VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS
Grade A++ Accredited Institution by NAAC
NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE
An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

BTECH Programme: AI&DS

Course Title: Mobile Application Development Lab

Course Code: OAE312P

Submitted To: Dr. Alisha Sikri Ghai

Submitted By: Krishana Suthar

Enrollment no.: 10317711922

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS
Grade A++ Accredited Institution by NAAC
NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE
An ISO 9001:2015 Certified Institution
SCHOOL OF ENGINEERING & TECHNOLOGY

VISION OF INSTITUTE

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

MISSION OF INSTITUTE

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building capabilities to enable them to make significant contributions to the world.

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS
Grade A++ Accredited Institution by NAAC
NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE
An ISO 9001:2015 Certified Institution
SCHOOL OF ENGINEERING & TECHNOLOGY

INDEX

S.No	Experiment Name	Date	Marks			Remark	Updated Marks	Faculty Signature
			Laboratory Assessment (15 Marks)	Class Participation (5 Marks)	Viva (5 Marks)			

Experiment 1

Aim: Study the frameworks of Mobile Application Development and design tools like sketch, adobe XD or flutter.

Objectives:

- Gain the understanding of the frameworks of Mobile Application Development.
- Explore the tools used for designing, building, compiling and testing mobile applications.

Theory:

Frameworks of Mobile Application Development

Mobile application development involves creating software applications that run on mobile devices, such as smartphones and tablets. Developers use various frameworks to streamline and simplify the development process by providing pre-built components, libraries, and tools. Frameworks play a critical role in ensuring efficient development, high performance, and seamless user experiences across different platforms.

Native Frameworks

Native frameworks are platform-specific and allow developers to create applications optimized for a particular operating system. Examples include **Swift** and **Objective-C** for iOS development, and **Kotlin** and **Java** for Android development. These frameworks provide complete access to platform-specific features and APIs, ensuring high performance and better integration with the device's hardware and software. However, native development often requires maintaining separate codebases for different platforms, increasing development time and effort.

Cross-Platform Frameworks

Cross-platform frameworks enable developers to write a single codebase that can be deployed across multiple platforms, including iOS and Android. Popular cross-platform frameworks include **Flutter**, **React Native**, and **Xamarin**. These frameworks reduce development time and cost by eliminating the need for separate codebases. For example, Flutter, powered by Google, uses the Dart programming language and offers a rich set of pre-designed widgets for building visually appealing applications. React Native, backed by Facebook, allows developers to use JavaScript and reuse components between platforms, enhancing efficiency.

Hybrid Frameworks

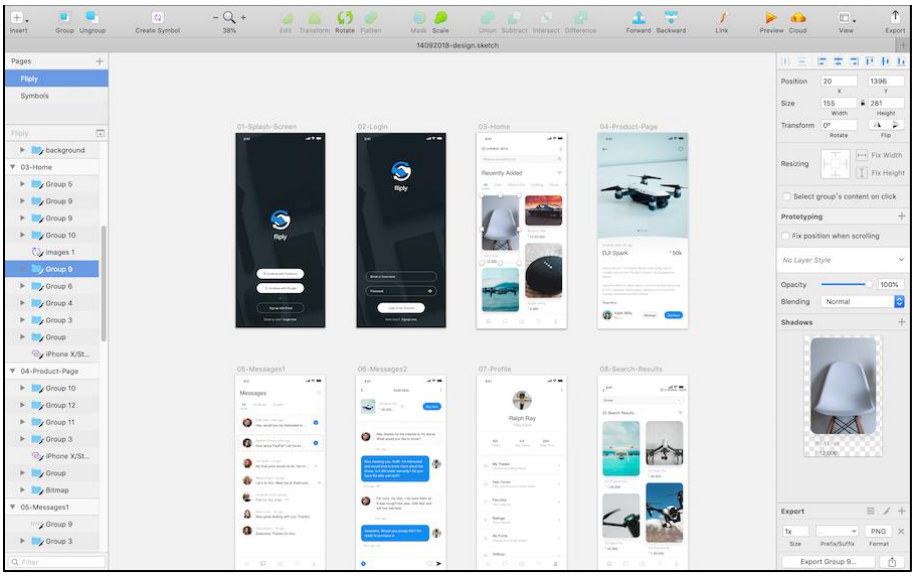
Hybrid frameworks, such as **Ionic** and **Cordova**, combine the advantages of web and native technologies. Applications developed using hybrid frameworks are essentially web apps wrapped in a native container. They rely on web technologies like HTML, CSS, and JavaScript but can access device-specific features through plugins. While hybrid frameworks enable faster development, they may face performance limitations compared to native applications.

Design Tools for Mobile Application Development

Design tools are an integral part of mobile application development, enabling designers to conceptualize, prototype, and refine user interfaces and experiences. Tools like **Sketch**, **Adobe XD**, and **Flutter's UI toolkit** have revolutionized the design process, making it more efficient and collaborative.

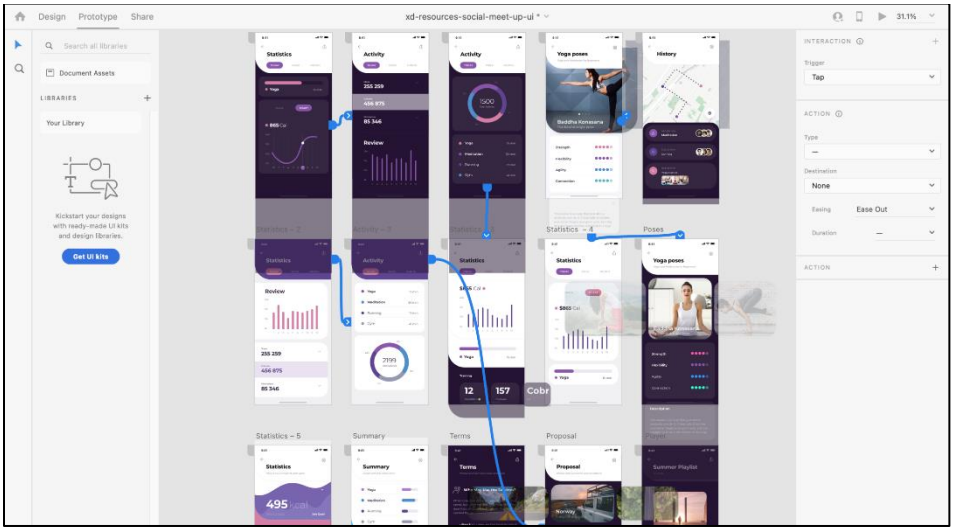
Sketch

Sketch is a vector-based design tool specifically tailored for UI and UX design. It allows designers to create wireframes, mockups, and high-fidelity prototypes. Sketch's simplicity, combined with its robust plugin ecosystem, makes it a favorite among designers. Features like reusable symbols, shared libraries, and support for collaborative workflows enhance productivity. Additionally, Sketch integrates seamlessly with third-party tools, making it easier to transition from design to development.



Adobe XD

Adobe XD is a powerful tool for designing and prototyping user experiences. It offers a wide range of features, including vector editing, responsive design capabilities, and interactive prototyping. Designers can create clickable prototypes to simulate user interactions and share them with stakeholders for feedback. Adobe XD's integration with other Adobe Creative Cloud apps, such as Photoshop and Illustrator, allows for a cohesive design workflow. Its collaborative features enable multiple team members to work on a project simultaneously, streamlining the design process.



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.

Objectives:

- Gain the understanding about User Interface and its significance.
- Gain knowledge of how to implement simple UI using any one framework.

Theory:

User Interface (UI) design is a crucial aspect of mobile application development, focusing on creating visually appealing and user-friendly interfaces. A well-structured UI enhances user experience (UX) by ensuring intuitive navigation, responsiveness, and accessibility.

Importance of UI in Mobile Applications

A good UI design improves usability, making the application easy to interact with. It enhances user engagement, reduces learning curves, and contributes to the overall success of an application. Key factors such as color schemes, typography, and element placement play a significant role in shaping user perception.

Principles of Effective UI Design

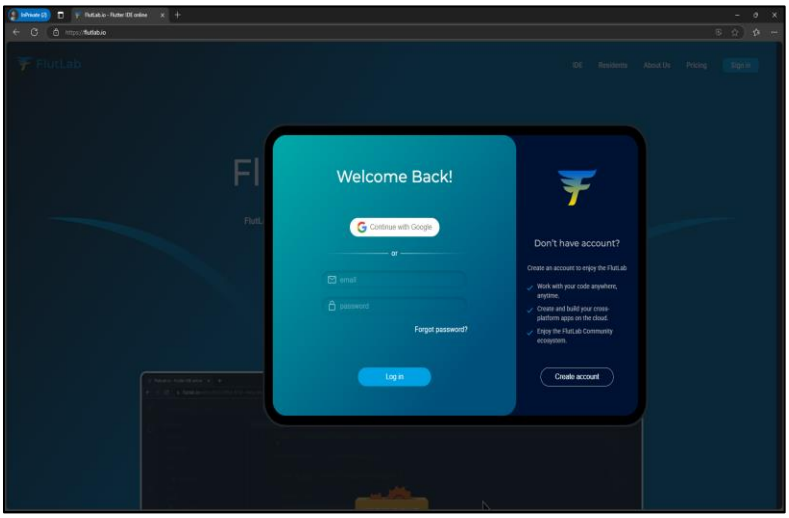
Modern UI design follows several essential principles:

- **Consistency:** Maintaining uniform design elements across the application.
- **Simplicity:** Avoiding clutter and keeping interactions straightforward.
- **Responsiveness:** Ensuring compatibility across different screen sizes and devices.
- **Accessibility:** Designing for users with diverse needs, including those with disabilities.

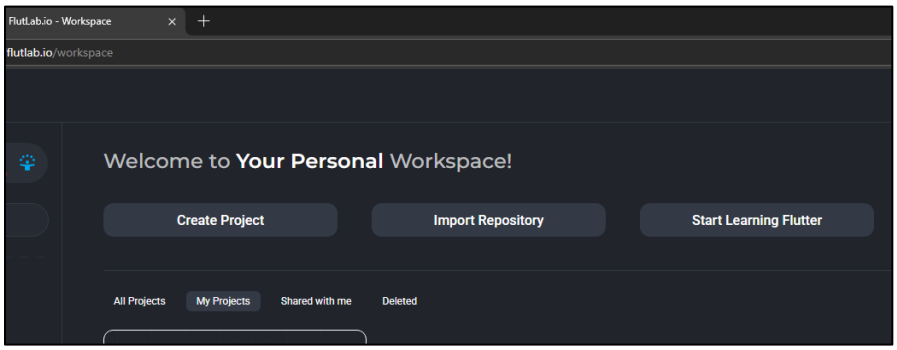
Steps to Design a simple UI using FlutLab:

Step 1: Go to flutlab.io

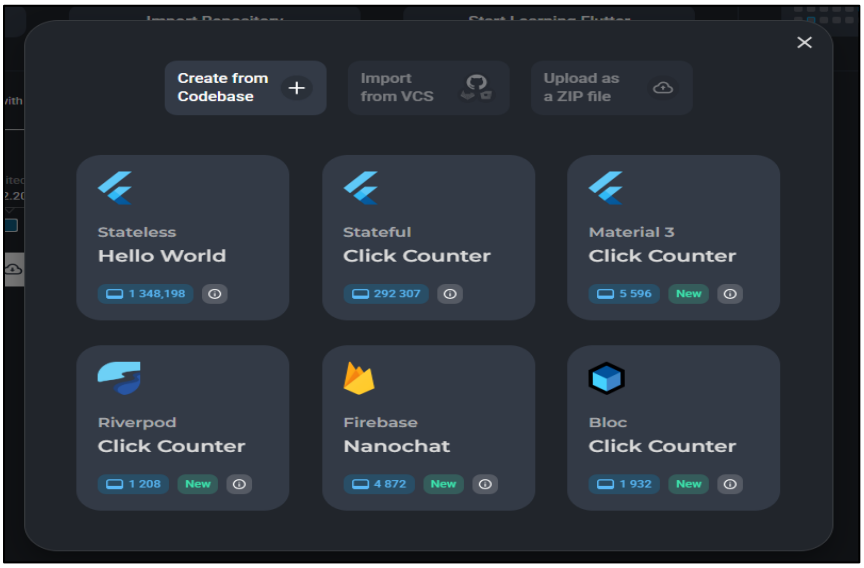
Step 2: Sign-In using your credentials or Create a new account.



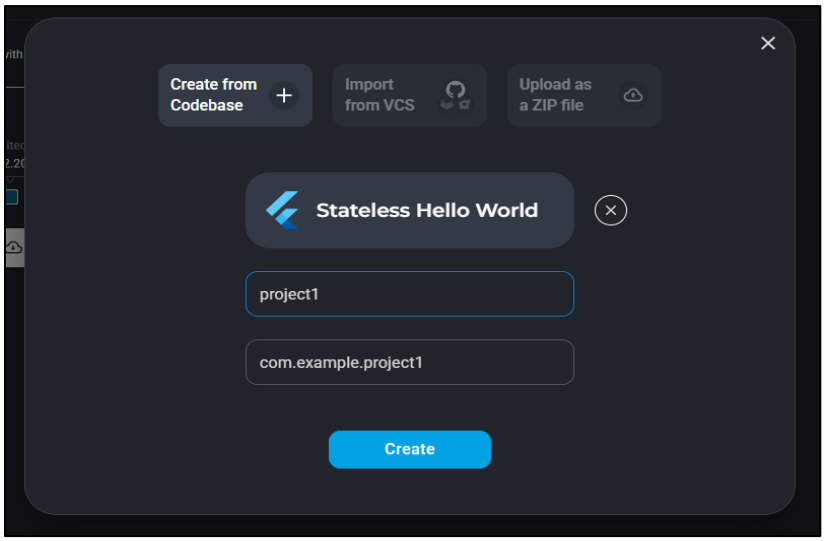
Step 3: Click on Create Project



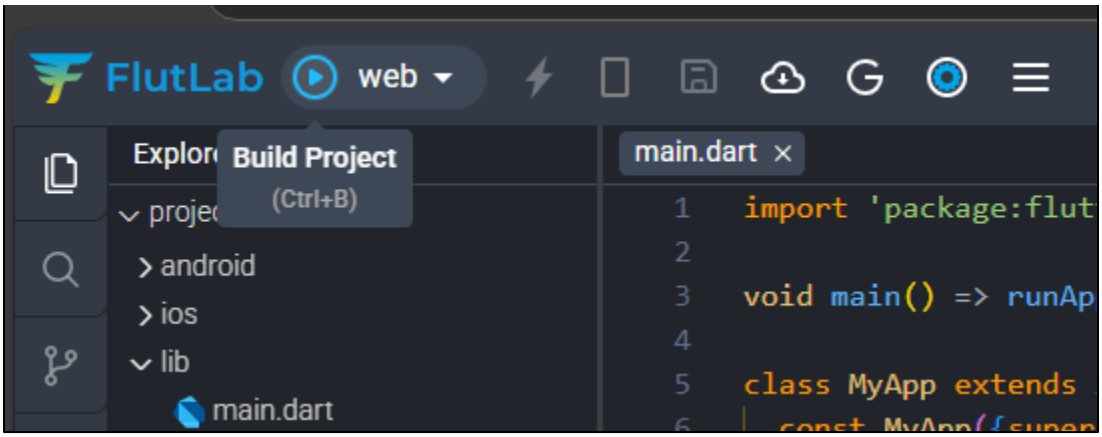
Step 4: Select 'Hello World'



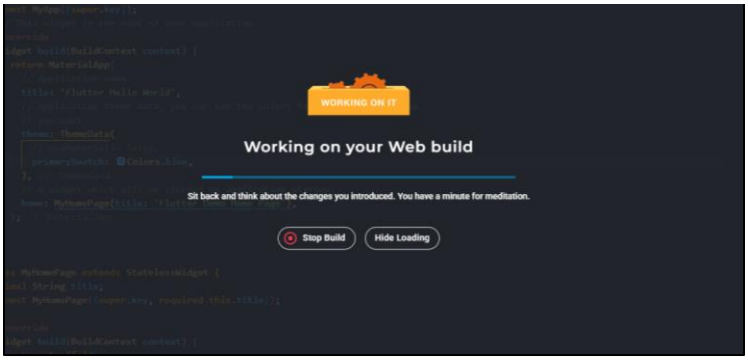
Step 5: Give a project Name



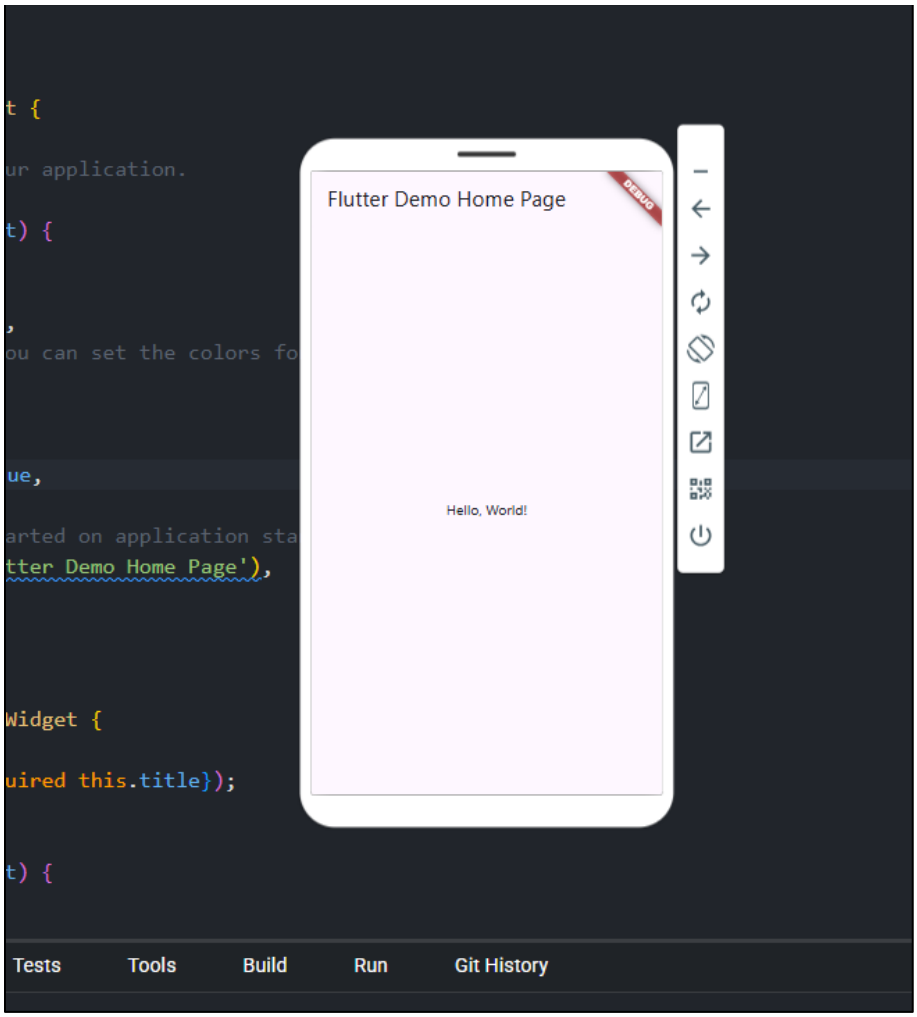
Step 6: Click on Build Project to Run the Provided Code



Step 7: Wait for the build to get complete.



Step 8: Output Screen is Shown.



Learning Outcomes:

Experiment 3

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.

Objectives:

- To Understand the basic development of Mobile Applications.
- To gain knowledge and practice of printing basic texts on a screen of mobile app.

Theory:

Introduction to Mobile Application Development

Mobile application development is the process of creating software applications that run on mobile devices such as smartphones and tablets. These applications can be developed for different platforms, primarily Android and iOS, using platform-specific or cross-platform development environments. A "Hello World" application is a fundamental starting point in mobile app development, serving as an introductory exercise to understand the development workflow, tools, and programming languages involved.

Fundamental Concepts in a Mobile Application

A basic "Hello World" application introduces key concepts in mobile development:

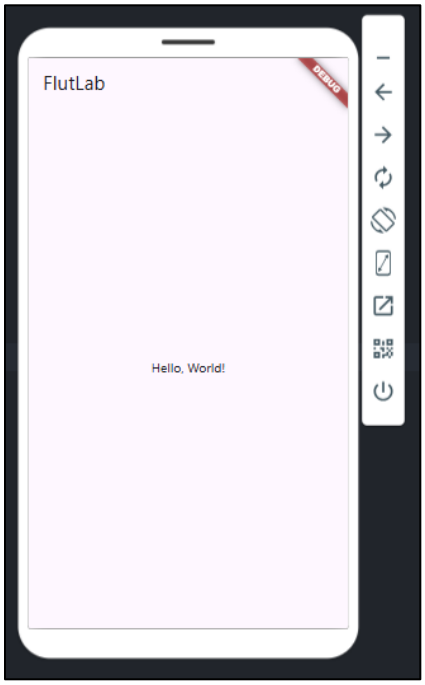
- **User Interface (UI):** The app's visual representation, usually defined using XML (Android), SwiftUI (iOS), or Flutter widgets.
- **Activity/ViewController:** The main screen or component that controls UI interaction.
- **Rendering & Lifecycle:** Understanding how the app initializes and displays content on the screen.

Code:

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

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Hello World',

      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'FlutLab'),
    );
```

```
  }
}

class MyHomePage extends StatelessWidget {
  final String title;
  const MyHomePage({super.key, required this.title});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title),
      ),
      body: Center(
        child: Text(
          'Hello, World!',
        ), ), ); }}
```

Output:



Learning Outcomes:

Experiment 4

Aim: Design a simple mobile user interface to add image on a mobile platform.

Objectives:

- To understand the basic development of Mobile Applications.
- To gain knowledge and practice of displaying Images on a screen of mobile app.

Theory:

Introduction

Flutter is an open-source UI software development framework by Google used for building natively compiled applications for mobile, web, and desktop from a single codebase. One of the fundamental tasks in Flutter application development is displaying images, which can be achieved using widgets like `Image.asset()` and `Image.network()`.

Objective

The objective of this experiment is to understand the process of displaying an image in a Flutter application using both local assets and network sources. This experiment aims to familiarize developers with Flutter's image handling mechanisms, asset management, and UI design considerations.

Concept and Implementation

Flutter provides the `Image` widget as a simple way to render images on the screen. There are two primary ways to load images:

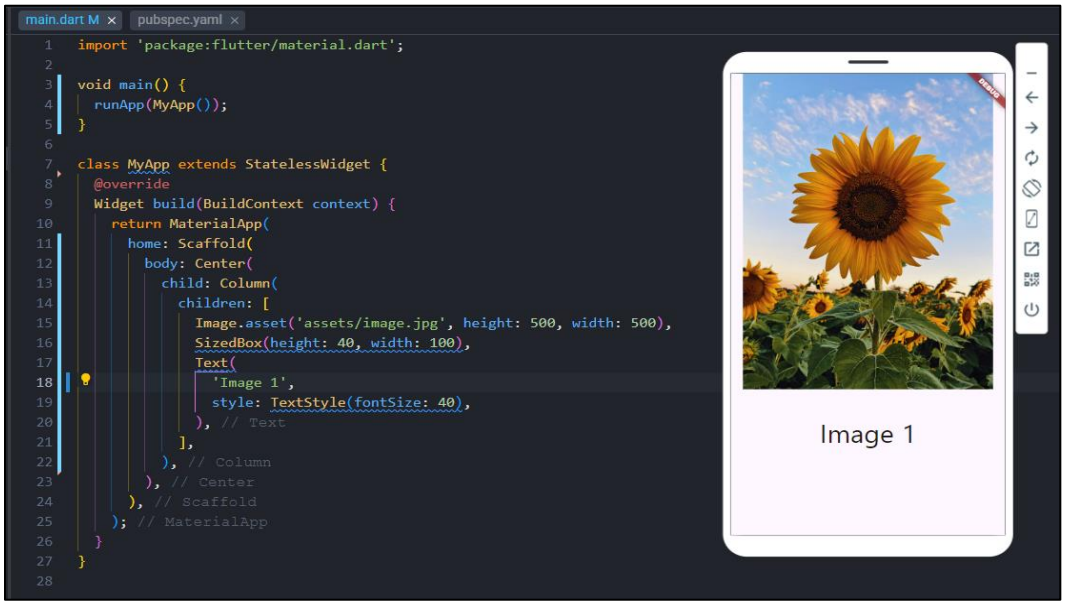
1. **Loading from Assets**: Images stored in the project's asset directory are displayed using the `Image.asset()` constructor. The asset must be declared in the `pubspec.yaml` file before use.
2. **Loading from a Network**: External images can be fetched from the internet using `Image.network()` by providing a valid URL.

Code:

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

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

class MyApp extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            children: [
              Image.asset('assets/image.jpg', height: 500, width: 500),
              SizedBox(height: 40, width: 100),
              Text(
                'Image 1',
                style: TextStyle(fontSize: 40),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:



Learning Outcomes:

Experiment 5

Aim: Develop a mobile application that interacts with a button.

Objectives:

- To understand the basic development of Mobile Applications.
- To gain knowledge and practice of displaying buttons on a screen of mobile app for interaction.

Theory:

Buttons are essential UI components in mobile applications that enable user interaction. In Flutter, buttons are widely used for triggering events, submitting forms, or navigating between screens. Buttons enhance the user experience by providing intuitive access to different functionalities. Flutter provides multiple types of buttons such as `ElevatedButton`, `TextButton`, `OutlinedButton`, and `IconButton`, each serving different UI requirements. Additionally, these buttons can be customized with different styles, colors, and animations to enhance usability and aesthetic appeal.

Types of Buttons in Flutter

1. **ElevatedButton**: A button with a shadow effect, useful for emphasizing primary actions and adding depth to the UI.
2. **TextButton**: A simple button with no elevation, typically used for secondary actions or inline links.
3. **OutlinedButton**: A button with an outlined border, offering a balance between prominence and subtlety, often used when a softer emphasis is needed.
4. **IconButton**: A button displaying an icon instead of text, often used for toolbars, navigation, or quick actions.

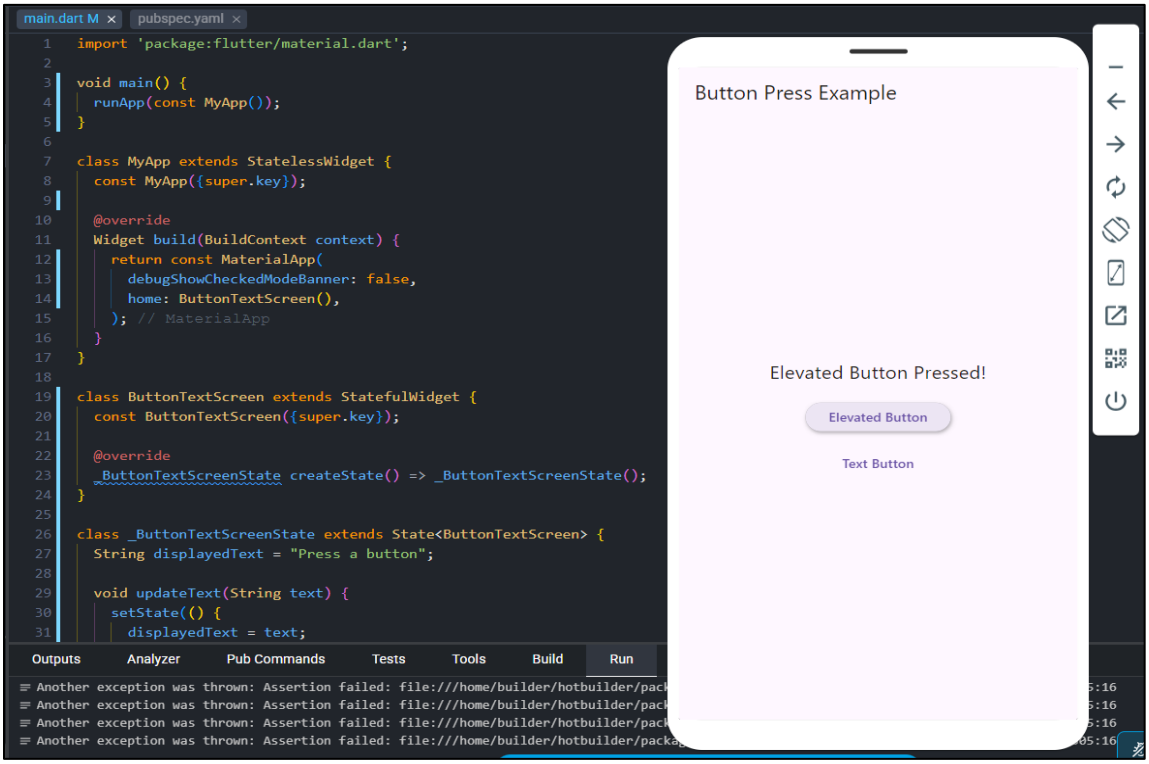
Code

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      home: ButtonTextScreen(),
```

```
    );
  }
}
class ButtonTextScreen extends StatefulWidget {
  const ButtonTextScreen({super.key});

  @override
  _ButtonTextScreenState createState() => _ButtonTextScreenState();
}

class _ButtonTextScreenState extends State<ButtonTextScreen> {
  String displayedText = "Press a button";

  void updateText(String text) {
    setState(() {
      displayedText = text;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Button Press Example')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              displayedText,
              style: const TextStyle(fontSize: 20),
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              onPressed: () => updateText("Elevated Button Pressed!"),
              child: const Text('Elevated Button'),
            ),
            const SizedBox(height: 20),
            TextButton(
              onPressed: () => updateText("Text Button Pressed!"),
              child: const Text('Text Button'),
            ),
          ],
        ),
      ),
    );
```

```
  }
}
```

Output:



Learning Outcomes:

Experiment 6

Aim: Develop a mobile application of portfolio.

Objectives:

- To implement various Flutter widgets for structured content presentation.
- To create a responsive and visually appealing digital resume accessible across multiple devices.

Theory:

The Portfolio Flutter App is a mobile application developed using the Flutter framework to showcase a professional portfolio. It serves as a digital resume, allowing users to present their skills, education, projects, and other relevant professional information in an interactive and visually appealing manner.

Technology Stack

The app is built using Flutter, an open-source UI software development toolkit created by Google. Flutter uses the Dart programming language and provides a cross-platform development environment, enabling the application to run seamlessly on both Android and iOS devices.

Features

1. **Profile Section**: Displays the user's profile picture, name, and a brief introduction.
2. **Education Section**: Lists academic qualifications, including university details.
3. **Skills Section**: Highlights key technical and soft skills.
4. **Projects Section**: Provides an overview of completed and ongoing projects.
5. **Interactive UI**: A well-structured and visually appealing interface using Material Design principles.
6. **Responsiveness**: Ensures a smooth experience across various screen sizes.
7. **Scalability**: Can be expanded to include additional sections like work experience, certifications, and contact details.

UI/UX Design Principles

The app follows modern UI/UX design principles with a clean layout, readable typography, and intuitive navigation. The design ensures that users can easily browse through different sections with a seamless experience.

Code:

Home:

```
import 'package:flutter/material.dart';
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
      appBar: AppBar(
        title: Text('Home'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'Welcome to My Portfolio!',
              style: TextStyle(fontSize: 24.0),
            ),
            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

```
import 'package:flutter/material.dart';
class PortfolioItemScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Portfolio Item'),
        backgroundColor: Color(0xffbeb5ef), // Set AppBar color for consistency
      ),
      body: Padding(
        padding:
            const EdgeInsets.all(16.0), // Padding to make the UI more spacious
        child: SingleChildScrollView(
          // Make it scrollable if content overflows
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
```

```
children: [
  // Profile Picture
  Center(
    child: CircleAvatar(
      radius: 60,
      backgroundImage: AssetImage(
          'assets/avatar.jpg'), // Replace with your image path
    ),
  ),
  SizedBox(height: 16),

  // Title with improved styling
  Center(
    child: Text(
      'Vikram Ranjan',
      style: TextStyle(
        fontSize: 28,
        fontWeight: FontWeight.bold,
        color: Color(0xff3d3664),
      ),
    ),
  ),
  SizedBox(height: 16),

  // Education section title
  Text(
    'Education',
    style: TextStyle(
      fontSize: 24,
      fontWeight: FontWeight.bold,
      color: Color(0xff3d3664),
    ),
  ),
  SizedBox(height: 16),

  // Education details with Icon and text aligned
  Row(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Icon(Icons.school, color: Colors.teal, size: 24),
      SizedBox(width: 10),
      Expanded(
        child: Text(
          'University: Vivekananda Institute of Professional Studies.',
          style: TextStyle(fontSize: 16),
        ),
      ),
    ],
  ),
  SizedBox(height: 20),
```

```
            // Skills section
            Text(
              'Skills',
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                color: Color(0xff3d3664),
              ),
            ),
            SizedBox(height: 10),
            Text(
              '• Flutter Development\n• Dart Programming\n• UI/UX Design',
              style: TextStyle(fontSize: 16),
            ),
            SizedBox(height: 20),

            // Projects Section
            Text(
              'Projects',
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                color: Color(0xff3d3664),
              ),
            ),
            SizedBox(height: 10),
            Text(
              '• Portfolio App\n• E-commerce App\n• Chat Application',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Contact Screen

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

class ContactScreen extends StatelessWidget {
  // Function to launch a URL
  Future<void> _launchURL(String url) async {
    if (await canLaunch(url)) {
      await launch(url);
    } else {
      throw 'Could not launch $url';
    }
  }
```

```
@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: 40),
        Row(
          children: [
            Icon(Icons.person, color: Colors.teal),
            SizedBox(width: 10),
            Text(
              'Vikram Ranjan',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
        SizedBox(height: 10),
        Row(
          children: [
            Icon(Icons.email, color: Colors.teal),
            SizedBox(width: 10),
            InkWell(
              onTap: () => _launchURL('mailto:vikramranjan71122@gmail.com'),
              child: Text(
                'vikramranjan71122@gmail.com',
                style: TextStyle(fontSize: 16, color: Colors.blue),
              ),
            ),
          ],
        ),
        SizedBox(height: 10),
        Row(
          children: [
            Icon(Icons.phone, color: Colors.teal),
            SizedBox(width: 10),
            InkWell(
              onTap: () => _launchURL('tel:+919871568696'),
              child: Text(
```

```
                '+91 9871568696',
                style: TextStyle(fontSize: 16, color: Colors.blue),
              ),
            ),
          ],
        ),
        SizedBox(height: 10),
        Row(
          children: [
            Icon(Icons.link, color: Colors.teal),
            SizedBox(width: 10),
            InkWell(
              onTap: () =>
                  _launchURL('https://linkedin.com/in/vikram-ranjan890'),
              child: Text(
                'linkedin.com/in/vikram-ranjan890',
                style: TextStyle(fontSize: 16, color: Colors.blue),
              ),
            ),
          ],
        ),
        SizedBox(height: 10),
        Row(
          children: [
            Icon(Icons.camera, color: Colors.teal),
            SizedBox(width: 10),
            InkWell(
              onTap: () =>
                  _launchURL('https://www.instagram.com/vikram-ranjan-353'),
              child: Text(
                '@vikram-ranjan-353', // Instagram ID
                style: TextStyle(fontSize: 16, color: Colors.blue),
              ),
            ),
          ],
        ),
      ],
    ),
  );
 }
}
```

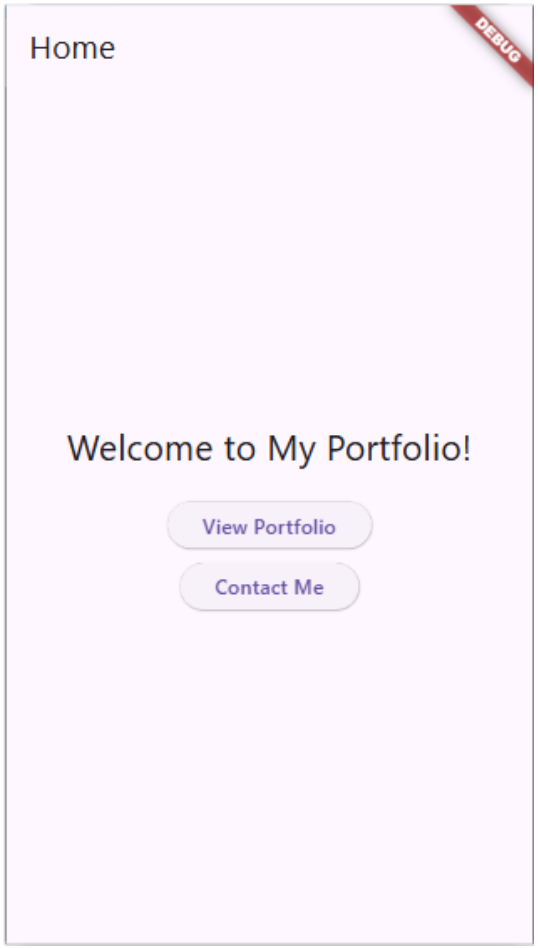
Main file

```
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(),
      },
    );
  }
}
```

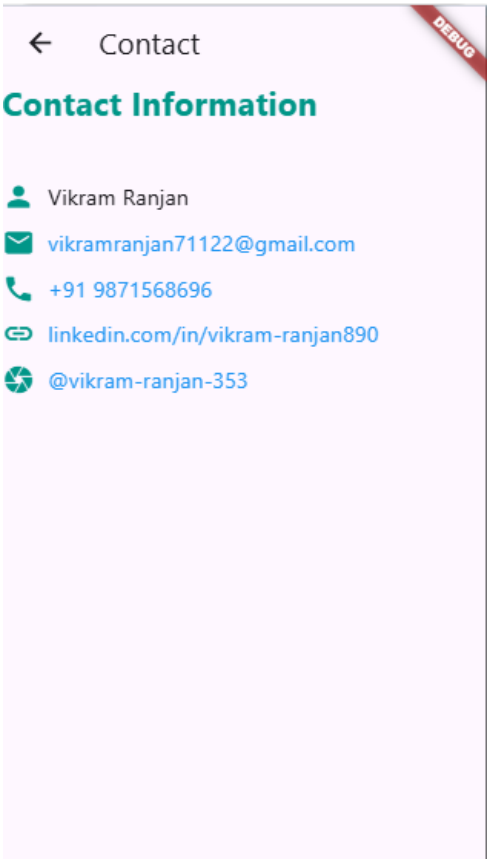
Output

Home



Portfolio

Contact



Learning Outcome

Experiment 7

Aim: Write Flutter Mobile Application for Data Store Shared Preferences.

Objectives:

- To store, retrieve, and delete student data using enrollment numbers as unique keys in Flutter with SharedPreferences.
- To implement a user-friendly interface that allows efficient data management with proper input validation and UI spacing.

Theory:

SharedPreferences in Flutter is a lightweight storage solution used for persisting key-value data locally on a device. It is commonly utilized for storing user preferences, session details, and small datasets that do not require a full-fledged database. In this project, SharedPreferences is employed to save student information using an enrollment number as the unique key. The app provides a structured UI, ensuring that users input necessary details before saving, retrieving, or deleting data. By incorporating text fields and buttons with proper spacing, the interface enhances usability while maintaining efficient data management.

Code

Main.dart

```
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(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primarySwatch: Colors.blue),
      home: DataStorageScreen(),
    );
  }
```

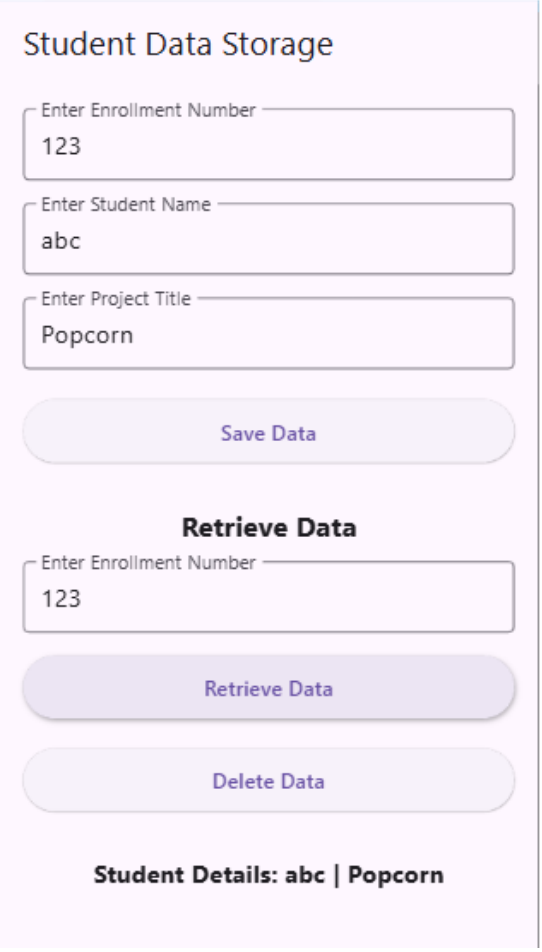
```
}

class DataStorageScreen extends StatefulWidget {
  @override
  _DataStorageScreenState createState() => _DataStorageScreenState();
}

class _DataStorageScreenState extends State<DataStorageScreen> {
  TextEditingController enrollmentController = TextEditingController();
  TextEditingController nameController = TextEditingController();
  TextEditingController projectController = TextEditingController();
  TextEditingController retrieveController = TextEditingController();
  String studentDetails = '';

  Future<void> saveStudentData(
      String enrollment, String name, String project) async {
    try {
      SharedPreferences prefs = await SharedPreferences.getInstance();
      await prefs.setString(enrollment, '$name | $project');
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Data saved successfully!')),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Error saving data: $e')),
      );
    }
  }

  Future<void> getStudentData(String enrollment) async {
    try {
      SharedPreferences prefs = await SharedPreferences.getInstance();
      String? data = prefs.getString(enrollment);
      setState(() {
```

```
      studentDetails = data ?? 'No data found for this enrollment number.';
    });
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Error retrieving data: $e')),
    );
  }
}

Future<void> deleteStudentData(String enrollment) async {
  try {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.remove(enrollment);
    setState(() {
      studentDetails = 'Data deleted successfully!';
    });
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Error deleting 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: SingleChildScrollView(
        child: Column(
          children: <Widget>[
            buildTextField(enrollmentController, 'Enter Enrollment Number'),
            SizedBox(height: 15),
```

```
        buildTextField(nameController, 'Enter Student Name'),
        SizedBox(height: 15),
        buildTextField(projectController, 'Enter Project Title'),
        SizedBox(height: 20),
        buildButton('Save Data', () {
          if (enrollmentController.text.isNotEmpty &&
              nameController.text.isNotEmpty &&
              projectController.text.isNotEmpty) {
            saveStudentData(enrollmentController.text,
                nameController.text, projectController.text);
          }
        }),
        SizedBox(height: 30),
        Text(
          'Retrieve Data',
          style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
        ),
        SizedBox(height: 10),
        buildTextField(retrieveController, 'Enter Enrollment Number'),
        SizedBox(height: 15),
        buildButton('Retrieve Data',
            () => getStudentData(retrieveController.text)),
        SizedBox(height: 20),
        buildButton('Delete Data',
            () => deleteStudentData(retrieveController.text)),
        SizedBox(height: 30),
        Text(
          'Student Details: $studentDetails',
          style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          textAlign: TextAlign.center,
        ),
      ],
    ),
  ),
```

```
      ),
    );
  }

  Widget buildTextField(TextEditingController controller, String hintText) {
    return TextField(
      controller: controller,
      decoration: InputDecoration(
        labelText: hintText,
        border: OutlineInputBorder(),
      ),
    );
  }

  Widget buildButton(String text, VoidCallback onPressed) {
    return ElevatedButton(
      style: ElevatedButton.styleFrom(minimumSize: Size(double.infinity, 50)),
      onPressed: onPressed,
      child: Text(text),
    );
  }
}
```

Output



Learning Outcome

Experiment 8

Aim: Create a flutter app using Restful API.

Objective:

To develop a cross-platform mobile application using Flutter that integrates with a RESTful API to fetch, display, and manipulate data dynamically, thereby enhancing real-time user interaction and demonstrating effective communication between the frontend and backend services.

Theory:

Flutter is an open-source UI software development toolkit created by Google, used to build natively compiled applications for mobile, web, and desktop from a single codebase. It uses Dart as its programming language and provides a rich set of pre-designed widgets to create visually appealing and responsive interfaces.

A **RESTful API (Representational State Transfer)** is an architectural style for designing networked applications. It allows client applications (like mobile apps) to communicate with backend services over HTTP using standard methods such as GET, POST, PUT, and DELETE. These APIs return data typically in JSON format, which can be easily parsed and displayed in a Flutter app.

By integrating a RESTful API in a Flutter app:

- The app becomes dynamic and data-driven.
- Backend data can be retrieved and updated without rebuilding the app.
- Real-time interaction between users and servers can be achieved, improving user experience.

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.deepPurple,
        colorScheme: ColorScheme.fromSwatch(primarySwatch: Colors.deepPurple)
            .copyWith(secondary: Colors.orange),
```

```
      textTheme: TextTheme(
        titleMedium: TextStyle(
            fontSize: 20,
            fontWeight:
                FontWeight.bold), // Replaced headline5 with titleMedium
        bodyLarge: TextStyle(
            fontSize: 16,
            color: Colors.black87), // Replaced bodyText1 with bodyLarge
      ),
      scaffoldBackgroundColor: Colors.grey[100],
    ),
    home: MyHomePage(),
  );
 }
}

class MyHomePage extends StatefulWidget {
 @override
 _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
 List<dynamic> _data = [];
 bool _isLoading = true;
 String? _errorMessage;

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

 Future<void> fetchData() async {
  setState(() {
   _isLoading = true;
   _errorMessage = null;
  });

  try {
   final response = await http
       .get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));
   if (response.statusCode == 200) {
    setState(() {
     _data = json.decode(response.body);
    });
   } else {
```

```
      setState(() {
        _errorMessage = 'Failed to load data. Please try again later.';
      });
    }
  } catch (e) {
    setState(() {
      _errorMessage = 'An error occurred. Please check your connection.';
    });
  } finally {
    setState(() {
      _isLoading = false;
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Flutter REST API Demo'),
    ),
    body: _isLoading
        ? Center(child: CircularProgressIndicator())
        : _errorMessage != null
            ? Center(
                child: Text(_errorMessage!,
                    style: TextStyle(color: Colors.red, fontSize: 18)))
            : ListView.builder(
                itemCount: _data.length,
                itemBuilder: (BuildContext context, int index) {
                  return Card(
                    margin: EdgeInsets.symmetric(vertical: 8, horizontal: 12),
                    elevation: 4,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(10),
                    ),
                    child: ListTile(
                      contentPadding: EdgeInsets.all(16),
                      title: Text(
                        _data[index]['title'],
                        style: Theme.of(context)
                            .textTheme
                            .titleMedium, // Use titleMedium
                      ),
                      subtitle: Text(
                        _data[index]['body'],
```

```
                    style: Theme.of(context)
                        .textTheme
                        .bodyLarge, // Use bodyLarge
                    maxLines: 3,
                    overflow: TextOverflow.ellipsis,
                  ),
                ),
              );
            },
          ),
    );
  }
}
```

Output

Learning Outcome

Experiment 9

Aim: To create a Flutter application that captures and displays real-time accelerometer sensor data using the sensors_plus package.

Objective:

- To understand how to access and use hardware sensors in a mobile device through Flutter.
- To subscribe to and handle real-time accelerometer events.
- To display the current X, Y, and Z axis values of the device's motion in a user-friendly UI.
- To practice proper subscription and disposal of sensor streams to manage memory efficiently.

Theory

The **accelerometer** is a hardware sensor used in mobile devices to measure the acceleration force applied to the device along the X, Y, and Z axes. This force can be due to gravity or movement and is useful in a variety of applications such as motion detection, orientation changes, and step counting.

In Flutter, the sensors_plus package provides easy access to device sensors including the **accelerometer**, **gyroscope**, and **magnetometer**. The accelerometer data is accessed through a stream (accelerometerEvents) which emits AccelerometerEvent objects containing the X, Y, and Z values.

This Flutter app:

- Initializes a subscription to the accelerometer stream in initState().
- Updates the UI with the latest accelerometer readings using setState().
- Properly cancels the subscription in dispose() to prevent memory leaks.

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 Outcome