GUI components, Font and Colors.

Date: 10-08-2023
Ex No: 1

AIM

To develop an application that uses GUI components, Font and Colors.

DEPENDENCIES REQUIRED

No external dependencies required

ALGORITHM

1. Create a new flutter project
2. Add dependencies to pubspec.yaml and Pub get all the dependencies.
3. Code the below logic (AppBar())
   - Buttons , Text and Text Fields are some of the GUI components used in this application.
   - IconButton(), Text() and TextField() are the parent widgets used. Icon(), Color() are the child widgets used.
   - IconButton() needs a required parameter ‘icon’ which takes in Icon() widget. Similarly Text() requires a string ‘text’.
   - TextEditingControllers() are used to capture texts from TextField.
   - Font and Colors are changed used TextStyle() widget in Text()
4. Choose output device and run the application.
5. Observe and interact with the application.

SOURCE CODE (widgets)

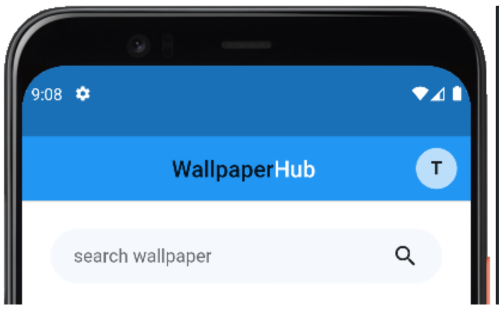
AppBar()

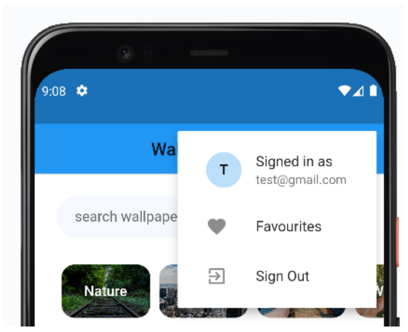
```
AppBar(
 actions: [
   Padding(
     padding: const EdgeInsets.all(8.0),
     child: PopupMenuButton<String>(
       child: CircleAvatar(
         child: Text("AKP"), // Replace with your image path
       ),
       itemBuilder: (BuildContext context) => [
         PopupMenuItem<String>(
           value: 'favorites',
           child: ListTile(
             leading: Icon(Icons.favorite),
             title: Text('Favorites'),
           ),
         ),
         PopupMenuItem<String>(
                value: 'signout',
                child: ListTile(
                  leading: Icon(Icons.exit_to_app),
                  title: Text('Sign Out'),
                ),
              ),
            ],
            onSelected: (value) {
              if (value == 'edit_profile') {
              } else if (value == 'favorites') {
              } else if (value == 'signout') {}
            },
          ),
        ),
      ],
      backgroundColor: Colors.blue,
      title: productName(),
      elevation: 0.0,
    ),
```

TextField()

```
TextField(
 controller: searchController,
 decoration: InputDecoration(
     hintText: "search wallpaper",
     border: InputBorder.none),
),
```

OUTPUT





RESULT

An application that uses GUI components, Font and Colors has been successfully implemented.

NAME: AADHITHYA K PRAVEEN REG NO: 205002001

Layout managers and Event Listeners

Date: 17-08-2023
Ex No: 2

AIM

To develop an application that uses Layout Managers and event listeners.

DEPENDENCIES REQUIRED

No external dependencies required

ALGORITHM

1. Create a new flutter project
2. Add dependencies to pubspec.yaml and Pub get all the dependencies.
3. Code the below logic
    - Row(), Column() and GridView() widgets are used as layout managers.
    - Widgets that need to be placed horizontally are made children to Row() while widgets that need to be placed vertically are made children to Column().
    - GridView() is used to place widgets with specified rows and columns.
    - In built event listeners like onPressed() is used in Search functionality. It fetches data from a public API and is displayed on the screen.
4. Choose output device and run the application.
5. Observe and interact with the application.

SOURCE CODE (widgets)

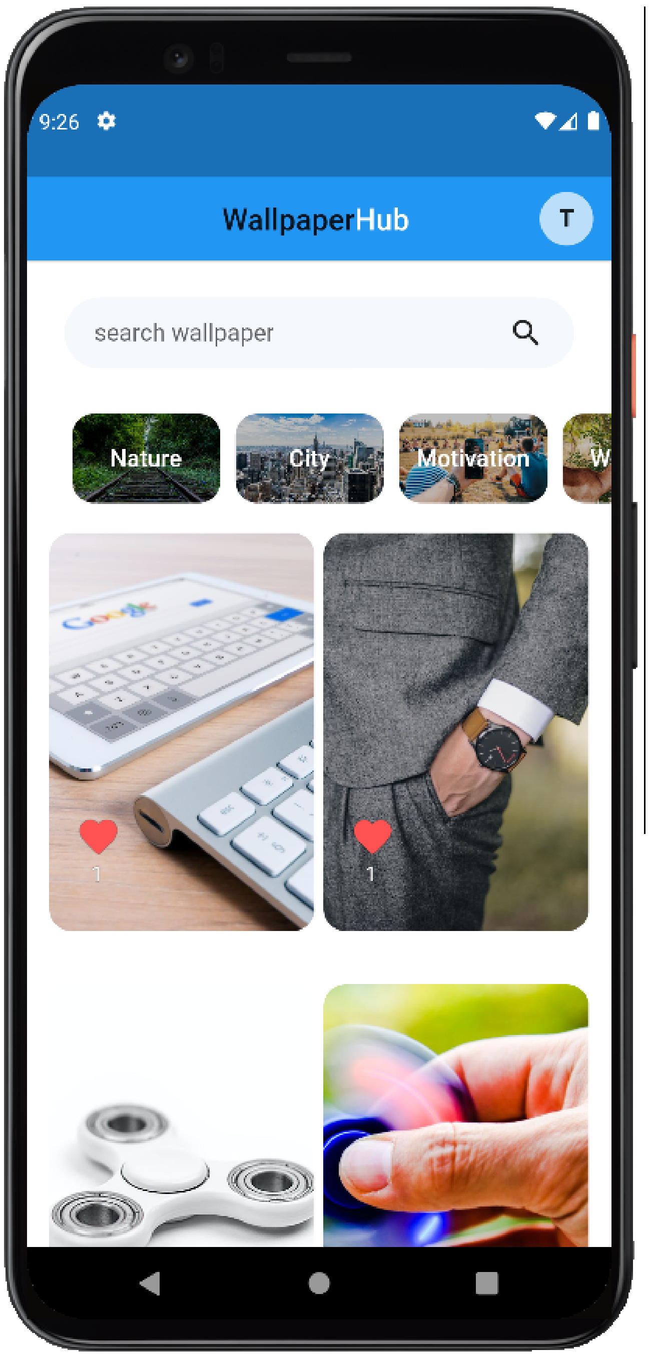
WallpaperList() -> Grid()

```
 class WallpaperList extends StatefulWidget {
  final List<WallpaperModel> wallpapers;

  const WallpaperList({Key? key, required this.wallpapers}) :
super(key: key);

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

class _WallpaperListState extends State<WallpaperList> {
  bool appear = false;

  @override
  void initState() {
    super.initState();
  }
  bool loggedIn = Hive.box('session').get('loggedIn') ?? false;
  var favBox = Hive.box('favourites');
  var favCountBox = Hive.box('favouriteCount');

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(horizontal: 15),
      child: GridView.count(
        shrinkWrap: true,
        physics: ClampingScrollPhysics(),
        crossAxisCount: 2,
        childAspectRatio: 0.6,
        mainAxisSpacing: 6.0,
        crossAxisSpacing: 6.0,
        children: widget.wallpapers.map((wallpaper) {
          int favCount = favCountBox.get(wallpaper.src.portrait) ??
0;
          bool isFav = favBox.get(wallpaper.src.portrait) ?? false;
          return WallpaperGridTile(
            favScreen: false,
            imgUrl: wallpaper.src.portrait,
            loggedIn: loggedIn,
            isFav: isFav,
            favCount: favCount,
            favCountBox: favCountBox,
            favBox: favBox,
          );
        }).toList(),
      ),
    );
  }

  @override
  void dispose() {
    super.dispose();
  }
}
```

WallpaperGridTile()

```
class WallpaperGridTile extends StatefulWidget {
  bool favScreen;
  String imgUrl;
  bool loggedIn;
  bool isFav;
  int favCount;
  Box<dynamic> favCountBox;
  Box<dynamic> favBox;
  WallpaperGridTile({
    super.key,
    required this.favScreen,
    required this.imgUrl,
    required this.loggedIn,
    required this.favCount,
    required this.isFav,
    required this.favBox,
    required this.favCountBox,
  });

  @override
  State<WallpaperGridTile> createState() =>
_WallpaperGridTileState();
}

class _WallpaperGridTileState extends State<WallpaperGridTile> {
  void _showSignUpDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: const Text('Sign Up Required'),
          content: const Text('Please sign in to manage
favourites.'),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.pop(context);
              },
              child: const Text('OK'),
            ),
          ],
        );
      },
    );
  }
```

OUTPUT



RESULT

An application that uses Layout Managers and event listeners has been successfully implemented.

NAME: AADHITHYA K PRAVEEN REG NO: 205002001

Native calculator application

Date: 24-08-2023
Ex No: 3

AIM

Develop a native calculator application.

DEPENDENCIES REQUIRED

math_expressions: ^2.1.1

ALGORITHM

1. Create a new Flutter project.
2. Add dependencies to pubspec.yaml for any required packages.
3. Run flutter pub get to fetch the dependencies.
4. Use a Column or Row widget as the main container for the calculator layout.
5. Include buttons for digits (0-9), arithmetic operators (+, -, *, /), and additional controls (clear, equals).
6. Implement the GUI components using widgets like FlatButton or ElevatedButton for buttons and Text for display.
7. Utilize TextEditingController to capture input from the user via the buttons.
8. Implement functions to handle arithmetic operations based on user input.
9. Display the input and results dynamically using the Text widget.
10. Set up event handlers for button presses, allowing the user to input numbers and perform calculations.
11. Update the display dynamically as the user interacts with the calculator.
12. Ensure error handling for scenarios like division by zero or invalid input.

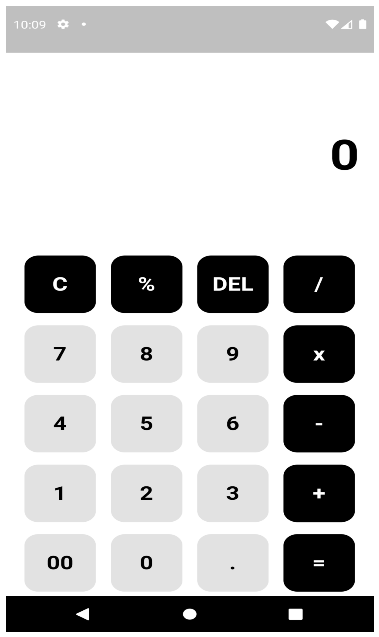
SOURCE CODE (widgets)

CalculatorApp()

```
class CalculatorApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Calculator',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: CalculatorScreen(),
    );
  }
}
```

```
class CalculatorScreen extends StatefulWidget {
  @override
  _CalculatorScreenState createState() => _CalculatorScreenState();
}
class _CalculatorScreenState extends State<CalculatorScreen> {
  String _input = '';
  String _output = '';
  void _handleButtonPress(String buttonText) {
    setState(() {
      if (buttonText == 'C') {
        _input = '';
        _output = '';
      } else if (buttonText == '=') {
        try {
          _output = _evaluateExpression(_input).toString();
        } catch (e) {
          _output = 'Error';
        }
      } else {
        _input += buttonText;
      }
    });
  }
  double _evaluateExpression(String expression) {
    return double.parse(expression);
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Flutter Calculator'),
      ),
      body: Column(
        children: [
          Expanded(
            child: Container(
              alignment: Alignment.bottomRight,
              padding: EdgeInsets.all(16.0),
              child: Text(
                _input,
                style: TextStyle(fontSize: 24.0),
              ),
            ),
          ),
          Expanded(
            child: Container(
              alignment: Alignment.bottomRight,
              padding: EdgeInsets.all(16.0),
              child: Text(
                _output,
                style: TextStyle(fontSize: 48.0, fontWeight:
FontWeight.bold),
              ),
            ),
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _buildButton('7'),
              _buildButton('8'),
              _buildButton('9'),
              _buildButton('/'),
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _buildButton('4'),
              _buildButton('5'),
              _buildButton('6'),
              _buildButton('*'),
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _buildButton('1'),
              _buildButton('2'),
              _buildButton('3'),
              _buildButton('-'),
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _buildButton('0'),
              _buildButton('C'),
              _buildButton('='),
              _buildButton('+'),
            ],
          ),
        ],
      ),
    );
  }
  Widget _buildButton(String buttonText) {
    return Expanded(
      child: Padding(
        padding: EdgeInsets.all(8.0),
        child: FlatButton(
          onPressed: () => _handleButtonPress(buttonText),
          child: Text(
            buttonText,
            style: TextStyle(fontSize: 24.0),
          ),
          color: Colors.grey[300],
        ),
      ),
    );
  }
}
```

OUTPUT



RESULT

A native calculator application has been successfully implemented.

NAME: AADHITHYA K PRAVEEN REG NO: 205002001

Basic graphical primitives

Date: 31-08-2023
Ex No: 4

AIM

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

DEPENDENCIES REQUIRED

No external dependencies required

ALGORITHM

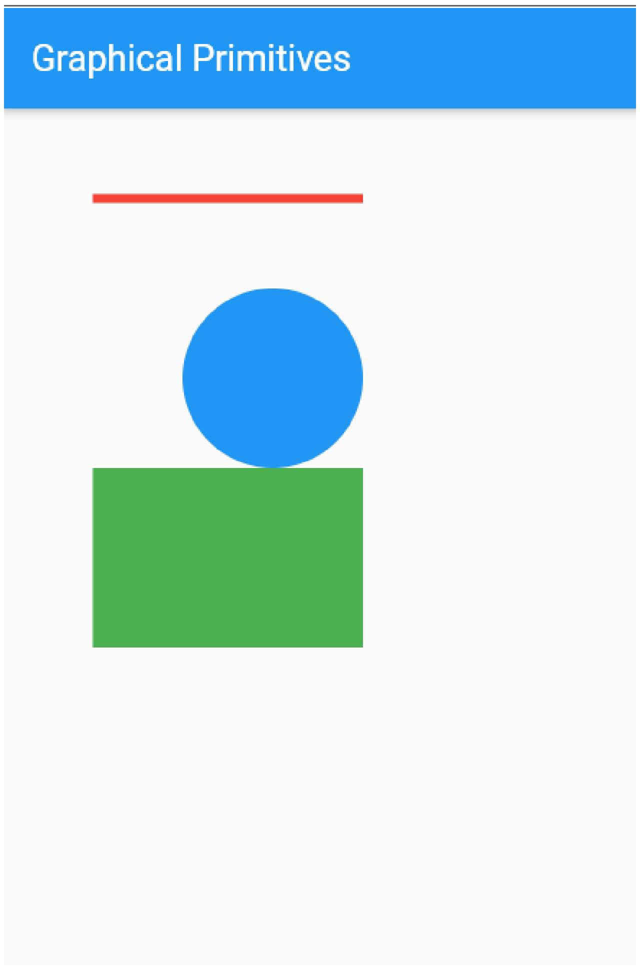
1. Create a new Flutter project.
2. Add any necessary dependencies to pubspec.yaml for drawing graphics if needed.
3. Run flutter pub get to fetch the dependencies.
4. Use a CustomPaint widget as the main container for drawing graphics.
5. Implement a custom painter class that extends CustomPainter.
6. Define the graphical primitives you want to draw, such as lines, circles, rectangles, etc., within the paint method of the custom painter.
7. Override the paint method in the custom painter class to define the drawing logic.
8. Utilize the Canvas class to draw basic shapes and lines using methods like drawLine, drawCircle, drawRect, etc.
9. Customize the appearance of the primitives using attributes like color, stroke width, etc.
10. If user interaction is desired, implement gesture detectors or other event handlers.
11. Allow users to interact with the graphics, such as drawing lines with touch input.
12. Update the graphics dynamically based on user actions.

SOURCE CODE (widgets)

GraphicalPrimitivesApp()

```
class GraphicalPrimitivesApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Graphical Primitives App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: GraphicalPrimitivesScreen(),
    );
  }
}

class GraphicalPrimitivesScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Graphical Primitives'),
      ),
      body: Center(
        child: CustomPaint(
          painter: GraphicalPrimitivesPainter(),
          child: Container(),
        ),
      ),
    );
  }
}

class GraphicalPrimitivesPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    // Draw a line
    Paint linePaint = Paint()
      ..color = Colors.red
      ..strokeWidth = 5.0;
    canvas.drawLine(Offset(50.0, 50.0), Offset(200.0, 50.0),
linePaint);

    // Draw a circle
    Paint circlePaint = Paint()
      ..color = Colors.blue
      ..style = PaintingStyle.fill;
    canvas.drawCircle(Offset(150.0, 150.0), 50.0, circlePaint);

    // Draw a rectangle
    Paint rectPaint = Paint()
      ..color = Colors.green
      ..style = PaintingStyle.fill;
    Rect rect = Rect.fromPoints(Offset(50.0, 200.0), Offset(200.0,
300.0));
    canvas.drawRect(rect, rectPaint);
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) {
    return false;
  }
```

OUTPUT



RESULT

An application that draws basic graphical primitives on the screen has been successfully implemented.

Database application : Firebase/Hive

Date: 07-09-2023
Ex No: 5

AIM

Develop an application that makes use of database.

DEPENDENCIES REQUIRED

firebase_core: ^2.22.0
firebase_auth: ^4.10.1
google_sign_in: ^6.1.5
cloud_firestore: ^4.13.0
hive: ^2.2.3
hive_flutter: ^1.1.0

ALGORITHM

1. Choose a suitable database for your application (e.g., SQLite, Firebase, or any other Flutter-compatible database).
2. Initialize and configure the database connection.
3. Set up tables and schema for storing relevant data.\
4. Define Dart classes that represent the data entities in your application.
5. Use libraries like sqflite or moor to facilitate database interactions.
6. Include methods for CRUD operations (Create, Read, Update, Delete) in the model classes.
7. Design the UI components using Flutter widgets.
8. Create forms or input fields to gather data from the user.
9. Integrate the database model into the application logic to handle data storage and retrieval.
10. Implement logic to interact with the database based on user actions.
11. Use the model methods to insert, update, delete, or retrieve data from the database.
12. Ensure error handling for database operations, such as handling connection errors or data integrity issues.

SOURCE CODE (widgets)

Synchronizing Cloud firestore data with the local DB Hive.

```
import 'package:hive/hive.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

Future<void> syncFavouritesFromFirebase() async {

  var session = Hive.box('session');
  String userEmail = session.get('email');
  if (session.get('loggedIn')) {
    try {

      final DocumentReference userDocRef =

FirebaseFirestore.instance.collection('users').doc(userEmail);

      DocumentSnapshot userDocSnapshot = await userDocRef.get();
      Map<String, dynamic>? userData =
          userDocSnapshot.data() as Map<String, dynamic>?;

      if (userData != null && userData.containsKey('favourites')) {
        Hive.box('favourites').clear();
        Hive.box('favourites').putAll(userData['favourites']);
        fetchWallpaperFavoriteCount();
      }
    } catch (e) {
      print('Error syncing favourites from Firebase: $e');
    }
  }
}

Future<void> fetchWallpaperFavoriteCount() async {
  try {

    CollectionReference wallpapersCollection =
        FirebaseFirestore.instance.collection('wallpapers');

    DocumentSnapshot favouritesDocSnapshot =
        await wallpapersCollection.doc('favourites').get();

    if (favouritesDocSnapshot.exists) {

      Map<String, dynamic>? favouritesData =
          favouritesDocSnapshot.data() as Map<String, dynamic>?;

      if (favouritesData != null) {
        Hive.box('favouriteCount').clear();
        Hive.box('favouriteCount').putAll(favouritesData);
      }
    }
  } catch (e) {
    print('Error fetching wallpaper favorite count: $e');
  }
```

```
}
```

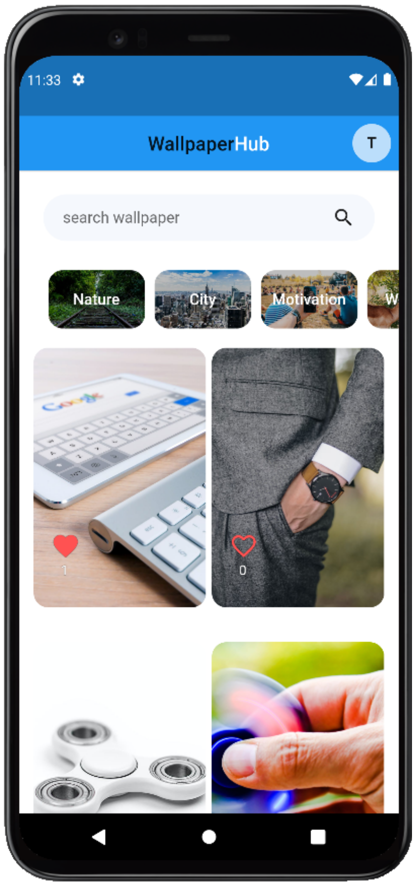
Mark as favorite function that marks a wallpaper as favorite and increases the favorites count in general

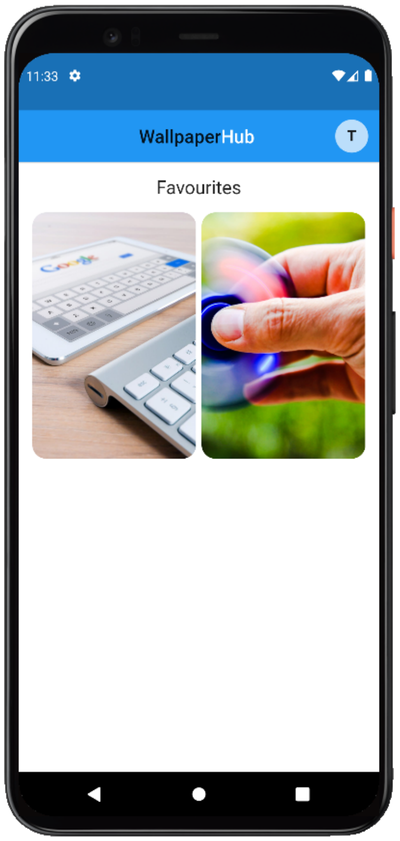
```
import 'package:hive/hive.dart';
import 'package:connectivity/connectivity.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

Future<void> markAsFavorite(String imgUrl, bool fav, int favCount)
async {

  Box<dynamic> favouritesBox = Hive.box('favourites');

  favouritesBox.put(imgUrl, fav);
  Box<dynamic> favouritesCountBox = Hive.box('favouriteCount');
  favouritesCountBox.put(imgUrl, favCount);

  Box<dynamic> userSession = Hive.box('session');
  String userEmail = userSession.get('email');


  var connectivityResult = await Connectivity().checkConnectivity();
  if (connectivityResult == ConnectivityResult.mobile ||
      connectivityResult == ConnectivityResult.wifi) {

    await pushToFavoritesFirebase(userEmail, favouritesBox,
favouritesCountBox);
  }
}

Future<void> pushToFavoritesFirebase(String? userEmail,
    Box<dynamic> favouritesBox, Box<dynamic> favouritesCountBox)
async {
  if (userEmail != null) {
    try {
      final DocumentReference userDocRef =

FirebaseFirestore.instance.collection('users').doc(userEmail);

      Map<dynamic, dynamic> favouritesMap = favouritesBox.toMap();
      await userDocRef.set({
        'favourites': favouritesMap,
      }, SetOptions(merge: true));
      final DocumentReference wallDocRef =
FirebaseFirestore.instance.collection('wallpapers').doc('favourites')
;
      Map<dynamic, dynamic> favouritesCountMap =
          favouritesCountBox.toMap().cast<String, dynamic>();
      await wallDocRef.set(favouritesCountMap, SetOptions(merge:
true));
    } catch (e) {
      print('Error pushing to Firebase: $e');
    }
  }
}
```

OUTPUT

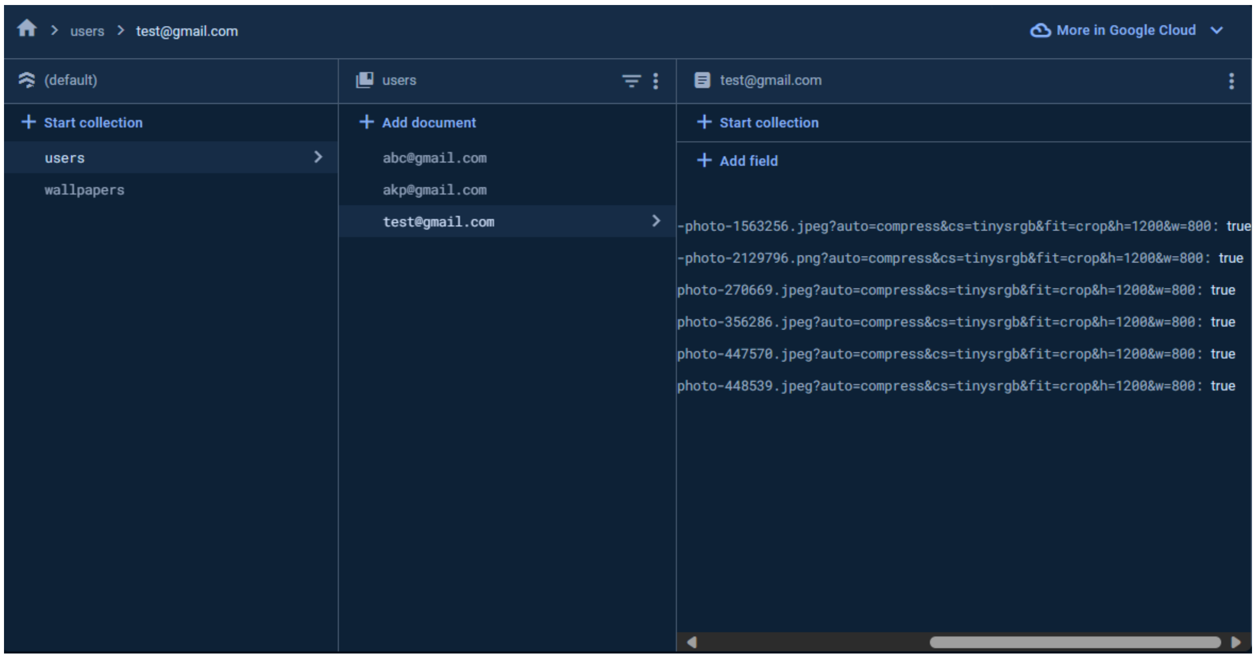
Home screen where Wallpapers can be marked as favorite

Favourites Screen





Firestore Cloud database



```
-photo-1563256.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
-photo-2129796.png?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
photo-270669.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
photo-356286.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
photo-447570.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
photo-448539.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800: true
```

NAME: AADHITHYA K PRAVEEN REG NO: 205002001

RESULT

An application that makes use of a database has been successfully implemented.

Use of RSS feed

Date: 14-09-2023
Ex No: 6

AIM

Develop an application that makes use of RSS Feed.

DEPENDENCIES REQUIRED

http: ^0.13.3
webfeed: ^3.0.2

ALGORITHM

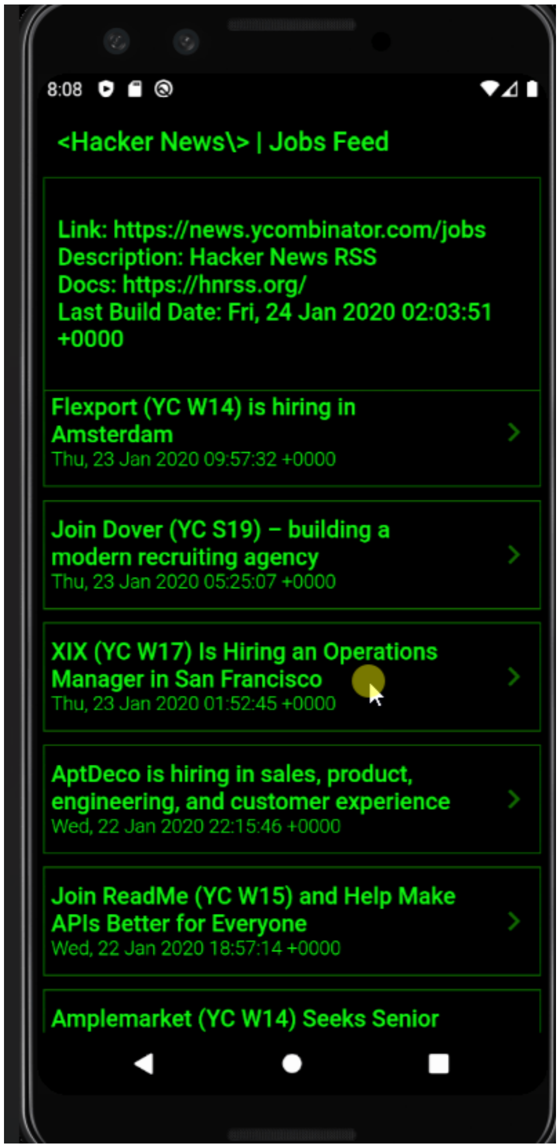
1. Create a new Flutter project.
2. Add dependencies to pubspec.yaml for any required packages related to RSS feed parsing and UI components.
3. Run flutter pub get to fetch the dependencies.
4. Utilize a ListView or GridView to display the list of RSS feed items.
5. Design a custom widget to represent each feed item, including elements like title, description, and publication date.
6. Consider using the FutureBuilder widget to handle asynchronous fetching of the RSS feed data.
7. Choose a package or library for RSS parsing, such as webfeed or any other suitable package.
8. Fetch and parse the RSS feed using the chosen library, extracting relevant information like titles, descriptions, and publication dates.
9. Create a data structure (e.g., a class) to hold the parsed RSS feed items.
10. Implement a mechanism to refresh or update the RSS feed data, allowing users to fetch the latest content.
11. Provide a way to tap on individual feed items to view more details or navigate to the original content.
12. Ensure error handling for scenarios like network connectivity issues or malformed RSS feed responses.

SOURCE CODE (widgets)

AppBar()

```
class RssFeedApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'RSS Feed App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: RssFeedScreen(),
    );
  }
}
class RssFeedScreen extends StatefulWidget {
  @override
  _RssFeedScreenState createState() => _RssFeedScreenState();
}

class _RssFeedScreenState extends State<RssFeedScreen> {
  List<RssItem> _feedItems = [];

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

  Future<void> _fetchRssFeed() async {
    final response = await http.get('YOUR_RSS_FEED_URL');

    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 App'),
      ),
      body: ListView.builder(
        itemCount: _feedItems.length,
        itemBuilder: (context, index) {
          final item = _feedItems[index];
          return ListTile(
            title: Text(item.title),
            subtitle: Text(item.pubDate.toString()),
            onTap: () {
              // Handle item tap if needed
            },
          );
        },
      ),
    );
  }
}
```

OUTPUT



8:08

<Hacker News\> | Jobs Feed

Link: https://news.ycombinator.com/jobs
Description: Hacker News RSS
Docs: https://hnrss.org/
Last Build Date: Fri, 24 Jan 2020 02:03:51 +0000

Flexport (YC W14) is hiring in Amsterdam
Thu, 23 Jan 2020 09:57:32 +0000

Join Dover (YC S19) – building a modern recruiting agency
Thu, 23 Jan 2020 05:25:07 +0000

XIX (YC W17) Is Hiring an Operations Manager in San Francisco
Thu, 23 Jan 2020 01:52:45 +0000

AptDeco is hiring in sales, product, engineering, and customer experience
Wed, 22 Jan 2020 22:15:46 +0000

Join ReadMe (YC W15) and Help Make APIs Better for Everyone
Wed, 22 Jan 2020 18:57:14 +0000

Amplemarket (YC W14) Seeks Senior

RESULT

An application that makes use of RSS Feed has been successfully implemented.

NAME: AADHITHYA K PRAVEEN REG NO: 205002001

Implement multi threading

Date: 21-09-2023
Ex No: 7

AIM

Implement an application that implements multi-threading.

DEPENDENCIES REQUIRED

No external dependencies required

ALGORITHM

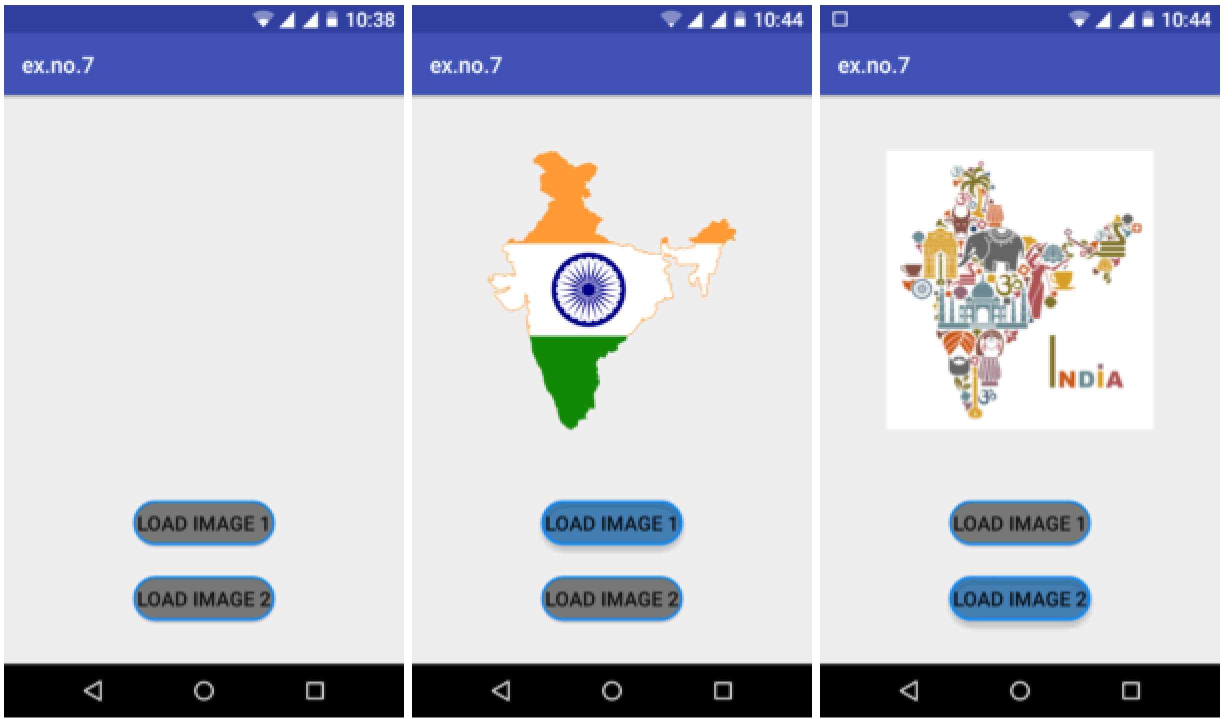
1. Identify specific tasks or operations in your Flutter application that can benefit from multi-threading.
2. Common scenarios include parallelizing heavy computations, network requests, or any operation that may cause UI blocking.
3. Select Appropriate Threading Mechanism:
4. Choose the appropriate threading mechanism based on your needs.
5. Flutter supports Dart's native Isolate for concurrent execution. Consider using isolates for computationally intensive tasks.
6. Identify the portion of your code that can be executed concurrently.
7. Create an isolate using Isolate.spawn and pass the necessary data to the isolate function.
8. In the isolate function, perform the desired computation or task independently of the main thread.
9. Establish communication channels between the main thread and the isolate for data exchange.
10. Use SendPort to send messages from the isolate to the main thread and vice versa.
11. Update the UI with the results obtained from the isolate, ensuring proper synchronization.

SOURCE CODE (widgets)

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: MyHomePage(),
      ),
    );
  }
}

class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Image Loader'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ImageWidget(),
            SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                LoadButton(imagePath: 'assets/india1.png'),
                SizedBox(width: 20),
                LoadButton(imagePath: 'assets/india2.png'),
              ],
            ),
          ],
        ),
      ),
    );
  }
}

class ImageWidget extends StatefulWidget {
  @override
  _ImageWidgetState createState() => _ImageWidgetState();
}

class _ImageWidgetState extends State<ImageWidget> {
  String _imagePath = 'assets/placeholder.png'; // Placeholder
image
```

```
  void setImage(String imagePath) {
    setState(() {
      _imagePath = imagePath;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Image.asset(
      _imagePath,
      width: 250,
      height: 250,
    );
  }
}

class LoadButton extends StatelessWidget {
  final String imagePath;

  const LoadButton({required this.imagePath});

  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        final imageWidget =
context.findAncestorWidgetOfExactType<_ImageWidgetState>();
        if (imageWidget != null) {
          imageWidget.setImage(imagePath);
        }
      },
      child: Text('Load Image'),
    );
  }
}
```

OUTPUT



RESULT

An application that implements multi-threading has been successfully implemented.

GPS based application - Campus Navigation

Date: 28-09-2023
Ex No: 8

AIM

Develop a native application that uses GPS location information.

DEPENDENCIES REQUIRED

google_maps_flutter: ^2.0.6
geolocator: 7.0.3
geocoding: 2.0.0
flutter_polyline_points: ^1.0.0

ALGORITHM

1. Create a new Flutter project.
2. Add necessary permissions for accessing location services in the AndroidManifest.xml (for Android) and Info.plist (for iOS).
3. Include the location package in the pubspec.yaml file.
4. Run flutter pub get to fetch the dependencies.
5. Create a simple interface with a map widget (use google_maps_flutter or similar) to display the user's location.
6. Include relevant information widgets for latitude, longitude, altitude, accuracy, and other GPS-related details.
7. Consider adding a button to trigger location updates or refresh the displayed information.
8. Use the location package to access the device's GPS functionality.
9. Set up a location service and request permissions from the user.
10. Implement listeners to receive updates when the user's location changes.
11. Retrieve and update the UI with the obtained location information.
12. Provide a mechanism for the user to initiate location updates or refresh the displayed information.
13. Use GCP's Google Maps API to pinpoint the location and layout polyline routes across the campus using the given data.
14. Ensure error handling for scenarios where location services are disabled or unavailable.
15. Optionally, implement features like displaying the user's location on a map or providing additional details about the current location.

SOURCE CODE (widgets)

MapView()

```
 final locations =
File('./assets/data/location.csv').readAsLinesSync();
  var location_details = [];
  _get_location_details(String place) {
    for (String line in locations) {
      final values = line.split(',');
      if (place == values[0]) {
        return [values[0], values[1]];
      }
      // location_details.add({
      //   values[0]: [values[1], values[2]],
      // });
    }
    // print(location_details);
    // return location_details;
  }

  // List<String> location_details  = [
  //   'Department of Information Technology',
  //   'Department of Computer Science Engineering',
  //   'Department of Electronics and Communications Engineering',
  //   'Department of Electrical and Electronics  Engineering',
  //   'Department of Chemical  Engineering',
  //   'Department of Biomedical  Engineering',
  //   'Department of Civil Engineering',
  //   'Department of Mechanical Engineering',
  //   'Fountain',
  //   'Admin Block',
  //   'SSN Clinic',
  //   'Mini Auditorium',
  //   'Main Auditorium',
  //   'Rishaabs Foods',
  //   'Snow Cube Foods',
  //   'Main Canteen',
  //   'Vishvaas Needs Store',
  // ];
```
Sample data and variable initialization
```
  late String dropdownvalue = "Select Destination";
  CameraPosition _initialLocation = CameraPosition(
      target: LatLng(12.752725872023166, 80.19659423867354), zoom:
18);
  late GoogleMapController mapController;

  late Position _currentPosition;
  String _currentAddress = '';

  final startAddressController = TextEditingController();
  final destinationAddressController = TextEditingController();
  final startAddressFocusNode = FocusNode();
  final desrinationAddressFocusNode = FocusNode();

  String _startAddress = '';
  String _destinationAddress = '';
  String? _placeDistance;

  Set<Marker> markers = {};
  // markers.add(Marker)
  late PolylinePoints polylinePoints;
  Map<PolylineId, Polyline> polylines = {};
  List<LatLng> polylineCoordinates = [];

  final _scaffoldKey = GlobalKey<ScaffoldState>();

  Widget _textField({
    required TextEditingController controller,
    required FocusNode focusNode,
    required String label,
    required String hint,
    required double width,
    required Icon prefixIcon,
    Widget? suffixIcon,
    required Function(String) locationCallback,
  }) {
    return Container(
      width: width * 0.8,
      child: TextField(
        onChanged: (value) {
          locationCallback(value);
        },
        controller: controller,
        focusNode: focusNode,
        decoration: new InputDecoration(
          prefixIcon: prefixIcon,
          suffixIcon: suffixIcon,
          labelText: label,
          filled: true,
          fillColor: Colors.white,
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.all(
              Radius.circular(10.0),
            ),
            borderSide: BorderSide(
              color: Colors.grey.shade400,
              width: 2,
            ),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.all(
              Radius.circular(10.0),
            ),
            borderSide: BorderSide(
              color: Colors.blue.shade300,
              width: 2,
            ),
          ),
          contentPadding: EdgeInsets.all(15),
          hintText: hint,
        ),
      ),
    );
  }

  // This function is triggered when the floating button is pressed
  // final input = new File('assets/data/marker_loc.csv').openRead();
  // late final fields = input
  //     .transform(utf8.decoder)
  //     .transform(new CsvToListConverter())
  //     .toList();
  // Future<List<List>> _data = fields;

  // _loadAsset() async {
  //   final myData = await
rootBundle.loadString("assets/Book1.csv");
  //   List<List<dynamic>> csvTable =
CsvToListConverter().convert(myData);

  //   data = csvTable;};
```

Method for getting current location using GeoLocator (GPS)

```
  // Method for retrieving the current location
  _getCurrentLocation() async {
    await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high)
        .then((Position position) async {
      setState(() {
        _currentPosition = position;
        print('CURRENT POS: $_currentPosition');
        mapController.animateCamera(
          CameraUpdate.newCameraPosition(
            CameraPosition(
              target: LatLng(position.latitude, position.longitude),
              zoom: 18.0,
            ),
          ),
        );
      });
      await _getAddress();
    }).catchError((e) {
      print(e);
    });
  }
```

Method for retrieving the address

```
  _getAddress() async {
    try {
      List<Placemark> p = await placemarkFromCoordinates(
          _currentPosition.latitude, _currentPosition.longitude);

      Placemark place = p[0];

      setState(() {
        _currentAddress =
            "${place.name}, ${place.locality}, ${place.postalCode},
${place.country}";
        startAddressController.text = _currentAddress;
        _startAddress = _currentAddress;
      });
    } catch (e) {
      print(e);
    }
  }
```

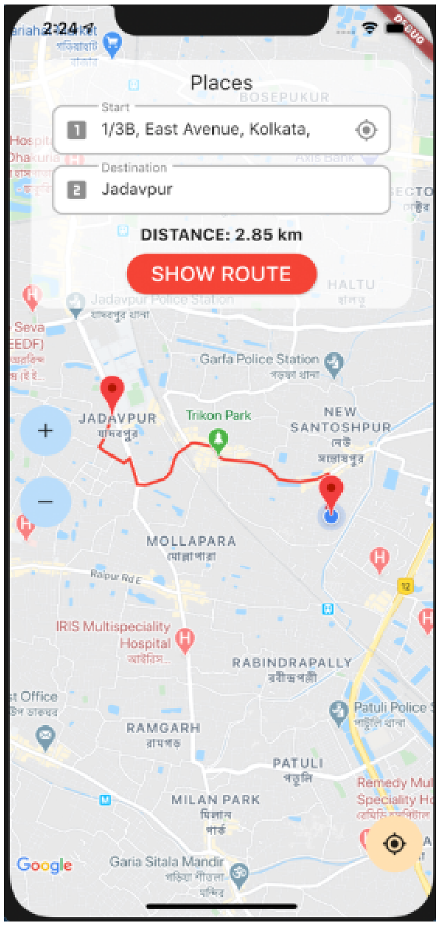
Method for calculating the distance between two places

```
  Future<bool> _calculateDistance() async {
    try {
      // Retrieving placemarks from addresses
      List<Location> startPlacemark = await
locationFromAddress(_startAddress);
      List<Location> destinationPlacemark =
          await locationFromAddress(_destinationAddress);

      // Use the retrieved coordinates of the current position,
      // instead of the address if the start position is user's
      // current position, as it results in better accuracy.
      double startLatitude = _startAddress == _currentAddress
          ? _currentPosition.latitude
          : startPlacemark[0].latitude;

      double startLongitude = _startAddress == _currentAddress
          ? _currentPosition.longitude
          : startPlacemark[0].longitude;

      double destinationLatitude = destinationPlacemark[0].latitude;
      double destinationLongitude =
destinationPlacemark[0].longitude;

      String startCoordinatesString = '($startLatitude,
$startLongitude)';
      String destinationCoordinatesString =
          '($destinationLatitude, $destinationLongitude)';

      // Start Location Marker
      late String dropdownvalue = 'Clock Tower';
      Marker startMarker = Marker(
        markerId: MarkerId(startCoordinatesString),
        position: LatLng(startLatitude, startLongitude),
        infoWindow: InfoWindow(
          title: 'Start $startCoordinatesString',
          snippet: _startAddress,
        ),
        icon: BitmapDescriptor.defaultMarker,
      );



      print(
        'START COORDINATES: ($startLatitude, $startLongitude)',
      );
      print(
        'DESTINATION COORDINATES: ($destinationLatitude,
$destinationLongitude)',
      );



      Map<MarkerId, Marker> markers = <MarkerId, Marker>{};
      // Calculating to check that the position relative
      // to the frame, and pan & zoom the camera accordingly.
      double miny = (startLatitude <= destinationLatitude)
          ? startLatitude
          : destinationLatitude;
      double minx = (startLongitude <= destinationLongitude)
          ? startLongitude
          : destinationLongitude;
      double maxy = (startLatitude <= destinationLatitude)
          ? destinationLatitude
          : startLatitude;
      double maxx = (startLongitude <= destinationLongitude)
          ? destinationLongitude
          : startLongitude;

      double southWestLatitude = miny;
      double southWestLongitude = minx;

      double northEastLatitude = maxy;
      double northEastLongitude = maxx;

      // Accommodate the two locations within the
      // camera view of the map
      mapController.animateCamera(
        CameraUpdate.newLatLngBounds(
          LatLngBounds(
            northeast: LatLng(northEastLatitude, northEastLongitude),
            southwest: LatLng(southWestLatitude, southWestLongitude),
          ),
          100.0,
        ),
      );



      await _createPolylines(startLatitude, startLongitude,
destinationLatitude,
          destinationLongitude);

      double totalDistance = 0.0;

      // Calculating the total distance by adding the distance
      // between small segments
      for (int i = 0; i < polylineCoordinates.length - 1; i++) {
        totalDistance += _coordinateDistance(
          polylineCoordinates[i].latitude,
          polylineCoordinates[i].longitude,
          polylineCoordinates[i + 1].latitude,
          polylineCoordinates[i + 1].longitude,
        );
      }

      setState(() {
        _placeDistance = totalDistance.toStringAsFixed(2);
        print('DISTANCE: $_placeDistance km');
      });

      return true;
    } catch (e) {
      print(e);
    }
    return false;
  }
```

Formula for calculating distance between two coordinates

```
  // https://stackoverflow.com/a/54138876/11910277
  double _coordinateDistance(lat1, lon1, lat2, lon2) {
    var p = 0.017453292519943295;
    var c = cos;
    var a = 0.5 -
        c((lat2 - lat1) * p) / 2 +
        c(lat1 * p) * c(lat2 * p) * (1 - c((lon2 - lon1) * p)) / 2;
    return 12742 * asin(sqrt(a));
  }

  // Create the polylines for showing the route between two places
  _createPolylines(
    double startLatitude,
    double startLongitude,
    double destinationLatitude,
    double destinationLongitude,
  ) async {
    polylinePoints = PolylinePoints();
    PolylineResult result = await
polylinePoints.getRouteBetweenCoordinates(
      Secrets.API_KEY, // Google Maps API Key
      PointLatLng(startLatitude, startLongitude),
      PointLatLng(destinationLatitude, destinationLongitude),
      travelMode: TravelMode.transit,
    );
    if (result.points.isNotEmpty) {
      result.points.forEach((PointLatLng point) {
        polylineCoordinates.add(LatLng(point.latitude,
point.longitude));
      });
    }

    PolylineId id = PolylineId('poly');
    Polyline polyline = Polyline(
      polylineId: id,
      color: Colors.red,
      points: polylineCoordinates,
      width: 3,
    );
    polylines[id] = polyline;
  }
```

OUTPUT



RESULT

A native application that uses GPS location information has been successfully implemented.

Data into SD card

Date: 05-10-2023
Ex No: 9

AIM

Implement an application that writes data to the SD card

DEPENDENCIES REQUIRED

cached_network_image: ^3.2.3
image_gallery_saver: ^2.0.3
permission_handler: ^11.0.0
path_provider: ^2.1.1

ALGORITHM

1. Declare the necessary permissions in the AndroidManifest.xml file for writing to external storage.
2. In the Flutter code, use the permission_handler package to request and check for permissions dynamically.
3. Create a user interface that allows users to input data or select data to be written to the SD card.
4. Use appropriate widgets for data input (e.g., TextField, DropdownButton, etc.) and a button to trigger the data-writing process.
5. Upon button press, gather the data from the input fields or selected sources.
6. Use the path_provider package to get the path to the external storage directory (SD card).
7. Open a file in write mode and write the data to the specified file on the SD card. Consider using File and dart:io for file operations.
8. Implement error handling to address scenarios like permission denial or issues with writing to the SD card.
9. Provide feedback to the user on the success or failure of the data-writing operation using Flutter widgets like SnackBar or AlertDialog

SOURCE CODE (widgets)

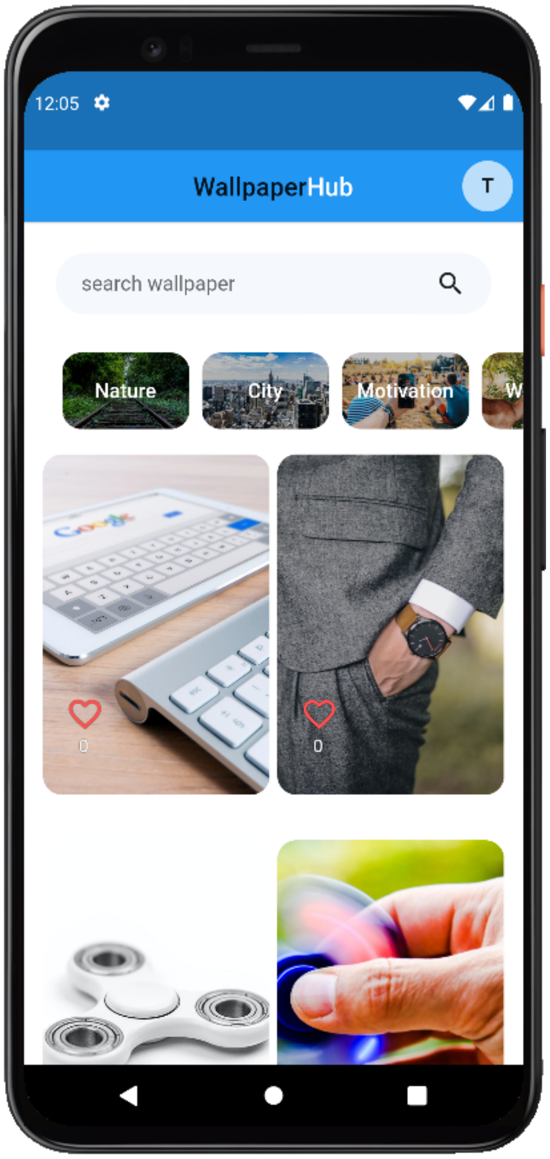
Hive initialization for Local memory storage

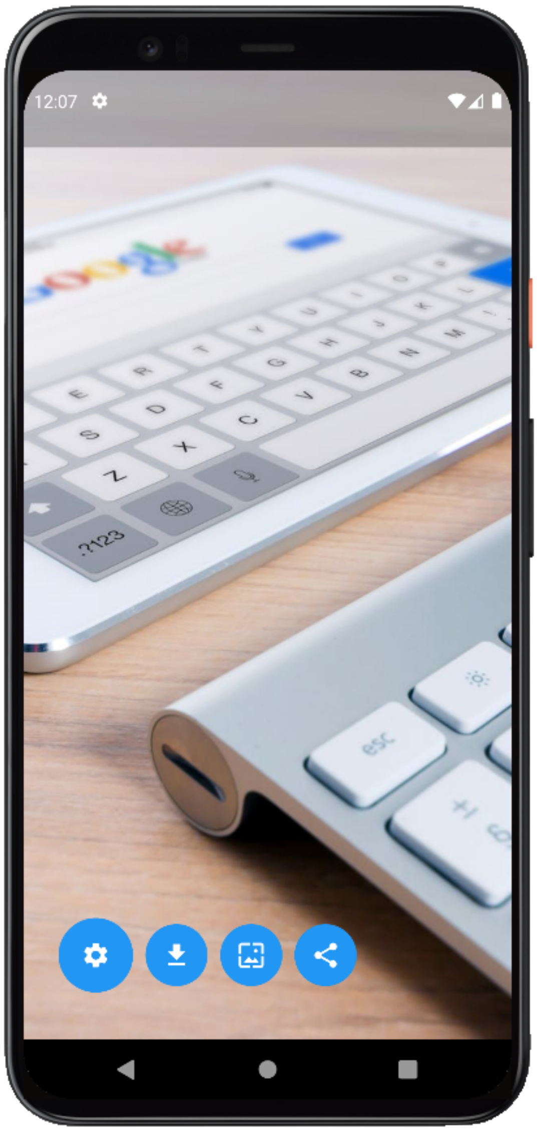
```
 final Directory extDir = await getExternalStorageDirectory();
  final String hivePath =
"${extDir.path}/your_custom_hive_directory";
  await Hive.initFlutter(hivePath);
```

Async functions to store in SD card

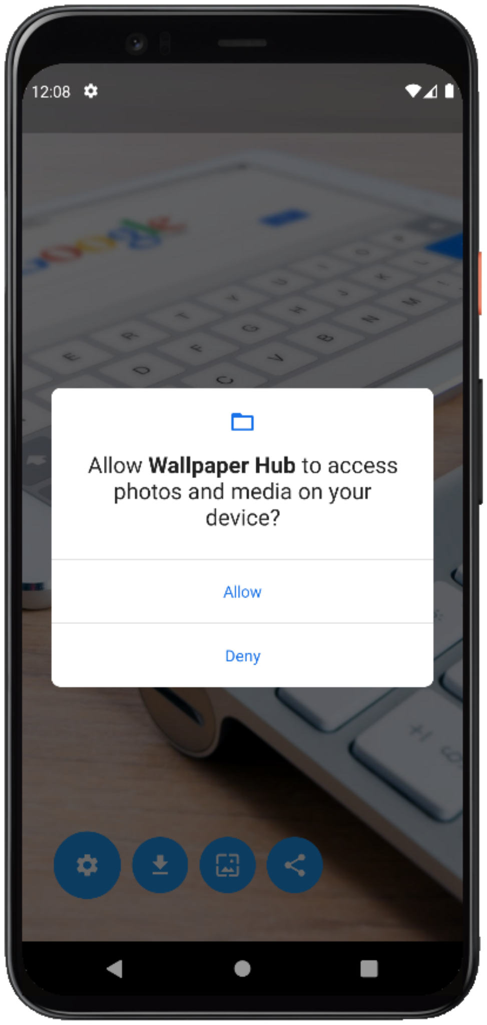
```
  _save() async {
    if (Platform.isAndroid) {
      await _askpermission();
    }
    var response = await Dio()
        .get(widget.imgUrl, options: Options(responseType:
ResponseType.bytes));
    await
ImageGallerySaver.saveImage(Uint8List.fromList(response.data));
  }

  _askpermission() async {
    if (Platform.isIOS) {
      await Permission.photos.request();
    } else {
      await Permission.storage.request();
    }
  }
  _askpermission() async {
    if (Platform.isIOS) {
      Map<PermissionGroup, PermissionStatus> permissions =
          await PermissionHandler()
              .requestPermissions([PermissionGroup.photos]);
    } else {
      PermissionStatus permission = await PermissionHandler()
          .checkPermissionStatus(PermissionGroup.storage);
    }
  }
```

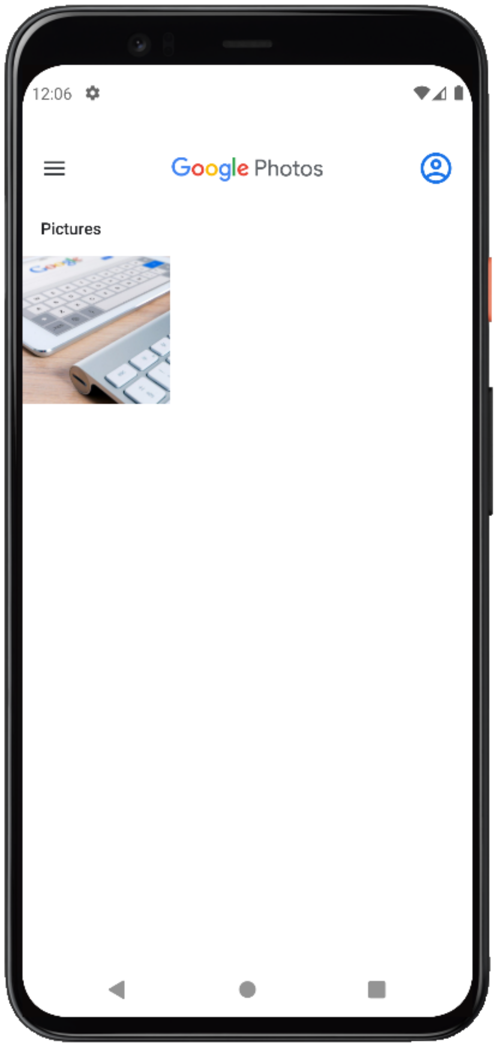
OUTPUT





NAME: AADHITHYA K PRAVEEN REG NO: 205002001


12:08
Allow **Wallpaper Hub** to access photos and media on your device?
Allow
Deny



RESULT

An application that writes data to the SD card has been successfully implemented.

Alert upon receiving a message

Date: 12-10-2023
Ex No: 10

AIM

Implement an application that creates an alert upon receiving a message

DEPENDENCIES REQUIRED

flutter_local_notifications: ^16.1.0

ALGORITHM

1. Create a new Flutter project.
2. Set up dependencies in the pubspec.yaml file as needed.
3. Run flutter pub get to fetch the dependencies.
4. Create a simple UI with a messaging interface.
5. Utilize a ListView or similar widget to display messages.
6. Include a function or button to simulate receiving a new message.
7. Integrate a package or use the built-in AlertDialog class in Flutter.
8. Set up a function to create and display an alert when a new message is received.
9. Customize the alert content with relevant information from the received message.
10. Trigger the alert function when a new message is simulated or received.
11. This can be done through a button press or automatically when a new message is detected.

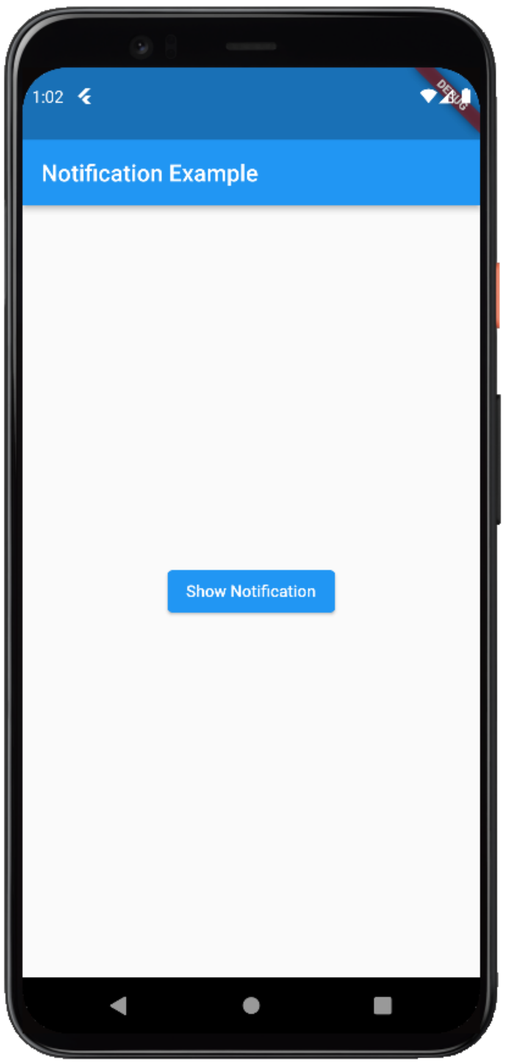
SOURCE CODE (widgets)
AppBar()

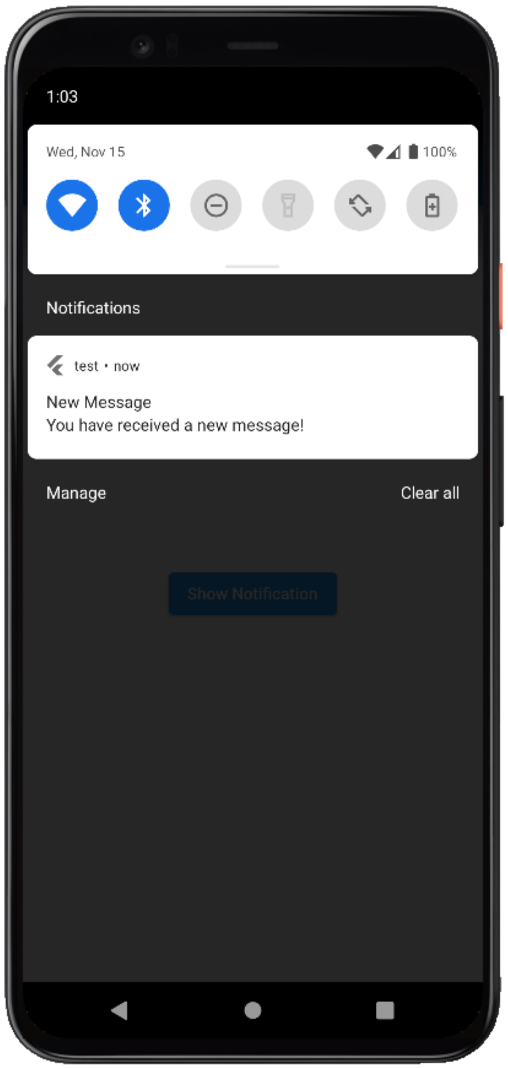
```
AppBar(
 actions: [
   Padding(
     padding: const EdgeInsets.all(8.0),
     child: PopupMenuButton<String>(
       child: CircleAvatar(
         child: Text("AKP"), // Replace with your image path
       ),
       itemBuilder: (BuildContext context) => [
         PopupMenuItem<String>(
           value: 'favorites',
           child: ListTile(
             leading: Icon(Icons.favorite),
             title: Text('Favorites'),
           ),
         ),
         PopupMenuItem<String>(
           value: 'signout',
           child: ListTile(
             leading: Icon(Icons.exit_to_app),
             title: Text('Sign Out'),
           ),
         ),
       ],
       onSelected: (value) {
         if (value == 'edit_profile') {
         } else if (value == 'favorites') {
         } else if (value == 'signout') {}
       },
     ),
   ),
 ],
 backgroundColor: Colors.blue,
 title: productName(),
 elevation: 0.0,
),
```

TextField()

```
TextField(
 controller: searchController,
 decoration: InputDecoration(
     hintText: "search wallpaper",
     border: InputBorder.none),
),
```

OUTPUT





New Message
You have received a new message!

RESULT

An application to trigger notifications upon simulated messages(webhooks) is developed successfully using flutter framework.

Alarm clock

Date: 19-10-2023
Ex No: 11

AIM

Write a mobile application that creates an alarm clock.

DEPENDENCIES REQUIRED

flutter_local_notifications: ^10.0.0
timezone: ^0.9.3
flutter_native_timezone: ^2.0.1

ALGORITHM

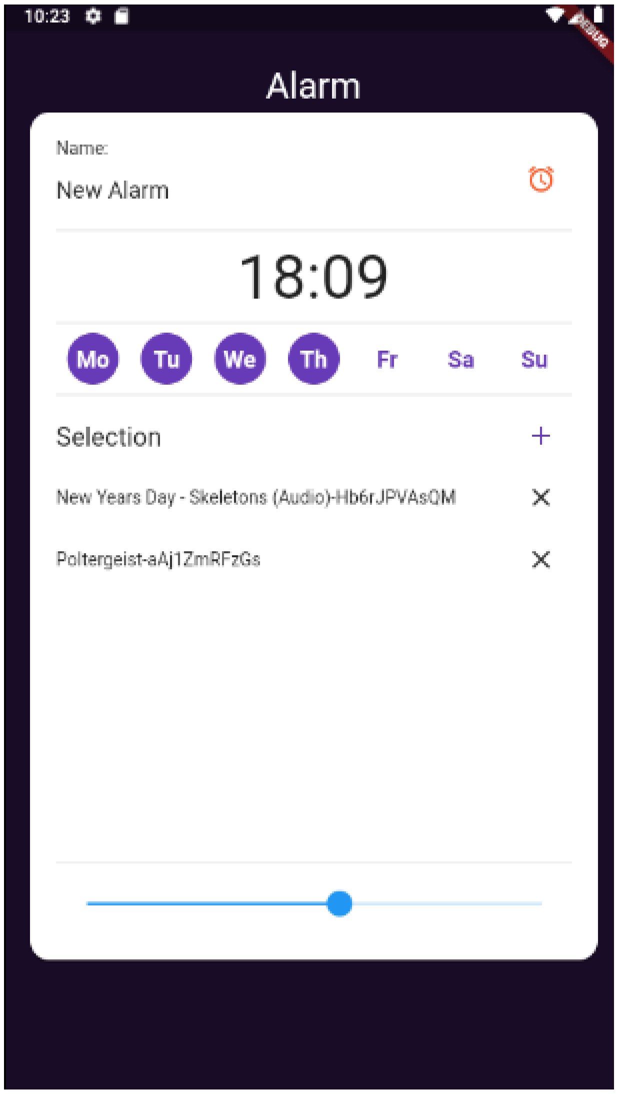
1. Create a new Flutter project.
2. Add dependencies to pubspec.yaml for any required packages related to date and time handling, and alarm management.
3. Run flutter pub get to fetch the dependencies.
4. Use a Column or ListView widget as the main container for the alarm clock layout.
5. Include components like TimePicker for setting the alarm time, a switch to enable/disable the alarm, and buttons for setting and deleting alarms.
6. Utilize the ListView.builder to display a list of alarms dynamically.
7. Use a data model to represent an alarm, including properties like time, status (enabled/disabled), and a unique identifier.
8. Implement functions to add, edit, and delete alarms.
9. Utilize the device's local notification system or a third-party package to trigger alarms at the specified times.
10. Handle the alarm-triggered events by showing a notification or playing a sound.
11. Set up event handlers for user actions, such as selecting a time, toggling the alarm status, and interacting with the list of alarms.
12. Update the UI dynamically to reflect changes in alarm settings and statuses.
13. Ensure proper error handling, such as preventing the creation of alarms with past times or handling conflicts with existing alarms.

SOURCE CODE (widgets)

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Alarm Clock App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

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

class _MyHomePageState extends State<MyHomePage> {
  FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =
      FlutterLocalNotificationsPlugin();
  TimeOfDay selectedTime = TimeOfDay.now();
  bool isAlarmSet = false;
  late StreamSubscription<TimeOfDay?> _timePickerSubscription;

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

    var initializationSettingsAndroid =
        AndroidInitializationSettings('@mipmap/ic_launcher');
    var initializationSettingsIOS = IOSInitializationSettings();
    var initializationSettings = InitializationSettings(
      android: initializationSettingsAndroid,
      iOS: initializationSettingsIOS,
    );


flutterLocalNotificationsPlugin.initialize(initializationSettings);

    _timePickerSubscription =
        _selectTime(context).listen((selectedTime) async {
      if (selectedTime != null) {
        await _scheduleNotification(selectedTime);
      }
    });
  }

  @override
  void dispose() {
    _timePickerSubscription.cancel();
    super.dispose();
  }
```

```
Future<void> _scheduleNotification(TimeOfDay selectedTime) async {
  var now = DateTime.now();
  var scheduledTime = DateTime(
    now.year,
    now.month,
    now.day,
    selectedTime.hour,
    selectedTime.minute,
  );

  if (scheduledTime.isBefore(now)) {
    scheduledTime = scheduledTime.add(Duration(days: 1));
  }

  var androidPlatformChannelSpecifics = AndroidNotificationDetails(
    'alarm_channel_id',
    'Alarm Notification',
    'Notification for Alarm',
    importance: Importance.high,
    priority: Priority.high,
  );

  var iOSPlatformChannelSpecifics = IOSNotificationDetails();
  var platformChannelSpecifics = NotificationDetails(
    android: androidPlatformChannelSpecifics,
    iOS: iOSPlatformChannelSpecifics,
  );

  await flutterLocalNotificationsPlugin.zonedSchedule(
    0,
    'Alarm',
    'Wake up! It\'s time!',
    TZDateTime.from(scheduledTime, tz.TZDateTime.local),
    platformChannelSpecifics,
    androidAllowWhileIdle: true,
    uiLocalNotificationDateInterpretation:
        UILocalNotificationDateInterpretation.absoluteTime,
  );

  setState(() {
    isAlarmSet = true;
  });
}

Stream<TimeOfDay?> _selectTime(BuildContext context) async* {
    yield await showTimePicker(
      context: context,
      initialTime: selectedTime,
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Alarm Clock App'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            isAlarmSet
                ? Text(
                    'Alarm is set for
${selectedTime.format(context)}',
                    style: TextStyle(fontSize: 18),
                  )
                : Text(
                    'No alarm set',
                    style: TextStyle(fontSize: 18),
                  ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () async {
                await _selectTime(context);
              },
              child: Text('Set Alarm'),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () async {
                await flutterLocalNotificationsPlugin.cancel(0);
                setState(() {
                  isAlarmSet = false;
                });
              },
              child: Text('Cancel Alarm'),
            ),
          ],
        ),
        ),
      );
    }
}
```

OUTPUT



New Years Day - Skeletons (Audio)-Hb6rJPVAsQM

Poltergeist-aAj1ZmRFzGs

RESULT

Flutter based alarm clock mobile application is successfully developed.

Gaming application with Multimedia support - Flappy bird

Date: 26-10-2023
Ex No: 12

AIM

Develop a simple gaming application with multimedia support.

DEPENDENCIES REQUIRED

flame: ^1.10.1
url_launcher: ^6.2.1

ALGORITHM

1. Create a new Flutter project.
2. Integrate necessary packages for multimedia support, such as audioplayers for audio and flame for game development.
3. Run flutter pub get to fetch the dependencies.
4. Use a Game widget from the flame package as the main container for the game.
5. Design game elements using widgets such as Sprite or Positioned for graphics.
6. Implement a game loop for continuous rendering and updating of game elements.
7. Integrate multimedia elements like background music and sound effects using the audioplayers package.
8. Implement functions to play, pause, and stop audio based on game events.
9. Use images and animations for game characters or objects, loading them into the game with the appropriate widgets.
10. Set up event handlers for user interactions, such as taps or swipes, to control game elements.
11. Implement game logic for scoring, levels, or challenges based on user actions.
12. Update the game state and UI in response to user input and game events.

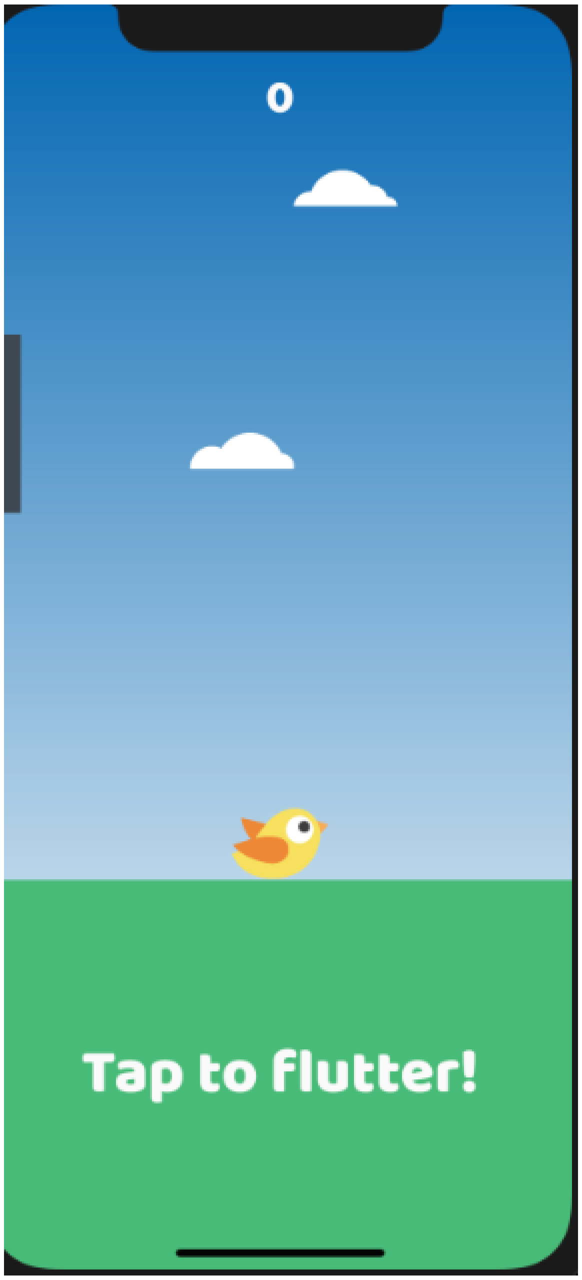
SOURCE CODE (widgets)

```
import 'package:flame/flame.dart';
import 'package:flame/util.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutters/flutters-game.dart';

void main() async {
  Util flameUtil = Util();
  WidgetsFlutterBinding.ensureInitialized();
  if (!kIsWeb) {
    await Flame.util.setPortrait();
    await Flame.util.fullScreen();
  }

  Flame.images.loadAll(<String>[
    'bird-0.png',
    'bird-1.png',
    'bird-0-left.png',
    'bird-1-left.png',
    'cloud-1.png',
    'cloud-2.png',
    'cloud-3.png',
  ]);

  final screenDimensions = await Flame.util.initialDimensions();
  FluttersGame game = FluttersGame(screenDimensions);
  TapGestureRecognizer tapSink = TapGestureRecognizer();
  tapSink.onTapDown = game.onTapDown;
  tapSink.onTapUp = game.onTapUp;
  RawKeyboard.instance.addListener((RawKeyEvent rawKeyEvent) {
    final space = ' ';
    if (rawKeyEvent.character == space) {
      game.onTapDown(TapDownDetails());
    }
  });

  runApp(game.widget);
  flameUtil.addGestureRecognizer(tapSink);
}
```

Background

```
import 'dart:ui';

import 'package:flutter/rendering.dart';
import 'package:flutters/components/cloud.dart';
import 'package:flutters/components/core/gameobject.dart';
import 'package:flutters/flutters-game.dart';

class Background extends GameObject {
  final Gradient gradient = new LinearGradient(
    begin: Alignment.topCenter,
    colors: <Color>[
      Color(0xff0165b1),
      Color(0xffFFFFFF),
    ],
    stops: [
      0.0,
      1.0,
    ],
    end: Alignment(0, 0.9),
  );
  Rect rect;
  Paint paint;
  Background(FluttersGame game, double x, double y, double width,
double height)
      : super(game) {
    paint = Paint();
    paint.color = Color(0xff77b5e1);

    rect = Rect.fromLTWH(x, y, width, height);
    paint = new Paint()..shader = gradient.createShader(rect);
    this.addChild(new Cloud(this.game, 0, game.tileSize * 1.7));
    this.addChild(new Cloud(this.game, 0, game.tileSize * 4.4));
  }

  @override
  void render(Canvas c) {
    c.drawRect(rect, paint);
    super.render(c);
  }

  @override
  void update(double t) {
    super.update(t);
  }
}
```

OUTPUT



RESULT

A gaming application using flutter and flame framework is successfully implemented with multimedia support.

REST API based Mobile Application

Date: 02-11-2023
Ex No: 13

AIM

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

DEPENDENCIES REQUIRED

dio: ^5.3.2
http: ^1.1.0
connectivity: ^3.0.6

ALGORITHM

1. Create a new mobile application project using a suitable framework like Flutter or React Native.
2. Set up the project structure, including necessary folders for screens, services, and models.
3. Dependency Setup:
4. Add dependencies to the pubspec.yaml or package.json file for packages that facilitate REST communication.
5. For REST, use packages like http or dio for making HTTP requests.
6. Design the application's user interface with screens and components using appropriate widgets.
7. Include input fields, buttons, and other UI elements to facilitate user interaction.
8. Implement loading indicators or error messages to provide feedback during data retrieval.
9. Create service classes to encapsulate REST API calls. Use classes like HttpClient or custom service classes.
10. For SOAP, define methods to construct SOAP envelopes and send requests to the backend service.
11. For REST, create methods to perform GET, POST, PUT, and DELETE requests, handling request parameters and headers.
12. Implement error handling to manage network errors, timeouts, and unexpected responses.
13. Parse the responses received from the backend services and convert them into usable data structures (e.g., Dart objects).
14. Configure the application to connect to the backend services hosted in a cloud environment.
15. Provide the necessary authentication mechanisms, such as API keys, OAuth tokens, or other security measures.
16. Ensure that the application can communicate securely with the cloud services, considering HTTPS and other encryption protocols.
17. Implement user interactions that trigger data requests to the backend services.
18. Display loading indicators during data retrieval to keep users informed about ongoing processes.
19. Update the UI with the retrieved data or display error messages in case of failures.

SOURCE CODE (widgets)

Querying more wallpapers

```
  getSearchWallpaper({String query = 'trending'}) async {
    String currPage = currentPage.toString();
    var response = await http.get(
        Uri.parse(

"https://api.pexels.com/v1/search?query=$query&per_page=20&page=$curr
Page"),
        headers: {"Authorization": apiKEY});

    Map<String, dynamic> jsonData = jsonDecode(response.body);
    // print(jsonData);
    jsonData["photos"].forEach((element) {
      WallpaperModel wallpaperModel;
      wallpaperModel = WallpaperModel.fromMap(element);
      wallpapers.add(wallpaperModel);
    });
    setState(() {
      isLoading = false;
    });
 }
```

Load more wallpapers (Virtual Paging)

```
 void loadMoreWallpapers() async {
    setState(() {
      isLoading = true;
      currentPage++;
    });
    await Future.delayed(Duration(seconds: 1));
    try {
      // Call the method to get the next set of wallpapers based on
the category
      // You may need to adjust the parameters of getSearchWallpaper
method
      List<WallpaperModel> nextWallpapers = await getSearchWallpaper(
          query:
              searchController.text != '' ? searchController.text :
'trending');
```

```
      // Append the new wallpapers to the existing list
      setState(() {
        wallpapers.addAll(nextWallpapers);
      });
    } catch (e) {
      print('Error loading wallpapers: $e');
    } finally {
      setState(() {
        isLoading = false;
      });
    }
  }
```

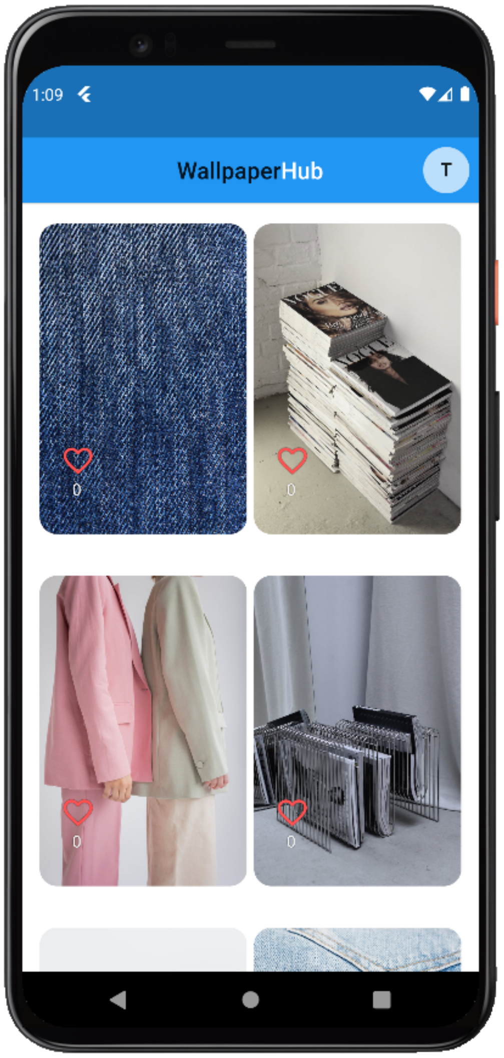
Sample GET request for wallpapers

```
curl -H "Authorization: YOUR_API_KEY" \
  "https://api.pexels.com/v1/search?query=nature&per_page=1"
```

Example response

```
{
  "total_results": 10000,
  "page": 1,
  "per_page": 1,
  "photos": [
    {
      "id": 3573351,
      "width": 3066,
      "height": 3968,
      "url":
"https://www.pexels.com/photo/trees-during-day-3573351/",
      "photographer": "Lukas Rodriguez",
      "photographer_url":
"https://www.pexels.com/@lukas-rodriguez-1845331",
      "photographer_id": 1845331,
      "avg_color": "#374824",
      "src": {
        "original":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png",
        "large2x":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&dpr=2&h=650&w=940",
        "large":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&h=650&w=940",
        "medium":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&h=350",
        "small":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&h=130",
        "portrait":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&fit=crop&h=1200&w=800",
        "landscape":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&fit=crop&h=627&w=1200",
        "tiny":
"https://images.pexels.com/photos/3573351/pexels-photo-3573351.png?au
to=compress&cs=tinysrgb&dpr=1&fit=crop&h=200&w=280"
      },
      "liked": false,
      "alt": "Brown Rocks During Golden Hour"
    }
  ],
  "next_page":
"https://api.pexels.com/v1/search/?page=2&per_page=1&query=nature"
}
```

OUTPUT



RESULT

A mobile application for data handling and connectivity via REST to backend services hosted in cloud (Pexels API). This is integrated with the mini project.

Application using GEO positioning, accelerometer and gesture based UI.

Date: 09-11-2023
Ex No: 14

AIM

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

DEPENDENCIES REQUIRED

No external dependencies required

ALGORITHM

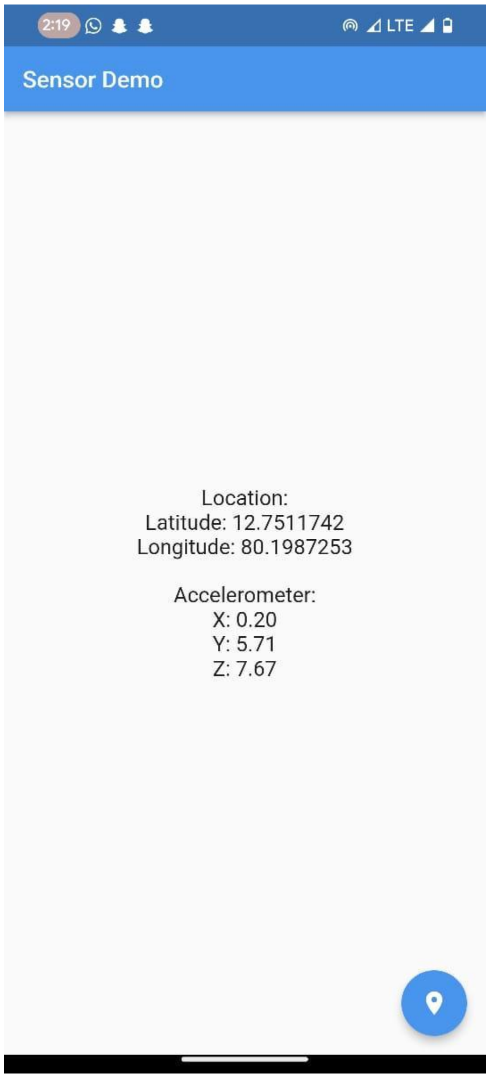
1. Create a new Flutter project with the necessary permissions declared in the AndroidManifest.xml (for Android) and Info.plist (for iOS) to access phone functionalities.
2. Add dependencies to pubspec.yaml for any required packages related to location services, accelerometer, and gesture recognition.
3. Run flutter pub get to fetch the dependencies.
4. Use the geolocator or a similar package to retrieve the device's current location.
5. Set up the necessary permissions and handle location updates, providing real-time or periodic updates to the user.
6. Utilize the sensors package or a similar one to access the device's accelerometer data.
7. Implement logic to interpret accelerometer data for specific gestures or actions, such as tilting or shaking the device.
8. Implement a rich and responsive UI that takes advantage of gestures.
9. Use the GestureDetector widget to capture various gestures like taps, swipes, and pinches.
10. Associate specific functionalities with these gestures, creating an interactive and intuitive user experience.
11. Design and implement a user interface that reflects the data from GEO positioning and accelerometer readings.
12. Display real-time information about the user's location.
13. Create visual elements that respond to gestures and accelerometer movements.
14. Test the application on real devices to ensure accurate geo positioning, accelerometer readings, and responsive gesture-based UI.
15. Implement error handling for scenarios where location services or accelerometer data might be unavailable.
16. Debug and refine the application for optimal performance.

SOURCE CODE (widgets)

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Sensor Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: SensorScreen(),
    );
  }
}

class SensorScreen extends StatefulWidget {
  @override
  _SensorScreenState createState() => _SensorScreenState();
}

class _SensorScreenState extends State<SensorScreen> {
  String positionInfo = 'Location not available';
  String accelerometerInfo = 'Accelerometer not available';

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

    // Initialize geolocation
    _getLocation();

    // Initialize accelerometer
    accelerometerEvents.listen((AccelerometerEvent event) {
      setState(() {
        accelerometerInfo =
            'Accelerometer:\nX: ${event.x.toStringAsFixed(2)}\nY:
${event.y.toStringAsFixed(2)}\nZ: ${event.z.toStringAsFixed(2)}';
      });
    });
  }

  Future<void> _getLocation() async {
    try {
      Position position = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.best,
      );

      setState(() {
        positionInfo =
            'Location:\nLatitude: ${position.latitude}\nLongitude:
${position.longitude}';
      });
    } catch (e) {
      setState(() {
        positionInfo = 'Error getting location: $e';
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Sensor Demo'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              positionInfo,
              style: TextStyle(fontSize: 18),
              textAlign: TextAlign.center,
            ),
            SizedBox(height: 20),
            Text(
              accelerometerInfo,
              style: TextStyle(fontSize: 18),
              textAlign: TextAlign.center,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          _getLocation();
        },
        child: Icon(Icons.location_on),
        ),
      );
    }
}
```

OUTPUT



```
Location:
Latitude: 12.7511742
Longitude: 80.1987253

Accelerometer:
X: 0.20
Y: 5.71
Z: 7.67
```

RESULT

A mobile application that will take advantage of underlying phone functionality including GEO positioning, accelerometer, and rich gesture-based UI handling has been successfully implemented.

Mobile application

Date: 16-11-2023
Ex No: 15

AIM

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

DEPENDENCIES REQUIRED

flutter_facebook_auth: ^6.0.2

ALGORITHM

1. Create a new Flutter project for your mobile application.
2. Set up necessary project files and folders.
3. Include necessary dependencies for integrating with Facebook and Twitter SDKs in the pubspec.yaml file.
4. Run flutter pub get to fetch the dependencies.
5. Design the main user interface of your mobile application using appropriate widgets like Scaffold, AppBar, and others.
6. Include sections for user authentication, app features, and social media integration.
7. Implement buttons or UI elements that users can interact with to connect their accounts with Facebook and Twitter.
8. Implement a user authentication system, allowing users to sign up or log in using traditional email/password or other methods.
9. Integrate Facebook login functionalities using their respective SDKs.
10. Handle user authorization to access necessary information from these platforms.
11. Implement features such as posting updates, sharing content, and retrieving user information.
12. Utilize SDK-provided widgets for seamless integration of social media features into your app.
13. Implement robust error handling for scenarios such as failed API calls, network issues, or user authentication problems.
14. Ensure the security of user data by following best practices provided by Facebook and Twitter, including token management and secure connections.
15. Test the application thoroughly to ensure a smooth user experience.
16. Consider aspects like user interface responsiveness, loading indicators during API calls, and proper feedback to users during social media interactions.

SOURCE CODE (widgets)

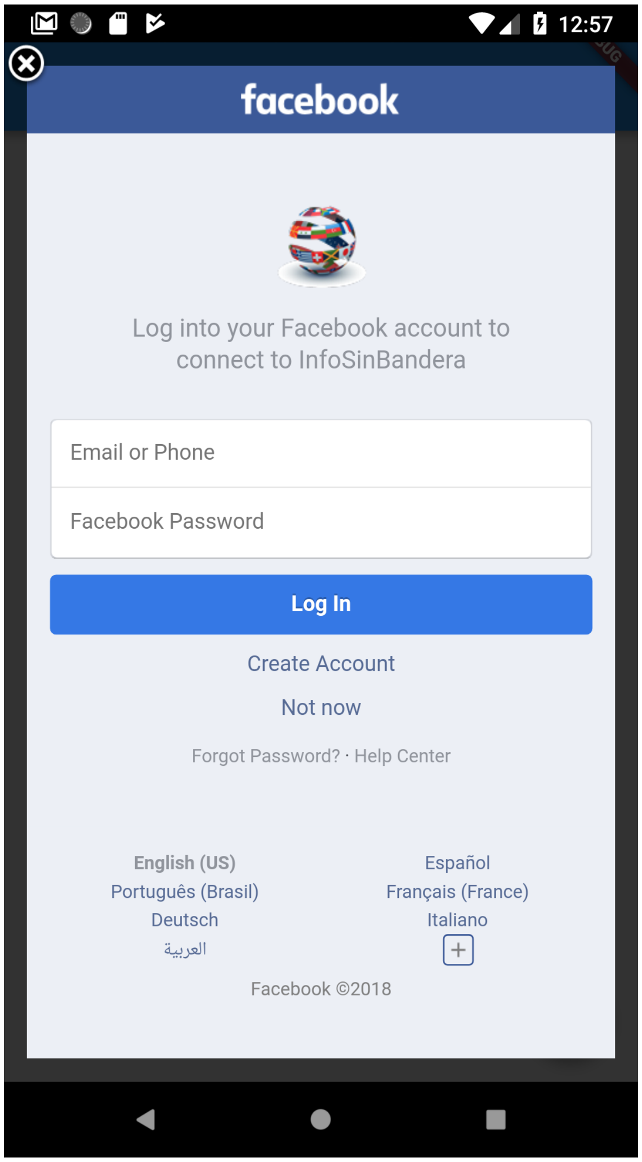
```
import 'package:flutter/material.dart';
import 'package:flutter_facebook_auth/flutter_facebook_auth.dart';

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  bool _isLoggedIn = false;
  Map _userObj = {};

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Codesundar"),
      ),
      body: Container(
        child: _isLoggedIn
            ? Column(
                children: [
                  Image.network(_userObj["picture"]["data"]["url"]),
                  Text(_userObj["name"]),
                  Text(_userObj["email"]),
                  TextButton(
                      onPressed: () {
                        FacebookAuth.instance.logOut().then((value) {
                          setState(() {
                            _isLoggedIn = false;
                            _userObj = {};
                          });
                        });
                      },
                      child: Text("Logout"))
                ],
              )
            : Center(
                child: ElevatedButton(
                  child: Text("Login with Facebook"),
                  onPressed: () async {
                    FacebookAuth.instance.login(
                        permissions: ["public_profile",
"email"]).then((value) {
```

```
FacebookAuth.instance.getUserData().then((userData) {
                        setState(() {
                          _isLoggedIn = true;
                          _userObj = userData;
                        });
                      });
                    });
                  },
                ),
              ),
      ),
    );
  }
}
```

Strings.xml

```
<span class="php"><span class="hljs-meta"><?</span>xml version=<span
class="hljs-string">"1.0"</span> encoding=<span
class="hljs-string">"utf-8"</span><span
class="hljs-meta">?></span></span>
<span class="hljs-tag"><<span
class="hljs-name">resources</span>></span>
    <span class="hljs-tag"><<span class="hljs-name">string</span>
<span class="hljs-attr">name</span>=<span
class="hljs-string">"app_name"</span>>c</span>odeSundar<span
class="hljs-tag"></<span class="hljs-name">string</span>></span>
    <span class="hljs-tag"><<span class="hljs-name">string</span>
<span class="hljs-attr">name</span>=<span
class="hljs-string">"facebook_app_id"</span>></span>000000000000<span
class="hljs-tag"></<span class="hljs-name">string</span>></span>
    <span class="hljs-tag"><<span class="hljs-name">string</span>
<span class="hljs-attr">name</span>=<span
class="hljs-string">"fb_login_protocol_scheme"</span>></span>fb0000000
000000<span class="hljs-tag"></<span
class="hljs-name">string</span>></span>
<span class="hljs-tag"></<span
class="hljs-name">resources</span>></span>
```

OUTPUT



12:57

facebook

Log into your Facebook account to connect to InfoSinBandera

Email or Phone

Facebook Password

Log In

Create Account

Not now

Forgot Password? · Help Center

English (US)
Español
Português (Brasil)
Français (France)
Deutsch
Italiano
العربية
+

Facebook ©2018

RESULT

An application for integrating mobile applications in the market, including social networking software integration with Facebook and Twitter has been successfully implemented.