Software Tools 4

Assignment 8

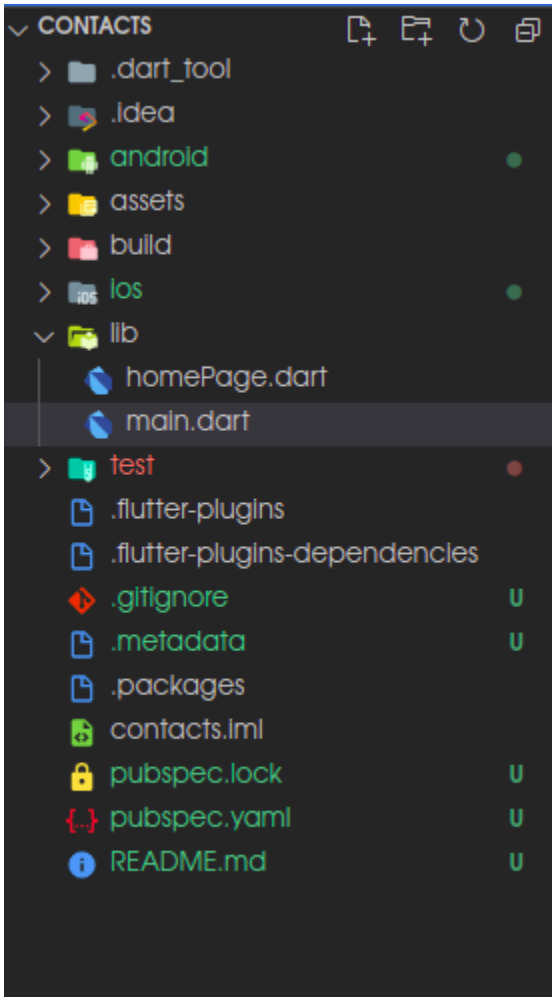
Krunal Rank
U18CO081

Develop a Roulette Game for Android.

Answer:

Tech Stack used :
Dart
Flutter SDK

Project Directory Structure:



Code:

./lib/main.dart:

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

import 'homePage.dart';

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

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Contacts',
     theme: ThemeData(
       primaryColor: Color(0xff3f51b5),
       accentColor: Color(0xff3f51b5),
       visualDensity: VisualDensity.adaptivePlatformDensity,
     ),
     home: HomePage(title: 'Contacts'),
   );
 }
}
```

./lib/homePage.dart:

```
import 'dart:math';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:sqflite/sqflite.dart';

class HomePage extends StatefulWidget {
 HomePage({Key key, this.title}) : super(key: key);

 final String title;

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

class _MyHomePageState extends State<HomePage>
   with SingleTickerProviderStateMixin {
 final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

 var nameTextController = new TextEditingController();
 var contactTextController = new TextEditingController();
 var emailTextController = new TextEditingController();
 var addressTextController = new TextEditingController();

 bool isLoading = true;
 String name = "", contact = "", address = "", email = "";
 List<Map> searchResults = List<Map>();
 int index = 0;
 Database db;
 var databasesPath;

 connectToDatabase() async {
   db = await openDatabase('contacts.db');
   await db.execute(
       'CREATE TABLE IF NOT EXISTS Contacts(Name TEXT, Contact TEXT PRIMARY KEY, Email
TEXT, Address TEXT)');
   this.setState(() {
     isLoading = false;
   });
   print(databasesPath);
 }
```

```dart
@override
void initState() {
  super.initState();
  connectToDatabase();
}

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

String validateName(String value) {
  if (value.length <= 0 || value.length >= 30)
    return 'Please enter a non empty Name with less than 30 characters!';
  final alpha = RegExp(r'^[a-zA-Z ]+$');
  if (alpha.hasMatch(value)) return null;
  return 'Please enter a valid Name!';
}

String validateContact(String value) {
  if (value.length <= 9 || value.length >= 13)
    return 'Please enter valid Phone Number!';
  final numeric = RegExp(r'^[0-9+]+$');
  if (numeric.hasMatch(value)) return null;
  return 'Please enter valid Phone Number!';
}

String validateEmail(String value) {
  final emailRegEx = RegExp(
      r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+");
  if (emailRegEx.hasMatch(value)) return null;
  return 'Please enter valid Email!';
}

String validateAddress(String value) {
  if (value.length <= 0 || value.length > 70)
    return 'Enter non empty valid Address with at most 70 characters';
  final alphanumeric = RegExp(r'^[a-zA-Z 0-9,:-]+$');
  if (alphanumeric.hasMatch(value)) return null;
  return 'Please enter a valid Address!';
}

showToast(msg) {
  Fluttertoast.showToast(
```

```
      msg: msg,
      toastLength: Toast.LENGTH_SHORT,
      gravity: ToastGravity.BOTTOM,
      backgroundColor: Color(0xff3f51b5),
      timeInSecForIosWeb: 1,
      fontSize: 16.0);
}

addEntry() async {
  if (validateName(name) != null ||
      validateEmail(email) != null ||
      validateContact(contact) != null ||
      validateAddress(address) != null) {
    showToast('Please enter valid Details!');
    return;
  }
  await db.transaction((txn) async {
    try {
      String query =
          "INSERT INTO Contacts(NAME,CONTACT,EMAIL,ADDRESS)
VALUES('$name','$contact','$email','$address');";
      await txn.rawInsert(query);
      showToast('Inserted Details in Database!');
    } catch (e) {
      showToast(
          'Failed to Insert Details in Database! Make sure the Contact is unique!');
    }
  });
}

removeEntry() async {
  if ((name.length != 0 && validateName(name) != null) ||
      (contact.length != 0 && validateContact(contact) != null) ||
      (email.length != 0 && validateEmail(email) != null) ||
      (address.length != 0 && validateAddress(address) != null)) {
    showToast(
        'Please make sure that the values you entered as a Delete Parameter are
valid!');
    return;
  }

  if (name.length == 0 &&
      contact.length == 0 &&
      email.length == 0 &&
```

```
      address.length == 0) {
    showToast('Please enter at least 1 value as a Delete Parameter!');
    return;
  }
  String whereQuery = "";
  if (name.length != 0) whereQuery += "NAME LIKE '$name' ";
  if (email.length != 0)
    whereQuery += whereQuery.length == 0
        ? "EMAIL = '$email' "
        : "AND EMAIL = '$email' ";
  if (contact.length != 0)
    whereQuery += whereQuery.length == 0
        ? "CONTACT = '%contact' "
        : "AND CONTACT = '$contact' ";
  if (address.length != 0)
    whereQuery += whereQuery.length == 0
        ? "ADDRESS = '$address' "
        : "AND ADDRESS = '$address' ";

  final count = await db.rawDelete('DELETE FROM Contacts WHERE $whereQuery');

  if (count == 0) {
    showToast('No Result Found using given Delete Parameters!');
    return;
  }
  showToast('$count Record(s) deleted!');
  return;
}

updateEntry() async {
  if (searchResults.length == 0) {
    showToast('Search the Record that you need to Update first!');
    return;
  }
  if (validateName(name) != null ||
      validateEmail(email) != null ||
      validateContact(contact) != null ||
      validateAddress(address) != null) {
    showToast('Please enter valid Details to be updated!');
    return;
  }

  var whereQuery =
```

```
      "NAME ='${searchResults[index]['NAME']}' AND CONTACT
='${searchResults[index]['CONTACT']}' AND EMAIL = '${searchResults[index]['EMAIL']}'
AND ADDRESS = '${searchResults[index]['ADDRESS']}'";
  var setQuery =
      "NAME = '$name',CONTACT='$contact',EMAIL='$email',ADDRESS='$address' ";

  final count =
      await db.rawUpdate('UPDATE Contacts SET $setQuery WHERE $whereQuery');
  if (count == 0) {
    showToast('Failed to update Record! Please modify at least one value!');
    return;
  }
  showToast('Record Updated Successfully!');
  return;
}

searchEntry() async {
  if ((name.length != 0 && validateName(name) != null) ||
      (contact.length != 0 && validateContact(contact) != null) ||
      (email.length != 0 && validateEmail(email) != null) ||
      (address.length != 0 && validateAddress(address) != null)) {
    showToast(
        'Please make sure that the values you entered as a Search Parameter are
valid!');
    return;
  }

  if (name.length == 0 &&
      contact.length == 0 &&
      email.length == 0 &&
      address.length == 0) {
    showToast('Please enter at least 1 value as a Search Parameter!');
    return;
  }
  String whereQuery = "";
  if (name.length != 0) whereQuery += "NAME LIKE '%$name%' ";
  if (email.length != 0)
    whereQuery += whereQuery.length == 0
        ? "EMAIL LIKE '%$email%' "
        : "AND EMAIL LIKE '%$email%' ";
  if (contact.length != 0)
    whereQuery += whereQuery.length == 0
        ? "CONTACT LIKE '%$contact%' "
        : "AND CONTACT LIKE '%$contact%' ";
```

```
  if (address.length != 0)
    whereQuery += whereQuery.length == 0
        ? "ADDRESS LIKE '%$address%' "
        : "AND ADDRESS LIKE '%$address%' ";
  List<Map> list =
      await db.rawQuery('SELECT * FROM Contacts WHERE $whereQuery');
  if (list.length == 0) {
    showToast('No Results found!');
    return;
  }
  showToast('${list.length} Result(s) found!');
  this.setState(() {
    searchResults = list;
    index = 0;
    name = list[0]['NAME'];
    contact = list[0]['CONTACT'];
    email = list[0]['EMAIL'];
    address = list[0]['ADDRESS'];
  });
  nameTextController.text = name;
  contactTextController.text = contact;
  emailTextController.text = email;
  addressTextController.text = address;
}

shiftLeft() {
  final newIndex = (index - 1 + searchResults.length) % searchResults.length;
  this.setState(() {
    index = newIndex;
    name = searchResults[newIndex]['NAME'];
    contact = searchResults[newIndex]['CONTACT'];
    email = searchResults[newIndex]['EMAIL'];
    address = searchResults[newIndex]['ADDRESS'];
  });
  nameTextController.text = name;
  contactTextController.text = contact;
  emailTextController.text = email;
  addressTextController.text = address;
}

shiftRight() {
  final newIndex = (index + 1) % searchResults.length;
  this.setState(() {
    index = newIndex;
```

```
    name = searchResults[newIndex]['NAME'];
    contact = searchResults[newIndex]['CONTACT'];
    email = searchResults[newIndex]['EMAIL'];
    address = searchResults[newIndex]['ADDRESS'];
  });
  nameTextController.text = name;
  contactTextController.text = contact;
  emailTextController.text = email;
  addressTextController.text = address;
}

@override
Widget build(BuildContext context) {
  final screenHeight = MediaQuery.of(context).size.height;
  final screenWidth = MediaQuery.of(context).size.width;
  return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
          child: isLoading
              ? CircularProgressIndicator()
              : Form(
                  key: _formKey,
                  autovalidate: true,
                  child: SingleChildScrollView(
                      child: Column(
                    children: [
                      Padding(
                        padding: EdgeInsets.all(screenHeight * 0.01),
                        child: TextFormField(
                          controller: nameTextController,
                          keyboardType: TextInputType.name,
                          decoration: const InputDecoration(
                            icon: Icon(Icons.person),
                            border: OutlineInputBorder(),
                            hintText: 'What do people call you?',
                            labelText: 'Name *',
                          ),
                          validator: validateName,
                          onChanged: (value) {
                            setState(() {
                              name = value;
                            });
```

```
        },
      ),
    ),
    Padding(
      padding: EdgeInsets.all(screenHeight * 0.01),
      child: TextFormField(
        controller: contactTextController,
        keyboardType: TextInputType.number,
        decoration: const InputDecoration(
          icon: Icon(Icons.phone),
          border: OutlineInputBorder(),
          hintText: 'Your Phone Number',
          labelText: 'Contact Number *',
        ),
        validator: validateContact,
        onChanged: (value) {
          setState(() {
            contact = value;
          });
        },
      ),
    ),
    Padding(
      padding: EdgeInsets.all(screenHeight * 0.01),
      child: TextFormField(
        controller: emailTextController,
        keyboardType: TextInputType.emailAddress,
        decoration: const InputDecoration(
          icon: Icon(Icons.email),
          border: OutlineInputBorder(),
          hintText: 'Your Email Address',
          labelText: 'Email Address *',
        ),
        validator: validateEmail,
        onChanged: (value) {
          setState(() {
            email = value;
          });
        },
      ),
    ),
    Padding(
      padding: EdgeInsets.all(screenHeight * 0.01),
      child: TextFormField(
```

```
        controller: addressTextController,
        keyboardType: TextInputType.streetAddress,
        decoration: const InputDecoration(
          icon: Icon(Icons.pin_drop),
          border: OutlineInputBorder(),
          hintText: 'Your Address',
          labelText: 'Address *',
        ),
        validator: validateAddress,
        maxLines: 4,
        onChanged: (value) {
          setState(() {
            address = value;
          });
        },
      ),
    ),
    searchResults.length <= 1
        ? Container()
        : Padding(
            padding: EdgeInsets.all(screenHeight * 0.01),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Padding(
                    padding:
                        EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: shiftLeft,
                      child: Icon(Icons.chevron_left),
                    )),
                Padding(
                    padding:
                        EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: shiftRight,
                      child: Icon(Icons.chevron_right),
                    )),
              ],
            )),
    Padding(
        padding: EdgeInsets.all(screenHeight * 0.01),
        child: Text(
```

```
                                'Note : While Searching Entries, enter only those
values that you want to be searched on.'))
                      ],
                    )))),
        floatingActionButton: isLoading
            ? null
            : Row(mainAxisAlignment: MainAxisAlignment.end, children: [
                Padding(
                    padding: EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: addEntry,
                      child: Icon(Icons.add),
                    )),
                Padding(
                    padding: EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: removeEntry,
                      child: Icon(Icons.remove),
                    )),
                Padding(
                    padding: EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: updateEntry,
                      child: Icon(Icons.update),
                    )),
                Padding(
                    padding: EdgeInsets.all(screenWidth * 0.01),
                    child: FloatingActionButton(
                      onPressed: searchEntry,
                      child: Icon(Icons.search),
                    )),
              ]) // This trailing comma makes auto-formatting nicer for build methods.
        );
  }
}
```

Screenshots: