**Ex No. 1**

**Date: 10/08/2023**

**Develop an application that uses GUI components, Font and Colors**

**Aim:** To create a Flutter application that utilizes GUI components, fonts, and colors for a visually appealing user interface.

**Procedure:**

* Create a new Flutter project.
* Design the GUI using Flutter's widget system, incorporating components like buttons, text fields, and images.
* Implement custom fonts by adding font files to the project and applying them to relevant text widgets.
* Experiment with color schemes by defining and using custom colors or choosing from Flutter's predefined colors.
* Ensure proper styling and alignment of GUI components for a cohesive visual experience.
* Test the application on different devices to ensure responsiveness and adaptability to various screen sizes.

**Code:**

import 'package:flutter/material.dart';

void main () => runApp(const MaterialApp(

home: Home()

));

class Home extends StatelessWidget {

const Home({super.key});

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text("Chennai Nightlife"),

centerTitle: true,

backgroundColor: Colors.black,

),

body: const Center(

child: Column(

children: [

Text(

"Nightlife Venues",

style: TextStyle(

fontFamily: "Montserrat",

fontSize: 40.0,

fontWeight: FontWeight.bold,

),

),

],

)

),

floatingActionButton: FloatingActionButton(

backgroundColor: const Color(0xFFff4c68),

child: const Text("Click"),

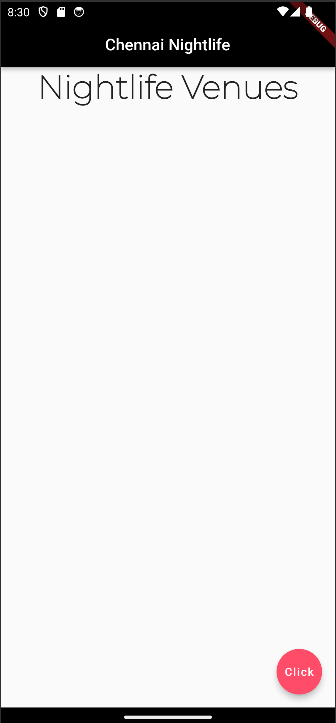
onPressed: () {},

),

);

}

}

**Output:**

**Result:** Code for a Flutter application with a visually appealing user interface that includes GUI components, custom fonts, and a well-thought-out color scheme has been written and executed.

**Ex No. 2**

**Date: 17/08/2023**

**Develop an application that uses Layout Managers and event listeners**

**Aim:** To develop a Flutter application that incorporates Layout Managers for effective widget positioning and utilizes event listeners to respond to user interactions.

**Procedure:**

* Utilize Flutter's layout widgets such as Column, Row, Stack, etc., to organize and position UI elements effectively.
* Implement event listeners by utilizing Flutter's gesture recognizers or callback functions for buttons and interactive components.
* Define specific actions or behaviors for different events, such as button clicks or gestures.
* Ensure the proper arrangement of widgets within the layout managers to achieve the desired user interface structure.
* Test the application thoroughly, ensuring that the layout managers work as intended and that events trigger the expected responses.

**Code:**

import 'dart:async';

import 'package:flutter/material.dart';

void main() => runApp(const MaterialApp(

home: Home(),

));

class Home extends StatelessWidget {

const Home({Key? key});

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text(

"Chennai Nightlife",

style: TextStyle(

fontFamily: "Montserrat",

fontWeight: FontWeight.w900,

),

),

centerTitle: true,

backgroundColor: Colors.black,

),

backgroundColor: Colors.grey[900],

body: SingleChildScrollView(

child: Padding(

padding: const EdgeInsets.all(8.0),

child: Column(

children: [

Text(

"Nightlife Venues",

style: TextStyle(

fontFamily: "Montserrat",

fontSize: 30.0,

fontWeight: FontWeight.bold,

color: Colors.white,

),

),

SizedBox(height: 8.0),

NightlifeCardRow(

cards: [

NightlifeCard(

image: AssetImage("assets/DJ1.jpg"),

name: "Venue 1",

address: "Address 1",

isFirstCard: true,

),

NightlifeCard(

image: AssetImage("assets/DJ2.jpg"),

name: "Venue 2",

address: "Address 2",

isFirstCard: true,

),

],

),

SizedBox(height: 8.0),

NightlifeCardRow(

cards: [

NightlifeCard(

image: AssetImage("assets/DJ3.jpg"),

name: "Venue 3",

address: "Address 3",

),

NightlifeCard(

image: AssetImage("assets/DJ4.jpg"),

name: "Venue 4",

address: "Address 4",

),

],

),

SizedBox(height: 8.0),

NightlifeCardRow(

cards: [

NightlifeCard(

image: AssetImage("assets/DJ5.jpg"),

name: "Venue 5",

address: "Address 5",

),

NightlifeCard(

image: AssetImage("assets/DJ6.jpg"),

name: "Venue 6",

address: "Address 6",

),

],

),

],

),

),

),

);

}

}

class NightlifeCardRow extends StatelessWidget {

final List<Widget> cards;

const NightlifeCardRow({required this.cards});

@override

Widget build(BuildContext context) {

return Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: cards.map((card) => Expanded(child: card)).toList(),

);

}

}

class NightlifeCard extends StatelessWidget {

final ImageProvider<Object> image;

final String name;

final String address;

final bool isFirstCard;

const NightlifeCard({

required this.image,

required this.name,

required this.address,

this.isFirstCard = false,

});

@override

Widget build(BuildContext context) {

double cardWidth = MediaQuery.of(context).size.width \* 0.45;

return Card(

color: Colors.grey[800],

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.vertical(

top: isFirstCard ? Radius.circular(16.0) : Radius.zero,

bottom: Radius.circular(16.0),

),

),

child: SizedBox(

width: cardWidth,

height: 250.0,

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

FutureBuilder(

future: \_getImageSize(image),

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.done &&

!snapshot.hasError) {

return ClipRRect(

borderRadius: BorderRadius.vertical(

top: isFirstCard ? Radius.circular(16.0) : Radius.zero,

),

child: Image(

image: image,

height: 150.0,

width: double.infinity,

fit: BoxFit.cover,

),

);

} else {

return Container(

height: 80.0,

color: Colors.grey,

child: Center(

child: Icon(Icons.error),

),

);

}

},

),

Padding(

padding: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 4.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(

name,

style: TextStyle(

fontWeight: FontWeight.bold,

fontSize: 18.0, // Increase the font size

color: Colors.white,

),

),

Text(

address,

style: TextStyle(

fontSize: 14.0, // Increase the font size

color: Colors.white,

),

),

],

),

),

Spacer(),

Container(

width: double.infinity,

color: const Color(0xFFff4c68),

child: TextButton(

onPressed: () {

// Add your booking logic here

},

child: Text(

"Book",

style: TextStyle(color: Colors.white),

),

),

),

],

),

),

);

}

Future<ImageInfo> \_getImageSize(ImageProvider<Object> imageProvider) {

Completer<ImageInfo> completer = Completer();

final ImageStream stream = imageProvider.resolve(ImageConfiguration.empty);

final listener = ImageStreamListener(

(ImageInfo info, bool synchronousCall) {

completer.complete(info);

},

onError: (dynamic exception, StackTrace? stackTrace) {

completer.completeError(exception);

},

);

stream.addListener(listener);

completer.future.then((\_) {

stream.removeListener(listener);

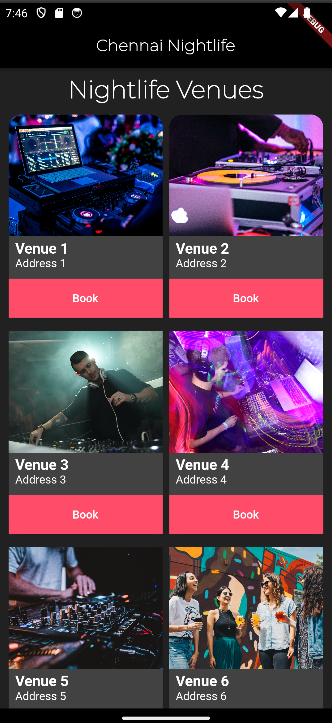
});

return completer.future;

}

}

**Output:**

****

**Result:** Code for a Flutter application with a well-organized user interface using layout managers for optimal widget positioning and user interactions was written and executed.

**Ex No. 3**

**Date: 24/08/23**

**Develop a native calculator application**

**Aim:** To develop a native calculator application using Flutter for basic arithmetic operations.

**Procedure:**

* Design the calculator's user interface with buttons for digits (0-9) and operators (+, -, \*, /).
* Implement logic to handle button clicks and update the display accordingly.
* Create functions for basic arithmetic operations (addition, subtraction, multiplication, division).
* Ensure the calculator follows the order of operations and provides accurate results.
* Implement a clear button to reset the calculator's state.
* Test the calculator thoroughly with various input scenarios to ensure accurate calculations.
* Fine-tune the user interface for a clean and intuitive user experience.

**Code:**

import 'package:flutter/material.dart';

void main() {

runApp(CalculatorApp());

}

class CalculatorApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: CalculatorScreen(),

);

}

}

class CalculatorScreen extends StatefulWidget {

@override

\_CalculatorScreenState createState() => \_CalculatorScreenState();

}

class \_CalculatorScreenState extends State<CalculatorScreen> {

String \_output = "0";

String \_currentInput = "";

double? \_num1; // Add "?" to indicate it can be null

String? \_operation; // Add "?" to indicate it can be null

void \_onNumberPressed(String value) {

setState(() {

if (\_currentInput.length < 10) {

if (\_currentInput == "0") {

\_currentInput = value;

} else {

\_currentInput += value;

}

}

});

}

void \_onOperationPressed(String value) {

setState(() {

\_num1 = double.parse(\_currentInput);

\_operation = value;

\_currentInput = "";

});

}

void \_onEqualPressed() {

setState(() {

if (\_operation != null && \_currentInput.isNotEmpty) {

double num2 = double.parse(\_currentInput);

double result;

switch (\_operation) {

case "+":

result = \_num1! + num2; // Add "!" to assert that \_num1 is not null

break;

case "-":

result = \_num1! - num2; // Add "!" to assert that \_num1 is not null

break;

case "\*":

result = \_num1! \* num2; // Add "!" to assert that \_num1 is not null

break;

case "/":

result = \_num1! / num2; // Add "!" to assert that \_num1 is not null

break;

}

\_output = result.toString();

\_num1 = null;

\_operation = null;

\_currentInput = \_output;

}

});

}

void \_onClearPressed() {

setState(() {

\_output = "0";

\_currentInput = "";

\_num1 = null;

\_operation = null;

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text("Calculator"),

),

body: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Container(

padding: EdgeInsets.all(16.0),

alignment: Alignment.centerRight,

child: Text(

\_currentInput.isEmpty ? "0" : \_currentInput,

style: TextStyle(fontSize: 36.0),

),

),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

\_buildButton("7"),

\_buildButton("8"),

\_buildButton("9"),

\_buildButton("/"),

],

),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

\_buildButton("4"),

\_buildButton("5"),

\_buildButton("6"),

\_buildButton("\*"),

],

),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

\_buildButton("1"),

\_buildButton("2"),

\_buildButton("3"),

\_buildButton("-"),

],

),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

\_buildButton("0"),

\_buildButton("C"),

\_buildButton("="),

\_buildButton("+"),

],

),

],

),

);

}

Widget \_buildButton(String text) {

return ElevatedButton(

onPressed: () {

if (text == "C") {

\_onClearPressed();

} else if (text == "=") {

\_onEqualPressed();

} else if (text == "+" || text == "-" || text == "\*" || text == "/") {

\_onOperationPressed(text);

} else {

\_onNumberPressed(text);

}

},

child: Text(

text,

style: TextStyle(fontSize: 20.0),

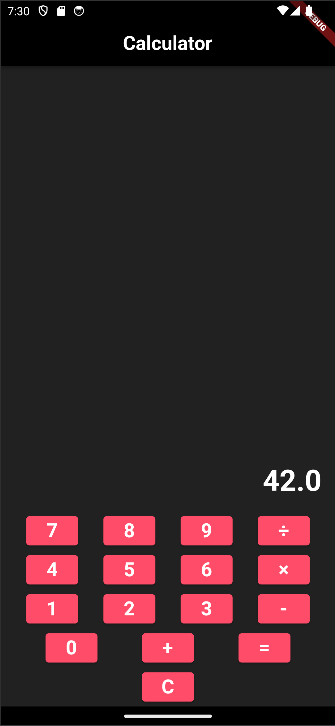
),

);

}

}

**Output:**

****

**Result:** A native calculator application was built using Flutter that allows users to perform basic arithmetic operations.

**Ex No. 4**

**Date: 31/08/23**

**Write an application that draws basic graphical primitives on the screen**

**Aim:** To develop a Flutter application that draws basic graphical primitives on the screen, such as lines, circles, and rectangles.

**Procedure:**

* Use a CustomPainter widget to define a custom painting canvas.
* Implement drawing functions for basic primitives, including lines, circles, and rectangles, within the CustomPainter class.
* Utilize the Canvas API to draw the graphical elements with specified dimensions, positions, and styles.
* Update the UI to incorporate the custom-painted canvas within the main application widget.
* Implement any necessary user interactions, such as tapping or dragging, to dynamically draw or modify the graphical primitives.
* Test the application to ensure proper rendering and responsiveness of the drawn elements.
* Refine the user interface for a visually appealing and intuitive experience.

**Code:**

import 'package:flutter/material.dart';

import 'dart:math' as math;

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({Key? key}) : super(key: key);

// This widget is the root of your application.

static const String \_title= 'Flutter Sample';

@override

Widget build(BuildContext context) {

return const MaterialApp(

title:\_title,

home:StatefulShapes(),

);

}

}

class StatefulShapes extends StatefulWidget {

const StatefulShapes({Key? key}) : super(key: key);

@override

State<StatefulShapes> createState() => \_StatefulShapesState();

}

class \_StatefulShapesState extends State<StatefulShapes> {

int \_selectedIndex=0;

static const List<Widget> \_widgetOptions=<Widget>[

CustomPaint(

size:Size(300,200),

painter: LinePainter(),

),

CustomPaint(

size:Size(300,200),

painter: CirclePainter(),

),

CustomPaint(

size:Size(300,200),

painter: TrianglePainter(),

)

];

void \_onItemTapped(int index){

setState((){

\_selectedIndex=index;

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar:AppBar(

title:const Text("Custom Paint Demo"),

),

body:Center(

child:\_widgetOptions.elementAt(\_selectedIndex),

),

bottomNavigationBar: BottomNavigationBar(

items:const<BottomNavigationBarItem>[

BottomNavigationBarItem(

icon: Icon(Icons.horizontal\_rule),

label:'Line'

),

BottomNavigationBarItem(

icon: Icon(Icons.circle),

label:'Circle'),

BottomNavigationBarItem(

icon: Icon(Icons.change\_history),

label:'Triangle'),

],

currentIndex: \_selectedIndex,

selectedItemColor: Colors.blue,

onTap: \_onItemTapped,

),

);

}

}

class LinePainter extends CustomPainter {

const LinePainter() ;

@override

void paint(Canvas canvas,Size size){

var paint = Paint()

..color = Colors.teal

..strokeWidth = 15;

Offset start=Offset(0,size.height/2);

Offset end=Offset(size.width,size.height/2);

canvas.drawLine(start,end,paint);

}

@override

bool shouldRepaint(covariant CustomPainter oldDelegate) {

return false;

}

}

class CirclePainter extends CustomPainter{

const CirclePainter() ;

@override

void paint(Canvas canvas,Size size){

var paint=Paint()

..color=Colors.teal

..strokeWidth=5

..style=PaintingStyle.stroke

..strokeCap=StrokeCap.round;

Offset center=Offset(size.width/2,size.height/2);

canvas.drawCircle(center,100,paint);

}

@override

bool shouldRepaint(covariant CustomPainter oldDelegate){

return false;

}

}

class TrianglePainter extends CustomPainter{

const TrianglePainter() ;

@override

void paint(Canvas canvas, Size size) {

// TODO: implement paint

var paint=Paint()

..color=Colors.teal

..strokeWidth=5

..style=PaintingStyle.stroke

..strokeCap=StrokeCap.round;

var path=Path();

var angle=(math.pi \*2)/3;

var radius=100;

Offset center=Offset(size.width/2,size.height/2);

Offset startPoint=Offset(radius\*math.cos(0.0),radius\*math.sin(0.0));

path.moveTo(startPoint.dx+center.dx,startPoint.dy+center.dy);

for(int i=1;i<=3;i++){

double x=radius\*math.cos(angle\*i)+center.dx;

double y=radius\*math.sin(angle\*i)+center.dy;

path.lineTo(x,y);

}

path.close();

canvas.drawPath(path,paint);

}

@override

bool shouldRepaint(covariant CustomPainter oldDelegate) {

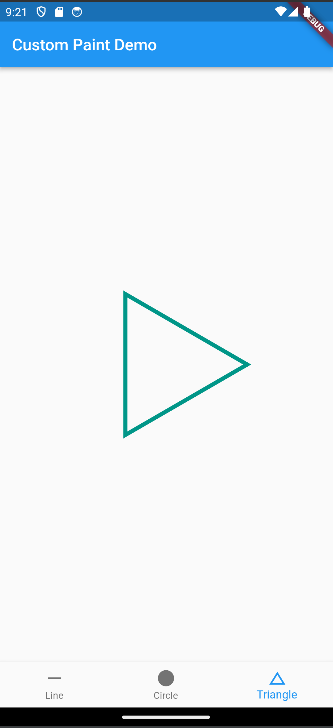
// TODO: implement shouldRepaint

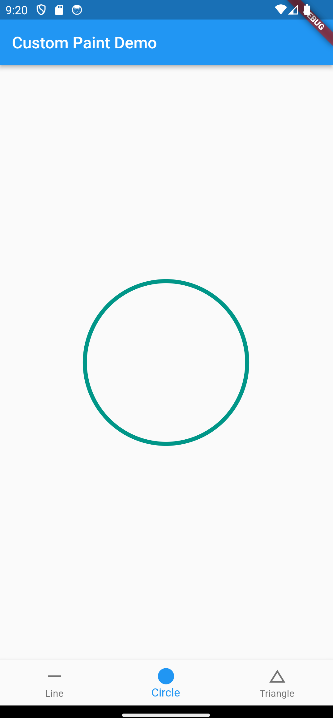
return false;

}

}

**Output:**





**Result:** Code for a Flutter application to display a custom-painted canvas with basic graphical primitives was written and executed.

**Ex No. 5**

**Date: 07/09/23**

**Develop an application that makes use of database**

**Aim:** To develop a Flutter application that integrates a database to store and retrieve data.

**Procedure:**

* Choose a database solution compatible with Flutter, such as SQLite, Firebase, or Moor.
* Configure the database connection and initialize the necessary tables or collections.
* Design the user interface to collect and display data, including input forms and data presentation widgets.
* Implement logic to interact with the database, including functions for inserting, updating, deleting, and querying data.
* Ensure proper error handling for database operations to maintain data integrity.
* Test the application to verify the successful storage and retrieval of data from the database.
* Fine-tune the user interface and functionality for a seamless user experience.

**Code:**

// profile.dart

import 'package:flutter/material.dart';

import 'main.dart'; // Assuming your main.dart is in the same directory

class ProfilePage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text(

"Profile",

style: TextStyle(

fontFamily: "Montserrat",

fontWeight: FontWeight.w900,

),

),

centerTitle: true,

backgroundColor: Colors.black,

automaticallyImplyLeading: false, // Disable the back button

),

backgroundColor: Colors.grey[900],

body: Align(

alignment: Alignment.topCenter,

child: Padding(

padding: const EdgeInsets.symmetric(horizontal: 16.0),

child: ProfileCard(

name: "Danush Gopinath",

email: "danushgopinath8502@gmail.com",

dateJoined: "November 16, 2023", // You can update this with the actual date

),

),

),

bottomNavigationBar: BottomNavigationBar(

currentIndex: 1, // Set the current index to 1 for the profile icon to be active

backgroundColor: Colors.black,

selectedItemColor: const Color(0xFFff4c68),

unselectedItemColor: Colors.white,

items: const [

BottomNavigationBarItem(

icon: Icon(Icons.home),

label: 'Home',

),

BottomNavigationBarItem(

icon: Icon(Icons.person),

label: 'Profile',

),

],

onTap: (index) {

if (index == 0) {

Navigator.pushReplacement(

context,

MaterialPageRoute(builder: (context) => Home()),

);

}

},

),

);

}

}

class ProfileCard extends StatelessWidget {

final String name;

final String email;

final String dateJoined;

const ProfileCard({

required this.name,

required this.email,

required this.dateJoined,

});

@override

Widget build(BuildContext context) {

return Card(

color: Colors.grey[800],

margin: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 16.0),

child: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

mainAxisSize: MainAxisSize.min,

children: [

Text(

"Name: ",

style: TextStyle(

fontWeight: FontWeight.bold,

fontSize: 18.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

Text(

name,

style: TextStyle(

fontSize: 16.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

SizedBox(height: 8.0),

Text(

"Email: ",

style: TextStyle(

fontWeight: FontWeight.bold,

fontSize: 18.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

Text(

email,

style: TextStyle(

fontSize: 16.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

SizedBox(height: 8.0),

Text(

"Date Joined: ",

style: TextStyle(

fontWeight: FontWeight.bold,

fontSize: 18.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

Text(

dateJoined,

style: TextStyle(

fontSize: 16.0,

color: Colors.white,

fontFamily: "Montserrat",

),

),

],

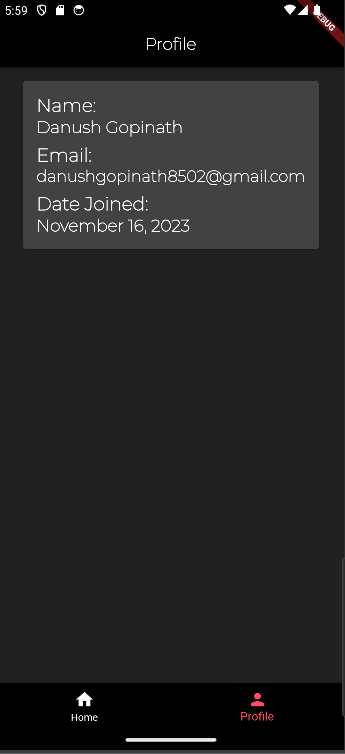
),

),

);

}

}

**Output:**

**Result:** Code for a Flutter application that effectively integrates a database for data storage and retrieval was written and executed.

**Ex No. 6**

**Date: 14/09/23**

**Develop an application that makes use of RSS Feed**

**Aim:** To develop a Flutter application that utilizes an RSS feed to display dynamic content.

**Procedure:**

* Choose a package or library for handling RSS feeds in Flutter (e.g., webfeed package).
* Fetch and parse the RSS feed data using the chosen package.
* Design the user interface to present the fetched data, considering elements like titles, descriptions, and publication dates.
* Implement a mechanism to refresh or update the RSS feed content to display the latest information.
* Handle errors gracefully, providing feedback to users in case of issues with fetching or parsing the RSS feed.
* Ensure a responsive and visually appealing layout for the displayed RSS feed content.
* Test the application with different RSS feeds to ensure compatibility and proper rendering.

**Code:**

import 'package:flutter/material.dart';

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

import 'package:webfeed/webfeed.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'RSS Feed Reader',

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

List<RssItem> \_feedItems = [];

@override

void initState() {

super.initState();

\_fetchFeed();

}

Future<void> \_fetchFeed() async {

final response = await http.get(Uri.parse('https://blog.google/products/news/rss'));

if (response.statusCode == 200) {

final feed = RssFeed.parse(response.body);

setState(() {

\_feedItems = feed.items ?? [];

});

} else {

throw Exception('Failed to load RSS feed');

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('RSS Feed Reader'),

),

body: \_buildFeedList(),

);

}

Widget \_buildFeedList() {

if (\_feedItems.isEmpty) {

return Center(

child: CircularProgressIndicator(),

);

} else {

return ListView.builder(

itemCount: \_feedItems.length,

itemBuilder: (context, index) {

final item = \_feedItems[index];

return ListTile(

title: Text(item.title ?? 'No Title'),

subtitle: Text(item.pubDate?.toString() ?? 'No Date'),

);

},

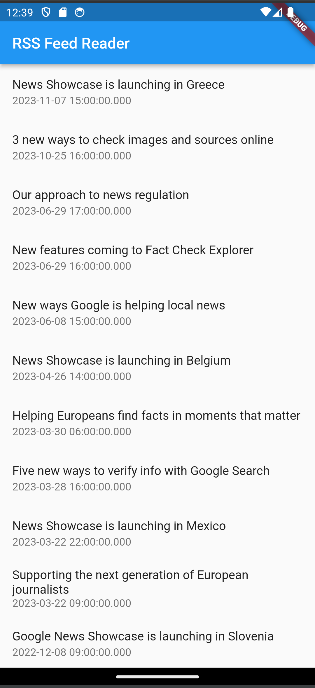
);

}

}

}

**Output:**

****

**Result:** Code for a Flutter application that effectively fetches and displays content from an RSS feed was written and executed.

**Ex No. 7**

**Date: 21/09/23**

**Implement an application that implements multi-threading**

**Aim:** To implement a Flutter application that utilizes multi-threading for improved performance and responsiveness.

**Procedure:**

* Identify tasks within the application that can benefit from multi-threading, such as data processing or background tasks.
* Utilize Dart's Isolate class or Flutter plugins like compute to run computationally intensive tasks on separate threads.
* Design the application's architecture to handle concurrent execution of tasks without affecting the main thread.
* Implement synchronization mechanisms, such as locks or message passing, to coordinate communication between threads.
* Test the application to ensure that multi-threading enhances performance, especially for tasks that run in the background.
* Address any potential issues related to thread safety and data consistency.
* Fine-tune the user interface for a smooth and responsive user experience.

**Code:**

import 'dart:async';

import 'dart:isolate';

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

String result = 'Result will be shown here';

void \_startTask() async {

setState(() {

result = 'Running task...';

});

// Start the time-consuming task in a separate isolate

final receivePort = ReceivePort();

await Isolate.spawn(\_backgroundTask, receivePort.sendPort);

// Listen for results from the isolate

final resultFromIsolate = Completer<String>();

receivePort.listen((data) {

resultFromIsolate.complete(data);

receivePort.close();

});

final finalResult = await resultFromIsolate.future;

setState(() {

result = finalResult;

});

}

static void \_backgroundTask(SendPort sendPort) {

// Simulate a time-consuming task

Future.delayed(Duration(seconds: 5), () {

sendPort.send('Task completed!');

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Multi-Threading Flutter App'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text(

result,

style: TextStyle(fontSize: 18),

),

SizedBox(height: 20),

ElevatedButton(

onPressed: \_startTask,

child: Text('Start Task'),

),

],

),

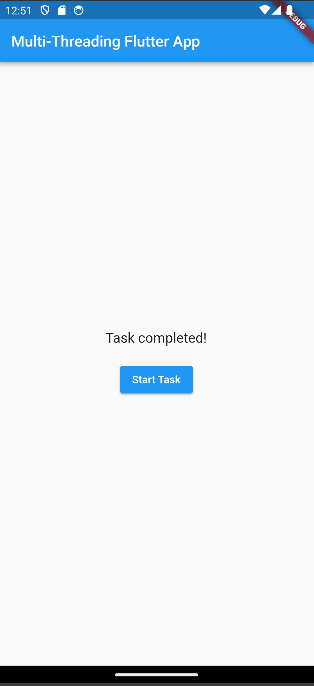
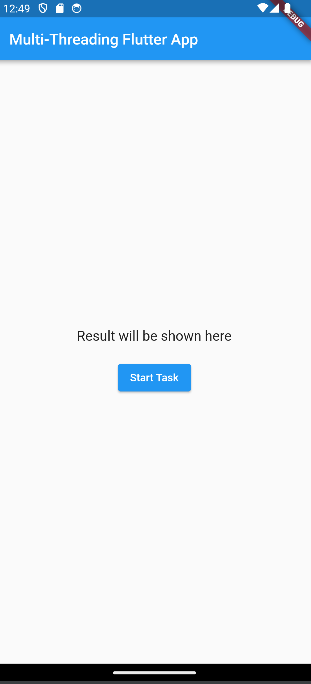
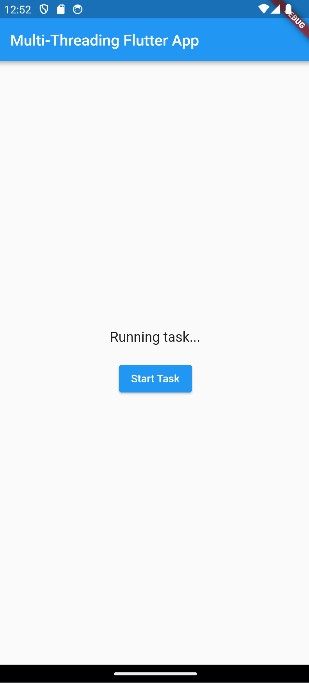
),

);

}

}

**Output:**

****

**Result:** Code for a Flutter application that leverages multi-threading to enhance performance, especially for computationally intensive or background tasks was written and executed.

**Ex No. 8**

**Date: 28/09/23**

**Develop a native application that uses GPS location information**

**Aim:** To develop a Flutter native application that utilizes GPS location information to provide location-based features.

**Procedure:**

* Integrate the appropriate Flutter plugin (e.g., geolocator) for accessing GPS location information.
* Request necessary permissions to access the device's location.
* Design the user interface to display current location information and relevant features based on location.
* Implement logic to fetch and update the device's GPS coordinates using the plugin.
* Utilize the obtained location data to offer location-based services or features, such as mapping or distance calculations.
* Handle scenarios where location services might be disabled or permissions are denied.
* Test the application on different devices to ensure accurate and reliable GPS location information.
* Optimize the application for power efficiency by considering location updates frequency and accuracy settings.

**Code:**

import 'package:flutter/material.dart';

import 'package:geolocator/geolocator.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Location App',

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

Position? \_currentPosition;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Location App'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text(

'Current Location:',

style: TextStyle(fontSize: 20),

),

SizedBox(height: 10),

if (\_currentPosition != null)

Text(

'Latitude: ${\_currentPosition!.latitude}\nLongitude: ${\_currentPosition!.longitude}',

style: TextStyle(fontSize: 16),

)

else

Text(

'Location not available',

style: TextStyle(fontSize: 16),

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

\_getCurrentLocation();

},

child: Text('Get Location'),

),

],

),

),

);

}

void \_getCurrentLocation() async {

try {

Position position = await Geolocator.getCurrentPosition(

desiredAccuracy: LocationAccuracy.high);

setState(() {

\_currentPosition = position;

});

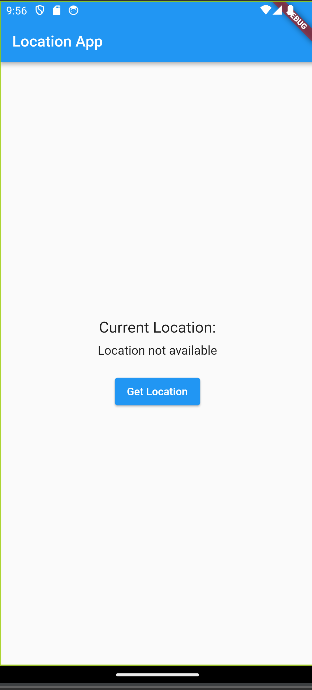
} catch (e) {

print(e);

}

}

}

**Output:**

**Result:** Code for a Flutter native application that successfully accesses and utilizes GPS location information was written and executed.

**Ex No. 9**

**Date: 05/11/23**

**Implement an application that writes data to the SD card**

**Aim:** To implement a Flutter application that writes data to the SD card for data storage or export purposes.

**Procedure:**

* Check and request necessary permissions to write to external storage (SD card).
* Utilize Dart's file I/O operations to create, write, and manage files on the SD card.
* Design the user interface to allow users to input data or select data to be written to the SD card.
* Implement logic to handle the data processing and write the formatted data to a file on the SD card.
* Ensure proper error handling for scenarios like insufficient storage space or write permissions denied.
* Provide feedback to the user upon successful data writing or in case of errors.
* Test the application on devices with SD card support to ensure compatibility.
* Optimize the application for efficient data writing, considering factors like asynchronous operations for responsiveness.

**Code:**

import 'dart:io';

import 'package:flutter/material.dart';

import 'package:path\_provider/path\_provider.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

TextEditingController \_textController = TextEditingController();

Future<void> \_writeToFile(String content) async {

try {

// Get the external storage directory

Directory? directory = await getExternalStorageDirectory();

if (directory != null) {

// Specify the file path

String filePath = '${directory.path}/my\_file.txt';

// Open the file and write the content

File file = File(filePath);

await file.writeAsString(content);

// Show a success message

showDialog(

context: context,

builder: (context) => AlertDialog(

title: Text('Success'),

content: Text('Data written to SD card successfully.'),

actions: [

TextButton(

onPressed: () => Navigator.pop(context),

child: Text('OK'),

),

],

),

);

} else {

// Handle the case when the directory is null

print('Error: External storage directory is null.');

}

} catch (e) {

print('Error writing to file: $e');

// Show an error message

showDialog(

context: context,

builder: (context) => AlertDialog(

title: Text('Error'),

content: Text('Failed to write data to SD card.'),

actions: [

TextButton(

onPressed: () => Navigator.pop(context),

child: Text('OK'),

),

],

),

);

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Write to SD Card'),

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

TextField(

controller: \_textController,

decoration: InputDecoration(labelText: 'Enter data to write'),

),

SizedBox(height: 16.0),

ElevatedButton(

onPressed: () {

String data = \_textController.text;

if (data.isNotEmpty) {

\_writeToFile(data);

}

},

child: Text('Write to SD Card'),

),

],

),

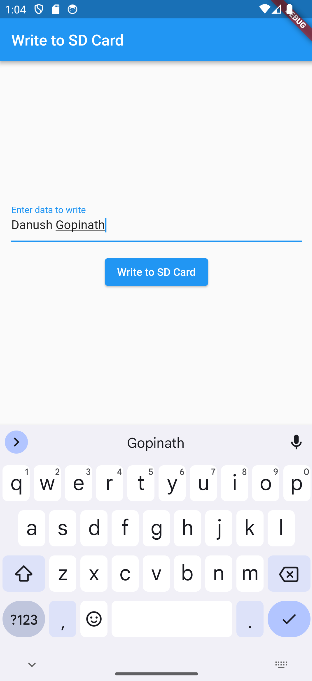
),

);

}

}

**Output:**

****

**This example creates a simple Flutter application with a text field and a button. Users can enter data into the text field and press the button to write the data to a file on the SD card.**

**Result:** Code for a Flutter application that allows users to write data to the SD card was written and executed.

**Ex No. 10**

**Date: 12/10/23**

**Implement an application that creates an alert upon receiving a message**

**Aim:** To implement a Flutter application that creates an alert when a message is received.

**Procedure:**

* Design the user interface to include a notification area or a space where alerts can be displayed.
* Implement a messaging mechanism, either through a simulated event or by integrating a messaging service (e.g., Firebase Cloud Messaging).
* Set up listeners to detect incoming messages or events that trigger the alert.
* Design the alert component or utilize Flutter's built-in AlertDialog to create visually appealing and informative alerts.
* Integrate logic to display the alert when a message is received.
* Provide options for the user to interact with the alert, such as dismissing it or taking specific actions.
* Test the application by simulating message events or using a messaging service to trigger alerts.
* Optimize the alert system for responsiveness and user engagement.

**Code:**

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

TextEditingController \_messageController = TextEditingController();

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Message Alert App'),

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

TextField(

controller: \_messageController,

decoration: InputDecoration(labelText: 'Enter a message'),

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

\_showAlert(\_messageController.text);

},

child: Text('Show Alert'),

),

],

),

),

);

}

void \_showAlert(String message) {

showDialog(

context: context,

builder: (BuildContext context) {

return AlertDialog(

title: Text('Alert'),

content: Text(message),

actions: [

TextButton(

onPressed: () {

Navigator.of(context).pop();

},

child: Text('OK'),

),

],

);

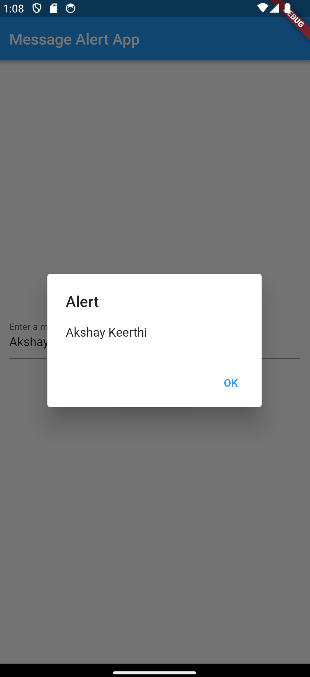
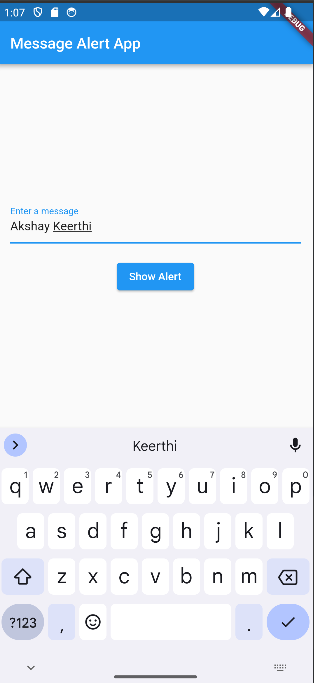
},

);

}

}

**Output:**



**Result:** Code for a Flutter application that effectively creates alerts upon receiving messages was written and executed.

**Ex No. 11**

**Date: 19/10/23**

**Write a mobile application that creates an alarm clock**

**Aim:** To develop a Flutter mobile application that functions as an alarm clock.

**Procedure:**

* Design the user interface to include components for setting alarm time, enabling/disabling alarms, and displaying the current time.
* Utilize Flutter's DateTime class or a time-related package to handle time-related functionalities.
* Implement a mechanism to set and store alarm times, considering data persistence for alarm settings.
* Integrate the necessary audio or notification functionalities to create alarm alerts.
* Develop logic to trigger alarms at the specified times, even if the application is in the background.
* Allow users to manage and edit existing alarms, including enabling, disabling, or deleting them.
* Implement snooze and dismiss functionalities for when alarms are triggered.
* Test the application thoroughly, ensuring alarms work as expected in various scenarios.
* Optimize the application for power efficiency, especially during background operations.

**Code:**

import 'dart:async';

import 'package:flutter/material.dart';

import 'package:intl/intl.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyClock(),

);

}

}

class MyClock extends StatefulWidget {

@override

\_MyClockState createState() => \_MyClockState();

}

class \_MyClockState extends State<MyClock> {

String \_timeString = ''; // Initialize with an empty string

@override

void initState() {

super.initState();

\_updateTime();

Timer.periodic(Duration(seconds: 1), (Timer t) => \_updateTime());

}

void \_updateTime() {

setState(() {

\_timeString = DateFormat('hh:mm:ss a').format(DateTime.now());

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Flutter Clock'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Text(

\_timeString,

style: TextStyle(fontSize: 48.0),

),

],

),

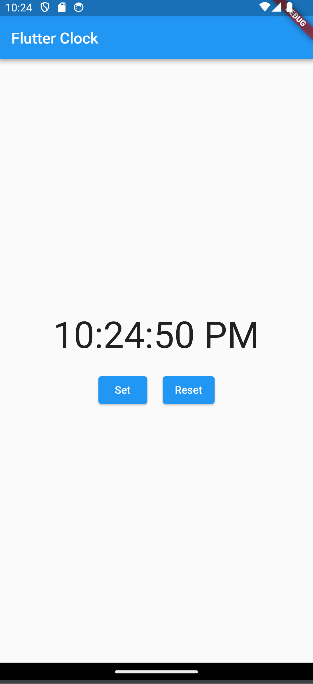
),

);

}

}

**Output:**

****

**Result:** Code for a Flutter mobile application that serves as an alarm clock, allowing users to set, manage, and be alerted by alarms at specified times was written and executed.

**Ex No. 12**

**Date: 26/10/23**

**Develop a simple gaming application with multimedia support**

**Aim:** To develop a simple Flutter gaming application with multimedia support, including audio and visual elements.

**Procedure:**

* Design the game's user interface, incorporating graphics, buttons, and other interactive elements.
* Integrate multimedia assets such as images, animations, and sound files for a rich gaming experience.
* Implement game logic, defining rules, objectives, and interactive elements based on user input.
* Utilize Flutter's animation framework for creating dynamic and engaging visual effects.
* Integrate audio features to enhance the gaming experience, including background music, sound effects, and interactive audio cues.
* Implement touch or gesture controls for user interactions within the game.
* Test the game thoroughly to ensure smooth multimedia playback, responsiveness, and adherence to game rules.
* Optimize the application for performance, considering factors like resource management and efficient rendering.

**Code:**

import 'dart:async';

import 'package:flutter/material.dart';

import 'package:audioplayers/audioplayers.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Memory Card Game',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: MemoryCardGame(),

);

}

}

class MemoryCardGame extends StatefulWidget {

@override

\_MemoryCardGameState createState() => \_MemoryCardGameState();

}

class \_MemoryCardGameState extends State<MemoryCardGame> {

List<String> cardImages = [

'assets/Club4.jpg',

'assets/Club5.jpg',

'assets/Club6.jpg',

'assets/Club-1.jpg',

'assets/Club-2.jpg',

'assets/Club-3.jpg',

'assets/hidden.jpg',

];

List<String> shuffledCards = [];

List<String> visibleCards = [];

bool isBusy = false;

AudioPlayer audioPlayer = AudioPlayer();

@override

void initState() {

super.initState();

loadCards();

}

void loadCards() {

shuffledCards = List.from(cardImages)..addAll(cardImages);

shuffledCards.shuffle();

visibleCards = List.filled(shuffledCards.length, 'hidden');

}

void playSound(String soundPath) async {

int result = await audioPlayer.play(soundPath, isLocal: true);

if (result != 1) {

print('Error playing sound');

}

}

void flipCard(int index) {

if (!isBusy && visibleCards[index] == 'hidden') {

setState(() {

visibleCards[index] = shuffledCards[index];

});

if (visibleCards.where((card) => card != 'hidden').length % 2 == 0) {

checkMatch();

}

}

}

void checkMatch() {

isBusy = true;

Timer(Duration(seconds: 1), () {

int firstIndex = visibleCards.indexWhere((card) => card != 'hidden');

int secondIndex = visibleCards.lastIndexOf(visibleCards[firstIndex]);

if (shuffledCards[firstIndex] == shuffledCards[secondIndex]) {

playSound('sounds/match.mp3');

} else {

playSound('sounds/no\_match.mp3');

setState(() {

visibleCards[firstIndex] = 'hidden';

visibleCards[secondIndex] = 'hidden';

});

}

isBusy = false;

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Memory Card Game'),

),

body: GridView.builder(

gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

crossAxisCount: 4,

),

itemCount: shuffledCards.length,

itemBuilder: (context, index) {

return GestureDetector(

onTap: () {

flipCard(index);

},

child: Card(

child: Image.asset(

visibleCards[index], // Use the visible card directly

fit: BoxFit.cover,

),

),

);

},

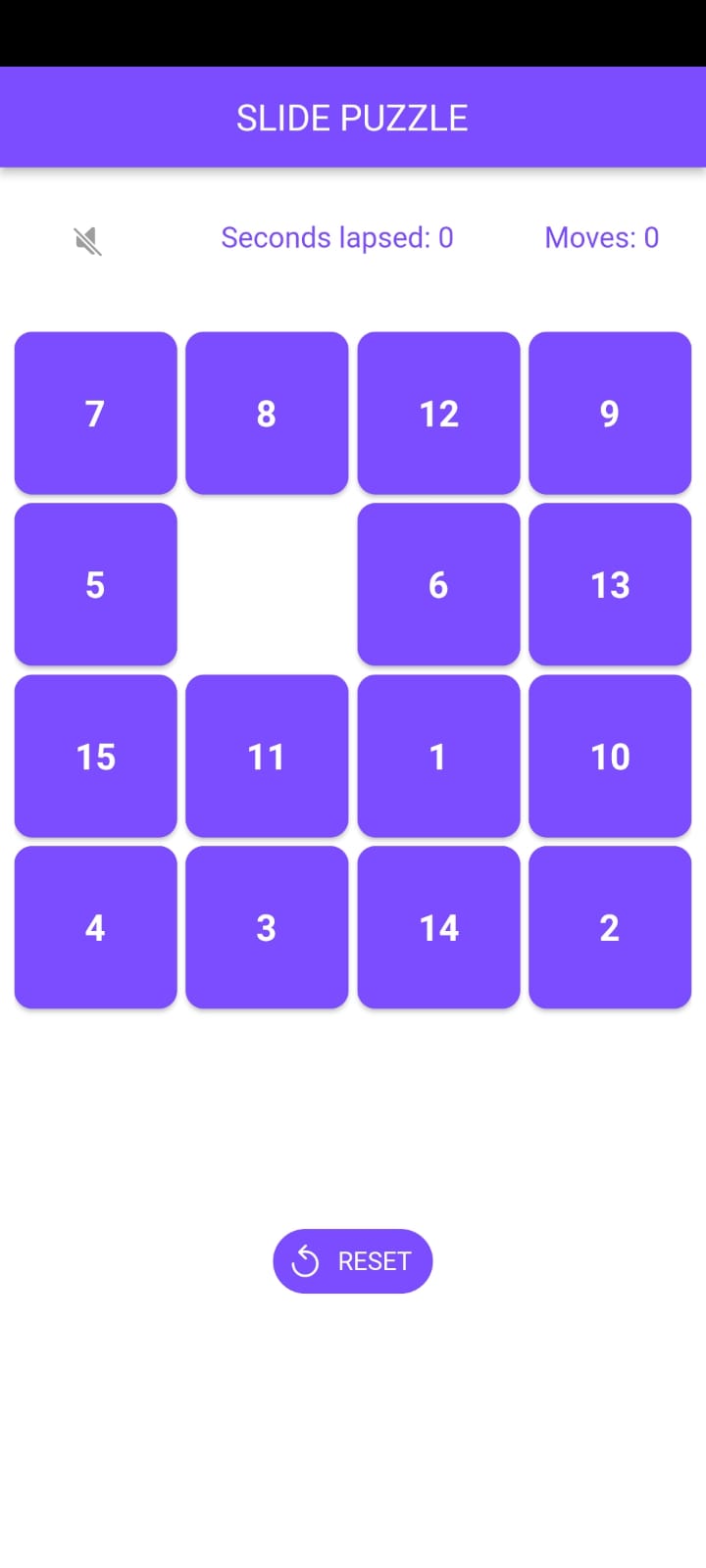
),

);

}

}

**Output:**



**Result:** Code for a Flutter gaming application with multimedia support, providing an engaging and interactive experience for users was written and execute

**Ex No. 13**

**Date: 02/11/23**

**Write a mobile application for data handling and connectivity via SOAP or REST to backend services potentially hosted in a cloud environment**

**Aim:** To develop a Flutter mobile application for data handling and connectivity via SOAP or REST to backend services, potentially hosted in a cloud environment.

**Procedure:**

* Design the user interface to include components for data input, display, and interaction.
* Integrate Flutter plugins for SOAP or REST communication (e.g., http package for REST).
* Implement logic for handling data requests, including authentication and authorization if required.
* Design and implement functions for data manipulation, such as fetching, updating, or deleting records.
* Configure connectivity to backend services, potentially hosted in a cloud environment (e.g., AWS, Firebase).
* Handle data serialization and deserialization for communication with the backend services.
* Implement error handling mechanisms for network requests and data processing.
* Test the application with various scenarios, including different network conditions and backend responses.
* Optimize the application for responsiveness, ensuring efficient data handling and a smooth user experience.

**Code:**

import 'package:flutter/material.dart';

import 'package:testlab/soap\_api\_service.dart';

import 'package:testlab/api\_service.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

final restApiService = ApiService('https://your-rest-api-endpoint.com');

final soapApiService = SoapApiService('https://your-soap-api-endpoint.com');

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(

title: Text('Flutter API Example'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ElevatedButton(

onPressed: () async {

final restData = await restApiService.fetchData('rest-endpoint');

print('REST Data: $restData');

},

child: Text('Fetch REST Data'),

),

ElevatedButton(

onPressed: () async {

final soapData = await soapApiService.fetchData('<soap-envelope>...</soap-envelope>');

print('SOAP Data: $soapData');

},

child: Text('Fetch SOAP Data'),

),

],

),

),

),

);

}

}

import 'dart:convert';

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

class ApiService {

final String baseUrl;

ApiService(this.baseUrl);

Future<Map<String, dynamic>> fetchData(String endpoint) async {

final response = await http.get(Uri.parse('$baseUrl/$endpoint'));

if (response.statusCode == 200) {

return json.decode(response.body);

} else {

throw Exception('Failed to load data');

}

}

}

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

import 'package:xml2json/xml2json.dart';

import 'dart:convert';

class SoapApiService {

final String baseUrl;

SoapApiService(this.baseUrl);

Future<Map<String, dynamic>> fetchData(String soapBody) async {

final headers = {

'Content-Type': 'text/xml',

};

final response = await http.post(

Uri.parse(baseUrl),

headers: headers,

body: soapBody,

);

if (response.statusCode == 200) {

final xml2json = Xml2Json();

xml2json.parse(response.body);

final jsonString = xml2json.toParker();

return json.decode(jsonString);

} else {

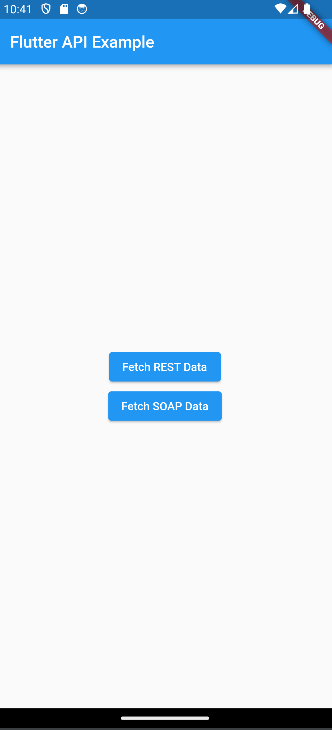
throw Exception('Failed to load data');

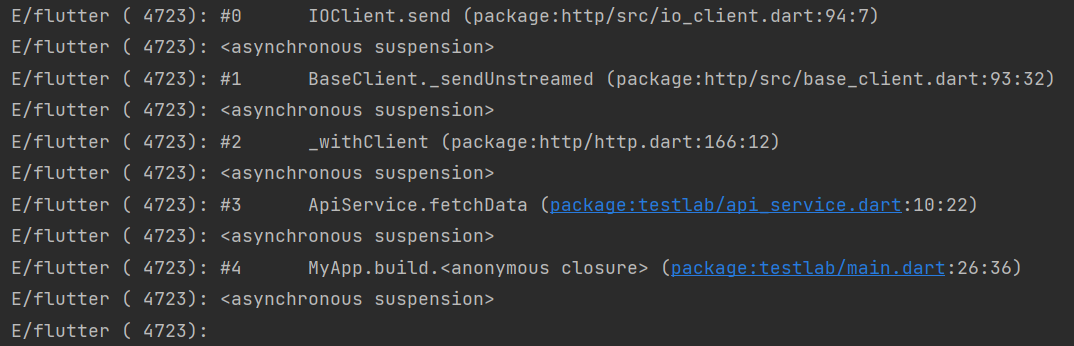
}

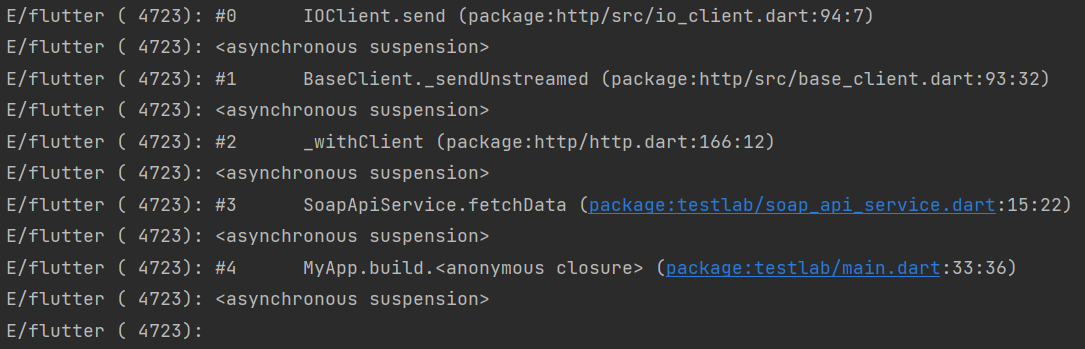
}

}

**Output:**







**Result:** Code for a Flutter mobile application that effectively handles data connectivity via SOAP or REST to backend services hosted in a cloud environment was written and executed.

**Ex No. 14**

**Date: 09/11/23**

**Write a mobile application that will take advantage of underlying phone functionality including GEO positioning, accelerometer, and rich gesture-based UI handling.**

**Aim:** To develop a Flutter mobile application that leverages underlying phone functionality, including GEO positioning, accelerometer, and rich gesture-based UI handling.

* **Procedure:**
* Design the user interface with components that utilize rich gestures, such as swiping, tapping, and dragging.
* Integrate Flutter plugins for accessing GEO positioning (e.g., geolocator) and accelerometer data.
* Implement logic to capture and display real-time GEO positioning information on the interface.
* Utilize accelerometer data for interactive features, such as tilting or shaking the device to trigger actions.
* Implement gesture-based UI handling, incorporating Flutter's gesture recognizers for a responsive user experience.
* Design features that dynamically respond to changes in GEO positioning, accelerometer data, and user gestures.
* Test the application on real devices to ensure accurate GEO positioning and accelerometer functionality.
* Fine-tune the UI and gestures for a seamless and engaging user experience.
* Optimize the application for performance, considering resource usage and responsiveness.

**Code:**

import 'package:flutter/material.dart';

import 'package:geolocator/geolocator.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

String \_locationData = 'Waiting for location...';

@override

void initState() {

super.initState();

\_getLocation();

}

Future<void> \_getLocation() async {

Position position = await Geolocator.getCurrentPosition(

desiredAccuracy: LocationAccuracy.high);

setState(() {

\_locationData =

'Latitude: ${position.latitude}\nLongitude: ${position.longitude}';

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Location App'),

),

body: Center(

child: Text(\_locationData),

),

);

}

}

import 'package:flutter/material.dart';

import 'package:sensors/sensors.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

String \_accelerometerData = 'Waiting for accelerometer data...';

@override

void initState() {

super.initState();

\_startAccelerometer();

}

void \_startAccelerometer() {

accelerometerEvents.listen((AccelerometerEvent event) {

setState(() {

\_accelerometerData =

'X: ${event.x.toStringAsFixed(2)}, Y: ${event.y.toStringAsFixed(2)}, Z: ${event.z.toStringAsFixed(2)}';

});

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Accelerometer App'),

),

body: Center(

child: Text(\_accelerometerData),

),

);

}

}

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

String \_gestureStatus = 'No gesture detected';

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Gesture App'),

),

body: GestureDetector(

onPanUpdate: (details) {

setState(() {

\_gestureStatus =

'Pan Update - Delta: ${details.delta.dx.toStringAsFixed(2)}, ${details.delta.dy.toStringAsFixed(2)}';

});

},

onDoubleTap: () {

setState(() {

\_gestureStatus = 'Double Tap Detected';

});

},

child: Center(

child: Text(\_gestureStatus),

),

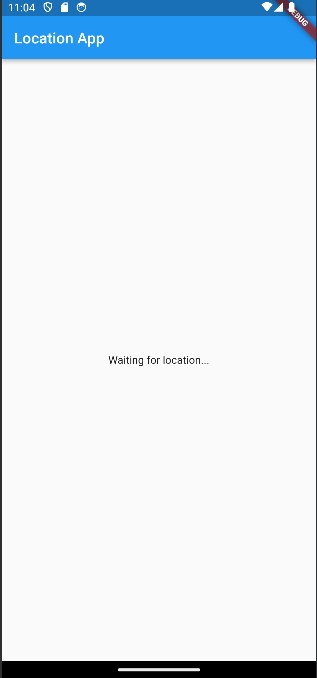
),

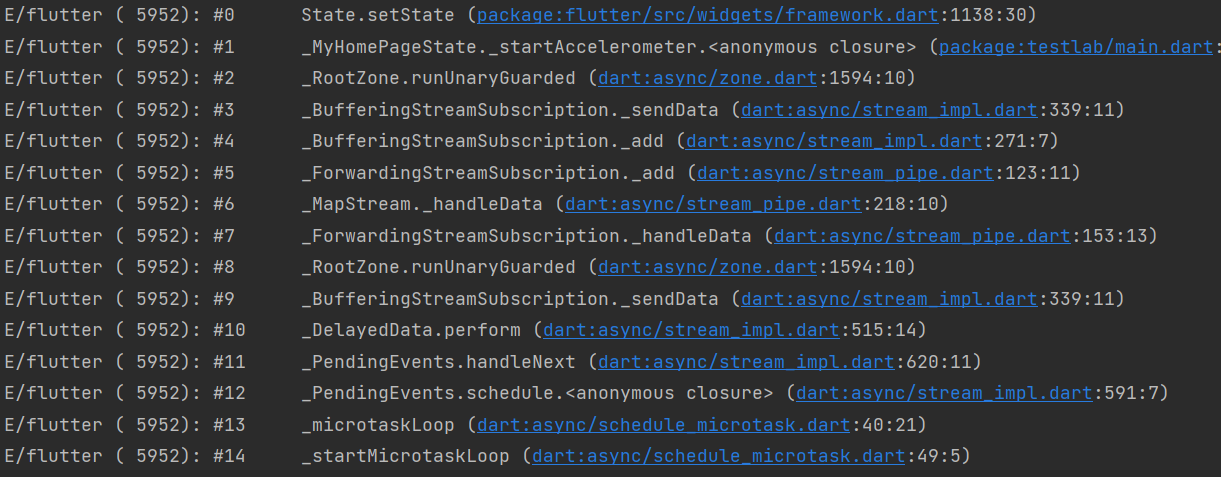
);

}

}

**Output:**

****

****

**Result:** Code for a Flutter mobile application that maximizes underlying phone functionality, utilizing GEO positioning, accelerometer data, and rich gesture-based UI handling was written and executed.

**Ex No. 15**

**Date: 16/11/23**

**Write an application for integrating mobile applications in the market, including social networking software integration with Facebook and Twitter.**

**Aim:** To develop a Flutter mobile application for integrating with mobile app markets and incorporating social networking software integration with Facebook and Twitter.

**Procedure:**

* Design the application's user interface with sections for app market integration and social media connections.
* Integrate Flutter packages for connecting to mobile app markets (e.g., in\_app\_purchase) to handle app purchases or subscriptions.
* Implement logic for displaying available apps, handling downloads, and managing user interactions within the app market.
* Integrate social media SDKs for Facebook and Twitter (e.g., flutter\_facebook\_auth and twitter\_login) to enable user authentication and sharing functionalities.
* Design and implement UI components that allow users to connect their accounts with Facebook and Twitter.
* Implement features for posting updates or sharing content on the user's Facebook and Twitter accounts.
* Ensure secure and user-friendly authentication processes for linking social media accounts.
* Test the application thoroughly to validate the integration with app markets and social media platforms.
* Optimize the application for performance, considering efficient communication with app markets and social media APIs.

**Code:**

import 'package:flutter/material.dart';

import 'package:flutter\_facebook\_auth/flutter\_facebook\_auth.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Facebook Integration App',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: MyHomePage(),

);

}

}

class MyHomePage extends StatefulWidget {

@override

\_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

String \_facebookToken = '';

Future<void> \_loginWithFacebook() async {

try {

final LoginResult result = await FacebookAuth.instance.login();

if (result.status == LoginStatus.success) {

setState(() {

\_facebookToken = result.accessToken!.token;

});

} else {

print('Facebook login failed');

}

} catch (e) {

print('Error during Facebook login: $e');

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Facebook Integration App'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

ElevatedButton(

onPressed: \_loginWithFacebook,

child: Text('Login with Facebook'),

),

SizedBox(height: 20),

Text('Facebook Token: $\_facebookToken'),

],

),

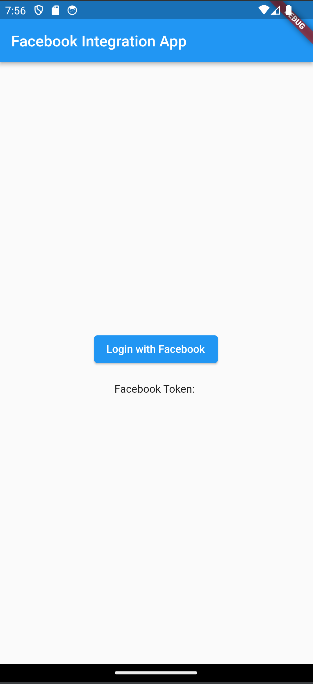
),

);

}

}

**Output:**

****

**Result:** Code for a Flutter mobile application that seamlessly integrates with mobile app markets, providing users access to available apps, and includes social networking software integration with Facebook and Twitter was written and executed.