WEEK-9

Aim: REST API Integration
a)Fetch data from a REST API.

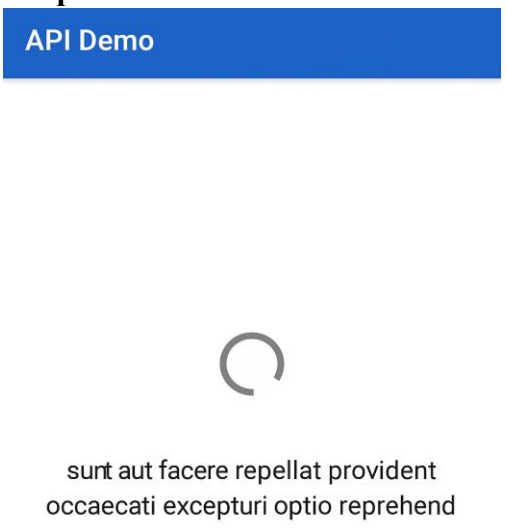
Description:
This Flutter app fetches a single post from a REST API using the **http** package.
The API used is https://jsonplaceholder.typicode.com/posts/1.
It decodes the JSON response into a Dart Map using **jsonDecode()**.
The data is stored in a state variable and displayed after fetching.
While loading, a **CircularProgressIndicator** spinner appears.
Once the data is loaded, it shows the **title** of the post on the screen.

Source Code:

```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'package:http/http.dart' as http;

void main() => runApp(MaterialApp(home: ApiDemo()));

class ApiDemo extends StatefulWidget {
  @override
  _ApiDemoState createState() => _ApiDemoState();
}

class _ApiDemoState extends State<ApiDemo> {
  Map<String, dynamic>? data;

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

  // Function to fetch data from REST API
  void fetchData() async {
    final response = await http.get(
      Uri.parse('https://jsonplaceholder.typicode.com/posts/1'),
    );
    if (response.statusCode == 200) {
      setState(() {
        data = jsonDecode(response.body);
      });
    } else {
      setState(() {
        data = {"error": "Failed to load data"};
      });
```

```
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: const Text('API Demo')),
   body: Center(
    child: data == null
      ? const CircularProgressIndicator() // Show loading spinner
      : data!['error'] != null
        ? Text(data!['error']) // Show error if API fails
        : Padding(
          padding: const EdgeInsets.all(16),
          child: Text(
           data!['title'].toString(), // Display fetched title
           textAlign: TextAlign.center,
           style: const TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.w500,
           ),
          ),
         ),
   ),
  );
 }
}
```

Output:



Aim:
b) Display the fetched data in a meaningful way in the UI.

Description:
This Flutter app demonstrates how to fetch and display data from a REST API using the http package.
It sends a GET request to https://jsonplaceholder.typicode.com/posts/1 to retrieve a sample post.
The received JSON data is converted into a Dart Map using jsonDecode().
While loading, a CircularProgressIndicator spinner appears in the center.
After fetching, it neatly displays the post's ID, title, and body inside a padded column.
The app uses a StatefulWidget so that the UI updates automatically when data is loaded.

Source Code:
```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'package:http/http.dart' as http;

void main() => runApp(MaterialApp(home: ApiDemo()));

class ApiDemo extends StatefulWidget {
  @override
  _ApiDemoState createState() => _ApiDemoState();
}

class _ApiDemoState extends State<ApiDemo> {
  Map<String, dynamic>? data;

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

  void fetchData() async {
    final response =
        await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts/1'));
    setState(() {
      data = jsonDecode(response.body);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('API Data Display')),
```

```
    body: Center(
      child: data == null
          ? const CircularProgressIndicator()
          : Padding(
              padding: const EdgeInsets.all(16),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Text('ID: ${data!['id']}'),
                  const SizedBox(height: 8),
                  Text('Title: ${data!['title']}',
                      style: const TextStyle(fontWeight: FontWeight.bold)),
                  const SizedBox(height: 8),
                  Text('Body: ${data!['body']}'),
                ],
              ),
            ),
      ),
    );
  }
}
```

Output:

ID: 1

Title: sunt aut facere repellat provident occaecati excepturi optio reprehenderit

Body: quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

WEEK-10

Aim:

Testing and Debugging

a)Write unit tests for UI components.

Description:

This Flutter program demonstrates a simple unit widget test using Flutter's testing framework. The app defines a stateless widget named GreetingWidget, which takes a name as input and displays a greeting message — "Hello, Appalakonda!". The test file checks whether the widget renders the correct text on the screen. Using the testWidgets() function, it ensures the expected greeting is shown and incorrect text does not appear. This verifies that the UI behaves as intended when given specific input. Running the test confirms that the widget logic and rendering are functioning correctly.

Source Code:

```
import 'package:flutter/material.dart';
void main() => runApp(MaterialApp(home: GreetingWidget(name: 'Appalakonda')));
class GreetingWidget extends StatelessWidget {
  final String name;
  const GreetingWidget({Key? key, required this.name}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Greeting Test Example')),
      body: Center(
        child: Text(
          'Hello, $name!',
          style: const TextStyle(fontSize: 24),
        ),
      ),
    );
  }
}
```

Step 2: Create Test File (test/greeting_widget_test.dart)

Create a folder named test in your project's root directory, then add this file inside it:

```
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter/material.dart';
import 'package:your_app_name/main.dart'; // change 'your_app_name' to your project name
void main() {
  testWidgets('GreetingWidget shows correct text', (WidgetTester tester) async {
    // Build our widget and trigger a frame
    await tester.pumpWidget(const MaterialApp(
      home: GreetingWidget(name: 'Appalakonda'),
    ));
```

```
    // Verify that the text is displayed correctly
    expect(find.text('Hello, Appalakonda!'), findsOneWidget);
    expect(find.text('Hello, John!'), findsNothing);
  });
}
```

Step 3: Run the Test
Run the following command in your terminal:
flutter test

Output:

00:00 +1: All tests passed!

Aim:
b)Use Flutter's debugging tools to identify and fix issues.

Description:
This Flutter program demonstrates the use of debugging tools and assertions to track widget behavior and runtime state changes. The app defines a stateful widget that displays a counter and increments it when the button is pressed. Each time the counter is updated, the debugPrint() function logs messages to the debug console, showing rebuilds and counter changes. An assertion ensures that the counter never exceeds 5, triggering an error in debug mode if it does. Developers can use Flutter DevTools and the Inspector to analyze widget rebuilds, inspect layout issues, and monitor performance, making it easier to identify bugs and optimize the app during development.

Source Code:
Step 1: Flutter Program (main.dart)

```
import 'package:flutter/material.dart';
void main() {
  // Enable debugging print statements
  debugPrint('App starting...');
  runApp(MaterialApp(home: DebugDemo()));
}
class DebugDemo extends StatefulWidget {
  @override
  _DebugDemoState createState() => _DebugDemoState();
}
class _DebugDemoState extends State<DebugDemo> {
  int counter = 0;
  void _incrementCounter() {
    setState(() {
      counter++;
      // Debug print for tracking value changes
      debugPrint('Counter updated to: $counter');

      // Example assertion: Counter should never exceed 5
      assert(counter <= 5, 'Counter value exceeded 5!');
    });
  }
  @override
  Widget build(BuildContext context) {
    // Print widget rebuilds in console for debugging
    debugPrint('Widget rebuilt with counter = $counter');
    return Scaffold(
      appBar: AppBar(title: const Text('Debugging Demo')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

```
          Text(
            'Counter Value: $counter',
            style: const TextStyle(fontSize: 24),
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: _incrementCounter,
            child: const Text('Increment Counter'),
          ),
        ],
      ),
    ),
  );
 }
}
```

Step 2: Run and Debug

i.Run the app using:

flutter run

ii. Open Flutter DevTools:

flutter pub global run devtools

Then open the provided URL in your browser.

iii. In Flutter Inspector, you can:

- Select widgets to inspect layout issues.
- Check Rebuild Counts.
- View widget tree hierarchy.

iv. In the Debug Console, observe:

- debugPrint() outputs each rebuild and counter change.
- When counter > 5, an assertion error will appear (only in debug mode).

Output:

```
vbnet

App starting...
Widget rebuilt with counter = 0
Counter updated to: 1
Widget rebuilt with counter = 1
Counter updated to: 2
Widget rebuilt with counter = 2
...
Counter updated to: 6
═══╡ EXCEPTION CAUGHT BY FOUNDATION LIBRARY ╞═══════════════════════════
Assertion failed: 'Counter value exceeded 5!'
```

```
yaml

AppBar Title: Debugging Demo
Counter Value: 0  → (increases each button press)
Button: Increment Counter
```