text('No data available', style: TextStyle(fontSize: 16)), ], ), ), ); }

```

Output:



Learning Outcomes :