EXPERIMENT NO: 01

Aim: - To install and configure Flutter Environment.

Theory: - To install and run Flutter, your development environment must meet these minimum requirements:

- Operating Systems: Windows 7 SPI or later (64-bit), x86-64 based.
- Disk Space: 1.64 GB (does not include disk space for IDE/tools).
- Tools: Flutter depends on these tools being available in your environment.
  - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
  - Git for Windows 2.X, with the **Use Git from the Windows Command Prompt** option. If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

Get the Flutter SDK:

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

   flutter_windows_3.7.11-stable.zip

   For other release channels, and older builds, check out the SDK archive.

2. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\src\flutter).

Warning: Do not install Flutter to a path that contains special characters or spaces.

Warning: Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the Flutter repo on GitHub, and change branches or tags as needed.

For example:–

```
C:\src>git clone https://github.com/flutter/flutter.git -b stable
```

You are now ready to run Flutter commands in the Flutter Console.

Update your path:

If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:

- From the Start search bar, enter 'env' and select **Edit environment variables for your account**.
- Under **User variables** check if there is an entry called **Path**:
  - If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  - If the entry doesn't exist, create a new user variable named Path with the full path to flutter\bin as its value.

You have to close and reopen any existing console windows for these changes to take effect.

Note: As of Flutter's 1.19.0 dev release, the Flutter SDK contains the dart command alongside the flutter command so that you can more easily run Dart command-line programs. Downloading the Flutter SDK also downloads the compatible version of Dart, but if you've downloaded the Dart SDK separately, make sure that the Flutter version of dart is first in your path, as the two versions might not be compatible. The following command tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible.

```
C:\>where flutter dart
C:\path-to-flutter-sdk\bin\flutter
C:\path-to-flutter-sdk\bin\flutter.bat
C:\path-to-dart-sdk\bin\dart.exe            ::this should go after `C:\path-to-flutter-
sdk\bin\` commands
C:\path-to-flutter-sdk\bin\dart
C:\path-to-flutter-sdk\bin\dart.bat
```

As shown above, the command dart from the Flutter SDK doesn't come first. Update your path to use commands from C:\path-to-flutter-sdk\bin\ before commands from C:\path-to-dart-sdk\bin\ (in this case). After restarting your shell for the change to take effect, running the where command again should show that the flutter and dart commands from the same directory now come first

```
C:\>where flutter dart
C:\dev\src\flutter\bin\flutter
C:\dev\src\flutter\bin\flutter.bat
C:\dev\src\flutter\bin\dart
C:\dev\src\flutter\bin\dart.bat
C:\dev\src\dart-sdk\bin\dart.exe
```

However, if you are using PowerShell, in it where is an alias of Where-Object command, so you need to use where.exe instead.

```
      PS C:\> where.exe flutter dart
```

To learn more about the dart command, run dart -h from the command line, or see the dart tool page.

Run flutter doctor:

From a console window that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

```
C:\src\flutter>flutter doctor
```

This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

For example:

```
[-] Android toolchain - develop for Android devices
    • Android SDK at D:\Android\sdk
      ✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ
    • Try re-installing or updating your Android SDK,
      visit https://docs.flutter.dev/setup/#android-setup for detailed instructions.
```

The following sections describe how to perform these tasks and finish the setup process. Once you have installed any missing dependencies, you can run the flutter doctor command again to verify that you've set everything up correctly.

Note: If flutter doctor returns that either the Flutter plugin or Dart plugin of Android Studio are not installed, move on to Set up an editor to resolve this issue.

Warning: The Flutter tool may occasionally download resources from Google servers. By downloading or using the Flutter SDK you agree to the Google Terms of Service.

For example, when installed from GitHub (as opposed to from a prepackaged archive), the Flutter tool will download the Dart SDK from Google servers immediately when first run, as it is used to execute the flutter tool itself. This will also occur when Flutter is upgraded (e.g. by running the flutter upgrade command).

The flutter tool uses Google Analytics to report feature usage statistics and send crash reports. This data is used to help improve Flutter tools over time.

Flutter tool analytics are not sent on the very first run. To disable reporting, run flutter config --no-analytics. To display the current setting, use flutter config. If you opt out of analytics, an opt-out event is sent, and then no further information is sent by the Flutter tool.

Dart tools may also send usage metrics and crash reports to Google. To control the submission of these metrics, use the following options on the dart tool:

- --enable-analytics: Enables anonymous analytics.
- --disable-analytics: Disables anonymous analytics.

The Google Privacy Policy describes how data is handled by these services.

Android setup:

Note: Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. However, you can write your Flutter apps in a number of editors; a later step discusses that.

Install Android Studio:

1. Download and install Android Studio.
2. Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.

3. Run flutter doctor to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run flutter config --android-studio-dir=<directory> to set the directory that Android Studio is installed to.

Set up your Android device:

To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the Android documentation.
2. Windows-only: Install the Google USB Driver.
3. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
4. In the terminal, run the flutter devices command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your adb tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the ANDROID_SDK_ROOT environment variable to that installation directory.

Set up the Android emulator:

To prepare to run and test your Flutter app on the Android emulator, follow these steps:

1. Enable VM acceleration on your machine.
2. Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device…**
   - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device…**. (The **Android** submenu is only present when inside an Android project.)
   - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device…**
3. Choose a device definition and select **Next**.
4. Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image is recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable hardware acceleration.
6. Verify the AVD configuration is correct, and select **Finish**.

   For details on the above steps, see Managing AVDs.

7. In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

Agree to Android Licenses:

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

1. Make sure that you have a version of Java 11 installed and that your JAVA_HOME environment variable is set to the JDK's folder.

   Android Studio versions 2.2 and higher come with a JDK, so this should already be done.

2. Open an elevated console window and run the following command to begin signing licenses.
   $ flutter doctor --android-licenses
3. Review the terms of each license carefully before agreeing to them.
4. Once you are done agreeing with licenses, run flutter doctor again to confirm that you are ready to use Flutter.

Conclusion: Thus, successfully install and configure the flutter environment.

EXPERIMENT NO: 02

Aim:- To develop a simple Hello World Mobile App using flutter.

Theory:-
Scaffold: The Scaffold is designed to be a top-level container for a MaterialApp. This means that adding a Scaffold to each route on a Material app will provide the app with Material's basic visual layout structure.

Text: The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

Stateless: A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely.

Build: The build method of a stateless widget is typically only called in three situations: the first time the widget is inserted in the tree, when the widget's parent changes its configuration, and when an InheritedWidget it depends on changes.

runApp(): The runApp() function takes the given Widget and makes it the root of the widget tree.

Child: A child widget can itself be a Row, Column, or other complex widget. You can specify how a Row or Column aligns its children, both vertically and horizontally. You can stretch or constrain specific child widgets. You can specify how child widgets use the Row 's or Column 's available space.

Code:-

```
import 'package:flutter/material.dart';
        void main() {
 runApp(const MyApp());
}
class MyApp extends StatelessWidget {
 const MyApp({super.key});
 @override
 Widget build(BuildContext context) {
  return MaterialApp(

    title: 'Flutter Demo',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
      useMaterial3: true,
    ),
    home: Scaffold(appBar: AppBar(
      title:const Text('06_SAHILCHALKE - Exp23',
      style: TextStyle(
        color: Colors.black,
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
      ),
      ),
```

```
body: (
    Material(
   child:  Container(
  color: Color.fromARGB(255, 223, 113, 106),
  child: const Center(
 child: Text(
      "Hello World",
      style: TextStyle(
        fontSize: 40,
        color: Color.fromARGB(255, 0, 0, 0),
        fontWeight: FontWeight.bold
      ),
     ),
   ),
 ),
    )
   ),
   )
 );
}
```

Output:

Conclusion: Thus, successfully develop a mobile app using flutter.

EXPERIMENT NO: 03

Aim: To design Flutter UI by including common widgets

Theory:-

Container: The container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets.

Scaffold: The Scaffold is designed to be a top-level container for a MaterialApp. This means that adding a Scaffold to each route on a Material app will provide the app with Material's basic visual layout structure.

Text: The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

Stateless: A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely.

Build: The build method of a stateless widget is typically only called in three situations: the first time the widget is inserted in the tree, when the widget's parent changes its configuration, and when an InheritedWidget it depends on changes.

runApp(): The runApp() function takes the given Widget and makes it the root of the widget tree.

Child: A child widget can itself be a Row, Column, or other complex widget. You can specify how a Row or Column aligns its children, both vertically and horizontally. You can stretch or constrain specific child widgets. You can specify how child widgets use the Row 's or Column 's available space.

Row: A widget that displays its children in a horizontal array. he Row widget does not scroll (and in general it is considered an error to have more children in a Row than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

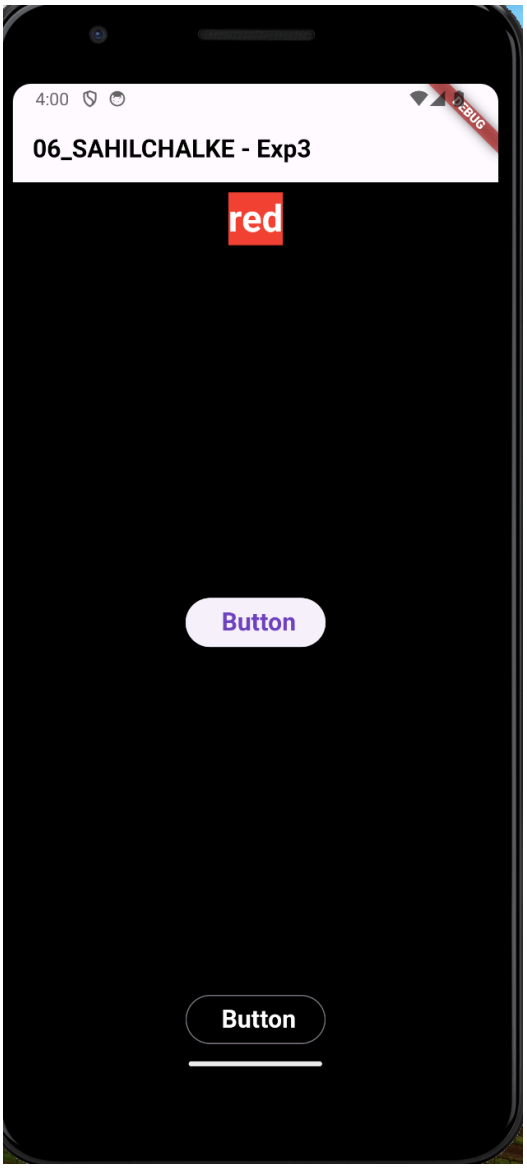
Column: A widget that displays its children in a vertical array. The Column widget does not scroll (and in general it is considered an error to have more children in a Column than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

Code:

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


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


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


 @override
 Widget build(BuildContext context) {
  return MaterialApp(

    title: 'Flutter Demo',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
      useMaterial3: true,
    ),
    home: Scaffold(appBar: AppBar(
      title:const Text('06_SAHILCHALKE - Exp3',
      style: TextStyle(
        color: Colors.black,
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
      ),
      ),

    body: (
      Material(

     child: Container(
    color: Color.fromARGB(255, 0, 0, 0),
    child: Column(
      mainAxisAlignment:
MainAxisAlignment.spaceBetween,


      children: [

        Center(
          child: TextButton(
            onLongPress: () => print("Long Pressed"),
            child: const Text(
              "red",
              style: TextStyle(
              fontSize: 30,

  ),
      )
    ),
    )
  );
```

```
fontWeight: FontWeight.bold,
              color: Colors.white,
              backgroundColor: Colors.red
            ),


            ),


            onPressed: (){
              print("Just pressed");
            },
          )


        ),


        const SizedBox(height: 20,),


        Center(
          child: ElevatedButton(
            onPressed: () => print("Clicked On Elevated
Button"),
            child: const Text(
              " Button",
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold
              ),
            ),
          ),
        ),


        const SizedBox(height: 20,),

        Center(
          child: OutlinedButton(
            onPressed: () => print("Clicked On Elevated
Button"),
            child: const Text(
              " Button",
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
                color: Colors.white
              ),
            ),
          ),
        )
      ],
    ),
```

Output:



Conclusion: Thus, successfully design flutter UI by including common widgets.

EXPERIMENT NO: 04

Aim:- To create an interactive Form using a form widget.

Theory:-
Scaffold: The Scaffold is designed to be a top-level container for a MaterialApp. This means that adding a Scaffold to each route on a Material app will provide the app with Material's basic visual layout structure.

Text: The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

Stateless: A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely.

Stateful: A stateful widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. Stateful widgets are useful when the part of the user interface you are describing can change dynamically.

Build: The build method of a stateless widget is typically only called in three situations: the first time the widget is inserted in the tree, when the widget's parent changes its configuration, and when an InheritedWidget it depends on changes.

runApp(): The runApp() function takes the given Widget and makes it the root of the widget tree.

Child: A child widget can itself be a Row, Column, or other complex widget. You can specify how a Row or Column aligns its children, both vertically and horizontally. You can stretch or constrain specific child widgets. You can specify how child widgets use the Row 's or Column 's available space.

Column: A widget that displays its children in a vertical array. The Column widget does not scroll (and in general it is considered an error to have more children in a Column than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

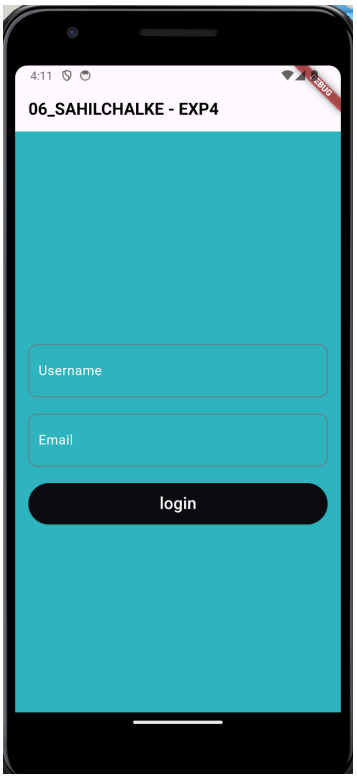
appBar: App bars are typically used in the Scaffold.appBar property, which places the app bar as a fixed-height widget at the top of the screen.

CODE:

```
import 'package:flutter/material.dart';
void main() {
 runApp(const MyApp());
}
 final GlobalKey<FormState> _formKey =
GlobalKey<FormState>();
 void _submitForm(){
  if(_formKey.currentState!.validate()){


  }
 }
 String? _validateEmail(value){
  if(value!.isEmpty){
   return "Please enter an email";
  }
  RegExp emailRegExp =
RegExp(r'/^[^\.\s][\w\-]+(\.[\w\-]+)*@([\w-]+\.)+[\w-]{2,}$/gm');
  if(!emailRegExp.hasMatch(value)){
   return  "Please enter a valid email";
  }
  return null;
 }
 String? _validatePassword(value){
  if(value!.isEmpty){
   return "Please enter the password";
  }
  if(value.length < 6){
   return 'Password length must be more than 60';
  }
  return null;
 }


class MyApp extends StatelessWidget {
 const MyApp({super.key});
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Flutter Demo',
   theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
     useMaterial3: true,
   ),
   home: Scaffold(appBar: AppBar(
    title:const Text('06_SAHILCHALKE - EXP4',
    style: TextStyle(

          TextFormField(
          decoration: InputDecoration(
           labelText: "Email",


           labelStyle: const TextStyle(
            color: Colors.white
           ),
           errorStyle: const TextStyle(
            fontSize: 18
           ),
           border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10.0),
           )
           ),
           style: const TextStyle(
            color: Colors.white
           ),
color: Colors.black,
    fontSize: 20,
    fontWeight: FontWeight.bold,
   ),
   ),
   ),
 body: (
  Material(
 child: Container(
color: Color.fromARGB(255, 48, 182, 192),
child: Padding(
 padding: const EdgeInsets.all(16.0),
 child: Form(
  key: _formKey,
  child: Column(
   mainAxisAlignment: MainAxisAlignment.center,
   children: [
    const SizedBox(height: 30,),
    TextFormField(
     decoration: InputDecoration(
      labelText: "Username",


      hintStyle: const TextStyle(
       color: Colors.white
      ),


      labelStyle: const TextStyle(
       color: Colors.white
      ),

      errorStyle: const TextStyle(
       fontSize: 18
      ),
      border: OutlineInputBorder(
       borderRadius: BorderRadius.circular(10.0),
      )
      ),


      style: const TextStyle(
       color: Colors.white
      ),
      validator: (value){
       if(value!.isEmpty){
        return "Please Enter a Username";
       }
       return null;
      },
    ),
    const SizedBox(height: 20,),
```

```
            validator: _validateEmail,
          ),
          const SizedBox(height: 20,),
          SizedBox(
            height: 50,
            width: double.infinity,
            child: ElevatedButton(
              onPressed: _submitForm,
              style:  ElevatedButton.styleFrom(
                backgroundColor: const Color.fromARGB(255,
9, 8, 8)
              ),
              child: const Text(
                "login",
                style: TextStyle(
                  fontSize: 20,
                  color: Colors.white
                ),
              ),
            ),
          )
        ],
      ),
    ),
  ),
    )
  ),
  )
);
```

OUTPUT:



EXPERIMENT NO: 05

AIM: To design a layout of Flutter App using Row widgets..

THEORY:

Theory:-

Row Widget: The Row widget in Flutter arranges its children widgets horizontally in a single line. It is useful for organizing UI elements in a row format, such as buttons, text fields, or images, within a parent widget.

MainAxisAlignment: With the Row widget, you can specify how its children are positioned along the main axis, which is horizontal by default. The MainAxisAlignment property allows you to align children horizontally, distributing space between them evenly or at the start, end, or center of the row..

CrossAxisAlignment: This property determines how children are aligned vertically within the Row. It enables you to align children at the start, end, or center of the row or stretch them to fill the vertical space.

Text Direction: In locales with right-to-left reading direction, such as Arabic or Hebrew, you can set the textDirection property to TextDirection.rtl to ensure proper layout of text and widgets within the Row.

Expanded Widget: To allow a child widget to expand and occupy the remaining space within the Row, you can wrap it with an Expanded widget. This is useful for creating flexible layouts where certain widgets should take up more space than others

runApp(): The runApp() function takes the given Widget and makes it the root of the widget tree.

Flexible Widget: Similar to Expanded, the Flexible widget can be used to control how a child widget flexes and shrinks within the Row. It provides more fine-grained control over how space is distributed among children.

MainAxisSize: The MainAxisSize property allows you to specify whether the Row should occupy the maximum available width (MainAxisSize.max) or only the minimum required width (MainAxisSize.min) along the main axis..

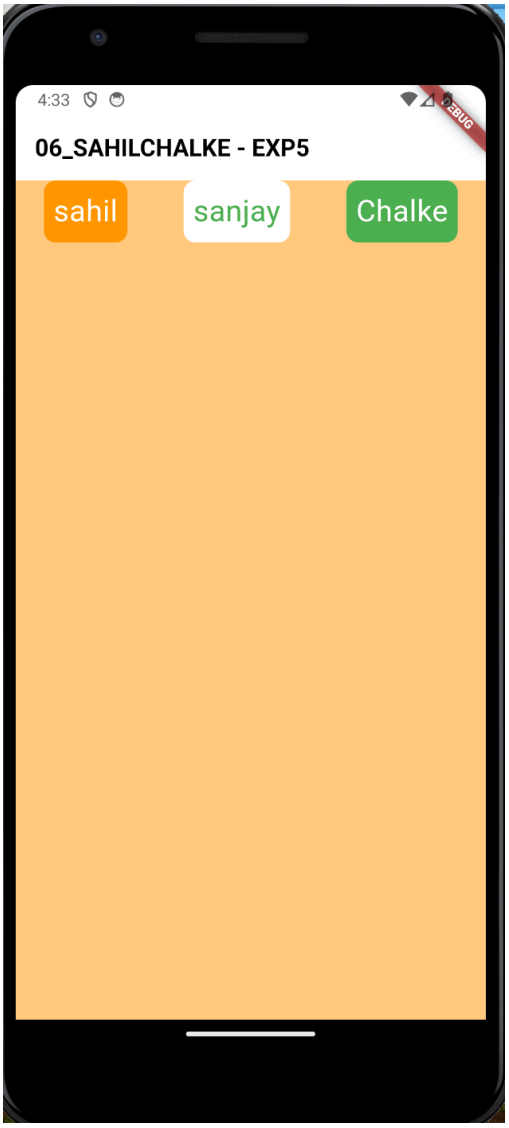
Spacing: You can add spacing between children in a Row using the SizedBox widget or by setting the crossAxisAlignment property to CrossAxisAlignment.center and adjusting the mainAxisAlignment property accordingly.

CODE:

The code for main.dart file.

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
final GlobalKey<FormState> _formKey =
GlobalKey<FormState>();
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSwatch(
          primarySwatch: Colors.orange,
        ).copyWith(
          secondary: Colors.white,
        ),
        useMaterial3: true,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text(
            '06_SAHILCHALKE - EXP5',
            style: TextStyle(
              color: Colors.black,
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
    body: Material(
          child: Row(
            mainAxisAlignment:
MainAxisAlignment.spaceAround,
            children: <Widget>[
              Container(
                decoration: BoxDecoration(
                  borderRadius:
BorderRadius.circular(10),
                  color: Colors.orange,
                ),
 Container(
                decoration: BoxDecoration(
                  borderRadius:
BorderRadius.circular(10),
                  color: Colors.white,
                ),
                child: const Padding(
                  padding: EdgeInsets.all(8.0),
                  child: Text(
                    "sanjay",
                    style: TextStyle(color: Colors.green,
fontSize: 25),
                  ),
                ),
              ),
              Container(
                decoration: BoxDecoration(
                  borderRadius:
BorderRadius.circular(10),
                  color: Colors.green,
                ),
                child: const Padding(
                  padding: EdgeInsets.all(8.0),
                  child: Text(
                    "Chalke",
                    style: TextStyle(color: Colors.white,
fontSize: 25),
                  ),
                ),
              )
            ],
          ),
        ),
      ),
    );
}
}
```

OUTPUT:



Conclusion: Thus, successfully design a layout of a flutter app using row widget.

EXPERIMENT NO: 06

AIM:To design a Flutter App using Column widgets.

THEORY:

Column Widget: The Column widget in Flutter organizes its children widgets vertically in a single column. It serves as a fundamental layout widget for arranging UI elements from top to bottom within a parent widget

MainAxisAlignment: Similar to the Row widget, the Column widget allows you to specify how its children are positioned along the main axis, which is vertical by default. The MainAxisAlignment property enables alignment of children vertically, distributing space between them evenly or at the start, end, or center of the column.

CrossAxisAlignment: This property determines how children are aligned horizontally within the Column. It provides options to align children at the start, end, or center of the column or stretch them to fill the horizontal space.

Text Direction: Just like with the Row widget, the textDirection property can be set to TextDirection.rtl in locales with right-to-left reading direction, ensuring proper layout of text and widgets within the Column..

Expanded Widget: To allow a child widget to expand and occupy the remaining space within the Column, the Expanded widget can be used. It's particularly useful for creating flexible layouts where certain widgets should take up more space than others vertically.

Flexible Widget: Similar to Expanded, the Flexible widget can control how a child widget flexes and shrinks within the Column. It provides fine-grained control over how space is distributed among children vertically.

MainAxisSize: The MainAxisSize property allows you to specify whether the Column should occupy the maximum available height (MainAxisSize.max) or only the minimum required height (MainAxisSize.min) along the main axis.

Spacing: You can add spacing between children in a Column using the SizedBox widget

Nested Columns: Columns can be nested within other Columns or within Rows to create complex layouts. This allows for the creation of grid-like structures or more intricate UI designs vertically.

CODE:

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}
final GlobalKey<FormState> _formKey =
GlobalKey<FormState>();
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSwatch(
          primarySwatch: Colors.orange,
        ).copyWith(
          secondary: Colors.white,
        ),
        useMaterial3: true,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text(
            '06_SAHILCHALKE - EXP5',
            style: TextStyle(
              color: Colors.black,
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        body: Material(
          child:  Column(
                      mainAxisAlignment:
MainAxisAlignment.spaceAround,
                      children: <Widget>[
                      Container(
                                  decoration: BoxDecoration(
                                              borderRadius:
BorderRadius.circular(10), color: Colors.green),
                                  child: const Padding(
                                  padding: EdgeInsets.all(8.0),
                                  child: Text(
                                              "sahil",
                                              style: TextStyle(color:
Colors.white, fontSize: 25),
                                  ),
                                  ),
                      ),
                      Container(
                                  decoration: BoxDecoration(
                                              borderRadius:
BorderRadius.circular(10), color: Colors.green),
                                  child: const Padding(
                                  padding: EdgeInsets.all(8.0),
                                  child: Text(
                                              "sanjay",
                                              style: TextStyle(color:
Colors.white, fontSize: 25),
                                  ),
                                  ),
                      Container(
                                  decoration: BoxDecoration(
                                              borderRadius:
BorderRadius.circular(10), color: Colors.green),
                                  child: const Padding(
                                  padding: EdgeInsets.all(8.0),
                                  child: Text(
                                              "chalke",
                                              style: TextStyle(color:
Colors.white, fontSize: 25),
                                  ),
                                  ),
                      )
                      ],
            ),
        ),
      ),
    );
  }
}
```

OUTPUT:

Conclusion : Thus, successfully implemented icons, images, charts in flutter app

EXPERIMENT NO: 07

AIM: To include icons, images, charts in flutter app.

THEORY:

Icon: A graphical icon widget drawn with a glyph from a font described in an IconData such as material's predefined IconDatas in Icons.

Image: A widget that displays an image.

Charts: The Flutter Charts package is a data visualization library written natively in Dart for creating beautiful, animated and high-performance charts, which are used to craft high-quality mobile app user interfaces using Flutter.

Scaffold: The Scaffold is designed to be a top-level container for a MaterialApp. This means that adding a Scaffold to each route on a Material app will provide the app with Material's basic visual layout structure.

Text: The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

Stateless: A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely.

Build: The build method of a stateless widget is typically only called in three situations: the first time the widget is inserted in the tree, when the widget's parent changes its configuration, and when an InheritedWidget it depends on changes.

runApp(): The runApp() function takes the given Widget and makes it the root of the widget tree.

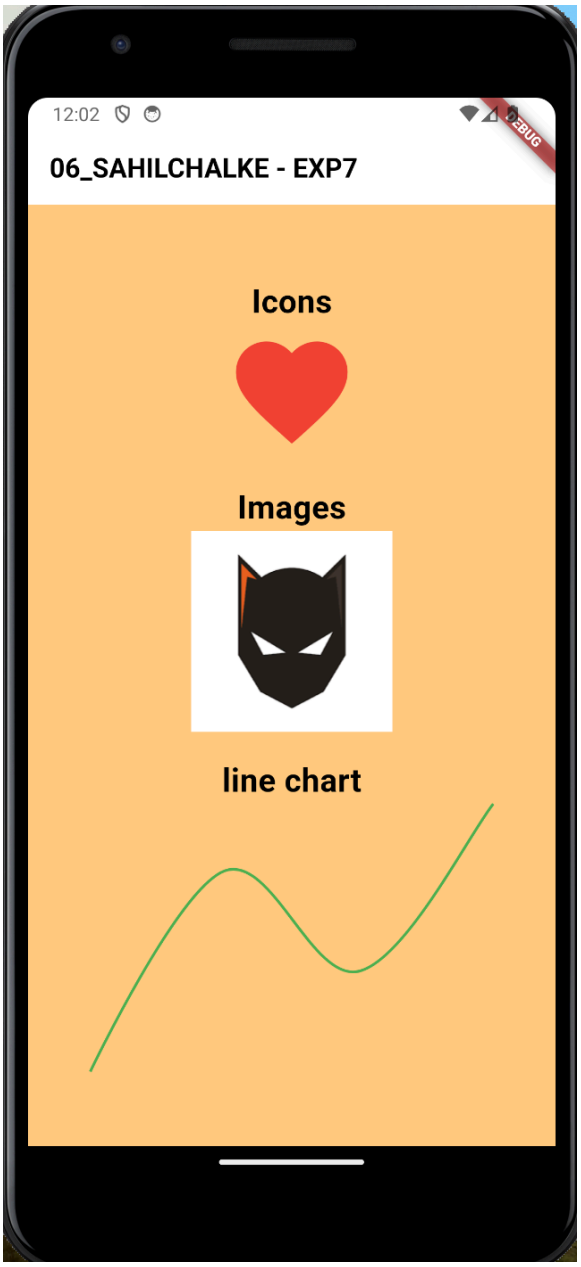
Child: A child widget can itself be a Row, Column, or other complex widget. You can specify how a Row or Column aligns its children, both vertically and horizontally. You can stretch or constrain specific child widgets. You can specify how child widgets use the Row 's or Column 's available space.

Column: A widget that displays its children in a vertical array. The Column widget does not scroll (and in general it is considered an error to have more children in a Column than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

CODE:-

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

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

final GlobalKey<FormState> _formKey =
GlobalKey<FormState>();

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSwatch(
          primarySwatch: Colors.orange,
        ).copyWith(
          secondary: Colors.white,
        ),
        useMaterial3: true,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text(
            '06_SAHILCHALKE - EXP8',
            style: TextStyle(
              color: Colors.black,
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
    body: Material(
          child: Center(
            child: Column(
              mainAxisAlignment:
MainAxisAlignment.center,
              children: <Widget>[
                Icon(
                  Icons.access_alarm,
                  size: 100,
                  color: Colors.red,
                ),
                SizedBox(height: 20),
                Image.network(

'https://avatars.githubusercontent.com/u/109215419?v=4',
                  width: 150,
                ),
                SizedBox(height: 20),
                Container(
                  width: 300,
                  height: 200,
                  child: LineChart(
                    LineChartData(
                      titlesData: FlTitlesData(
                        leftTitles: SideTitles(showTitles: false),
                        bottomTitles: SideTitles(showTitles:
false),
                      ),
                      borderData: FlBorderData(show: false),
                      gridData: FlGridData(show: false),
                      lineBarsData: [
                        LineChartBarData(
                          spots: [
                            FlSpot(0, 0),
                            FlSpot(1, 3),
                            FlSpot(2, 1.5),
                            FlSpot(3, 4),
                          ],
                          isCurved: true,
                          colors: [Colors.blue],
                          dotData: FlDotData(show: false),
                        ),
                      ],
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
      );
  }
}
```

OUTPUT:-



Conclusion:hence, we have performed an experiment to include ions ,images,charts in flutter app

EXPERIMENT NO: 08

AIM: To apply navigation, routing, and gestures in flutter app.

THEORY:

Navigation and Routing:
Navigation refers to the movement between different screens or pages within an app.
Routing is the mechanism that defines how navigation is handled, including which screen to display when navigating from one screen to another.
In Flutter, navigation and routing are typically managed using a Navigator widget and a set of routes defined within a MaterialApp or CupertinoApp widget.
You can use named routes to define mappings between route names and screen widgets, making it easier to navigate between screens using their names.

Gesture Detection and Handling:

Flutter provides built-in support for detecting and handling gestures such as taps, swipes, and drags.

Gestures are detected using specialized widgets like GestureDetector, which listens for specific types of user input.

Once a gesture is detected, you can define callback functions to handle the corresponding events, such as onTap, onDoubleTap, onLongPress, etc.

Gesture handling allows you to create interactive user interfaces and implement features like button presses, drag-and-drop functionality, and swipe-based navigation.

Implementation Steps:

Define Screens: Create individual screen widgets for each distinct view or page in your app.

Define Routes: Define a map of named routes in your app, mapping route names to corresponding screen widgets.

Implement Navigation: Use the Navigator widget to navigate between screens by pushing and popping routes onto and off the navigation stack.

Gesture Detection: Wrap relevant widgets with GestureDetector widgets to detect and handle user gestures.

Implement Gesture Handling: Define callback functions to respond to gesture events, such as navigating to a different screen in response to a tap gesture.

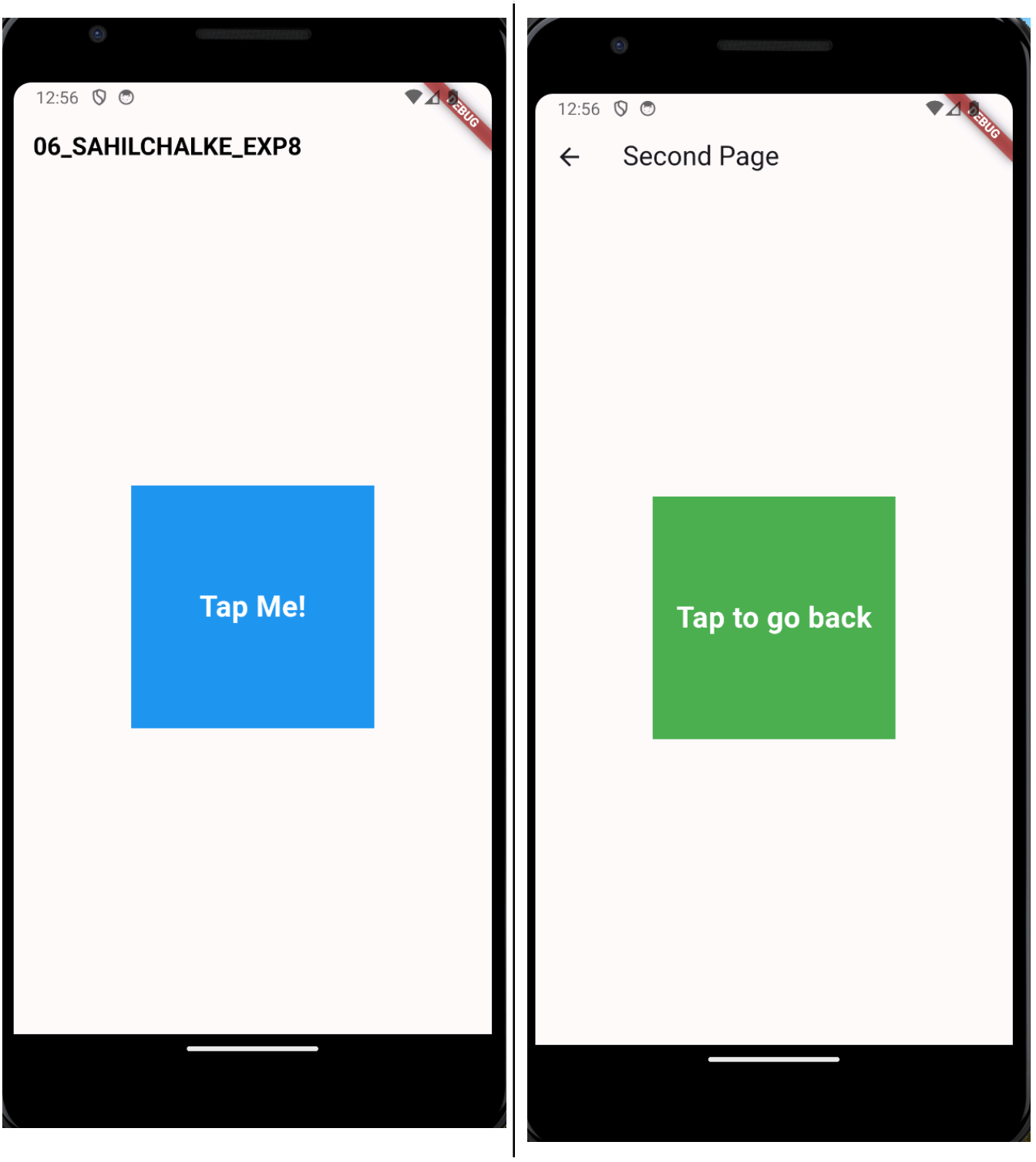
Testing and Debugging: Test your app to ensure that navigation and gestures work as expected, and debug any issues that arise.

Best Practices:
Keep navigation paths simple and intuitive for users to follow.
Provide visual feedback to indicate when gestures have been detected and actions are being performed.

CODE:

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


void main() {
  runApp(Experiment8());
}

class Experiment8 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => HomePage(),
        '/second': (context) => SecondPage(),
      },
    );
  }
}

class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('06_SAHILCHALKE_EXP8',
        style: TextStyle(
          fontSize: 20,
          color: Colors.black,
          fontWeight: FontWeight.bold,
        ),
        ),
      ),
    body: Center(
      child: GestureDetector(
        onTap: () {
          Navigator.pushNamed(context, '/second');
        },
        child: Container(
          width: 200,
          height: 200,
          color: Colors.blue,
          child: Center(
            child: Text(
              'Tap Me!',
              style: TextStyle(
                fontSize: 24,
                color: Colors.white,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ),
      ),
    ),
    );
  }
}

class SecondPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Second Page'),
      ),
      body: Center(
        child: GestureDetector(
          onTap: () {
            Navigator.pop(context);
          },
          child: Container(
            width: 200,
            height: 200,
            color: Colors.green,
            child: Center(
              child: Text(
                'Tap to go back',
                style: TextStyle(
                  fontSize: 24,
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

OUTPUT:



Conclusion: Thus, successfully implemented navigation ,routing, and gestures in flutter app.

EXPERIMENT NO: 09

AIM: To connect flutter UI with firebase database.

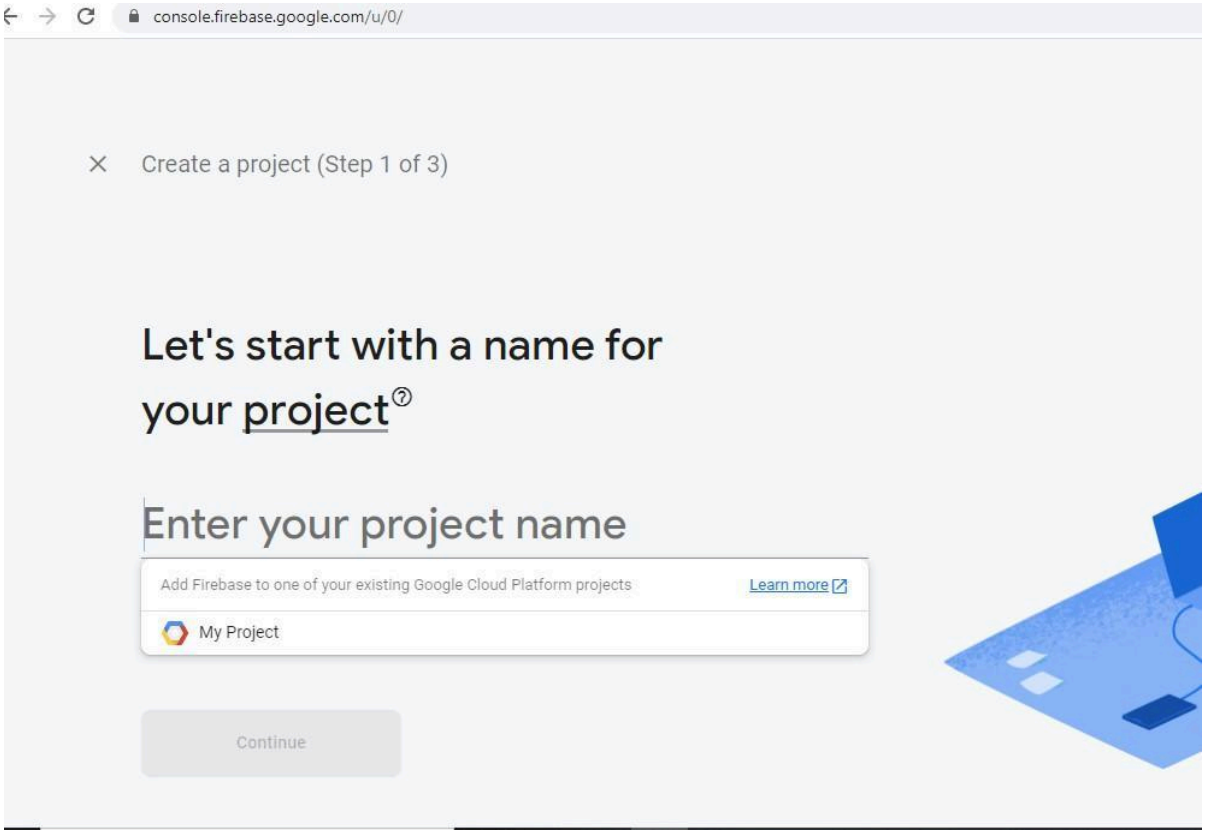
THEORY:

Open the pubspec.yaml file and add the following Flutter dependencies.

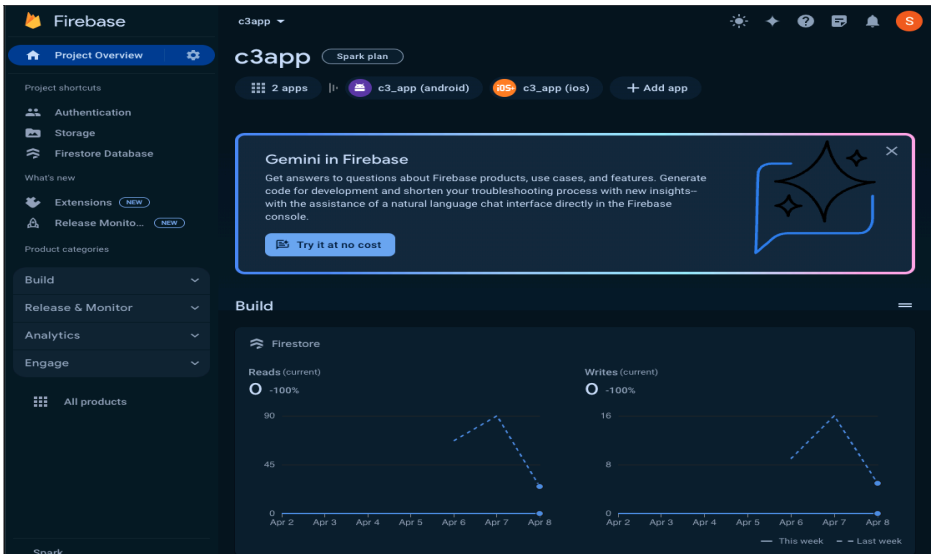
```
dev_dependencies:
  flutter_test:
    sdk: flutter
  firebase_database: ^4.4.0
  firebase_core : ^0.5.3
```

Use the pub get command to retrieve appropriate dependencies.

Next, open your browser and navigate to Firebase's website. We need to create a new firebase project, as shown below.



Add your Flutter application to Firebase by clicking on the Android icon.

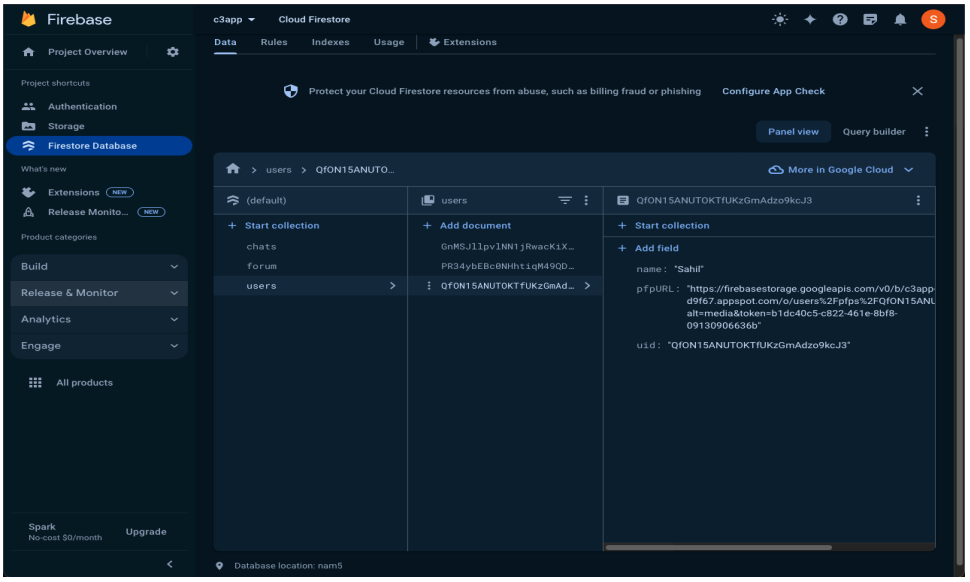


1. Initialize Firebase in Flutter App:

   In your Flutter app's main.dart file, initialize Firebase by calling Firebase.initializeApp():

   Code:

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```



Conclusion: Thus, successfully connected flutter UI with firebase database.

EXPERIMENT NO: 10

Aim - To create a responsive user interface using JQuery mobile for Ecommerce application.

Theory:-

Over the last few years the devices used to access web applications have grown like anything. We now have mobiles, tablets, desktops, laptops, TV etc. that can be used to access a web site. These all devices vary a lot in sizes. Writing individual applications to cater to all of these different sized and variety of devices is nearly impossible. So we need to design web sites in way that they adjust to the screen size on which they are displayed. The technique developed to solve this problem is Responsive Web Design. Responsive web design is a series of techniques and technologies that are combined to make delivering a single application across a range of devices as practical as possible. Below is an example of how Google news site looks on different devices.

Pillars of Responsive Design

Fluid Grids

A grid is a way to manage space in a layout in the web world. Fluid grids refer to flexible grid-based layouts that use relative sizing. Traditionally the grids were specified with fixed width columns with widths specified in pixels. But for responsive web design the grid columns widths should be relative to their containers. This helps the responsive web applications to adapt to different screen sizes. In responsive web design the widths are expressed in percentage most of the time.

Media Queries

Media queries assess the capabilities of device browser and apply styles based on the capabilities that match the query criteria. Media queries enable the web application to adjustitself for optimal viewing based on the browser capabilities.

Flexible Images and Media

This feature enables the images and other media to adapt according to the different device sizes and display resolutions. The most basic technique to adapt images and media is by usingscaling or CSS overflow property. Though these techniques adapt the image according to thedevice size but the image download still take up the user's mobile bandwidth. For better user experience one can use different set of images and media for different devices.

jQuery Mobile Framework

The jQuery Mobile Framework is an open source cross-platform UIframework.It is built usingHTML5, CSS3, and the very popular jQuery JavaScript library, and it follows Open Web standards. It provides touch-friendly UI widgets that are specially styled for mobile devices. Ithas a powerful theming framework to style your applications. It supports AJAX for various tasks, such as page navigation and transitions.

How jQuery Mobile supports RWD?

Unified UI

jQuery Mobile delivers unified user experience by designing to HTML5 and CSS3. A single jQuery codebase will render consistently across all supported platforms without needing anycustomizations.

Progressive Enhancement

Progressive enhancement defines layers of compatibility that allow any user to access the basic content, services, and functionality of a website, while providing an enhanced experience for browsers with better support of standards. jQuery Mobile is totally built usingthis technique. Below is an example of how a page built using jQuery Mobile will look on a basic phone as well as feature rich phone.

CSS Selector

jQuery Mobile has predefined set of CSS classes that it applies to the HTML elements depending on the current orientation or size of the device

```
$('.container').css('background-color', 'lightgray');
```

Orientation Classes

As with orientation classes, jQuery Mobile applies these breakpoint classes to the HTML elements depending on the screen size. These can be defined in the application CSS to override the default behavior. For example if we wanted to have different background imagesdepending on the screen size, the following CSS could be use

Flexible Layout

In jQuery Mobile most of the components and form elements are designed to be of flexible width so that they comfortably fit the width of any device. Additionally form elements and labels are represented differently based on the screen size. On smaller screens, labels are stacked on the top of form element while on wider screens, labels and elements are styled to be on the same line in 2 column grid layout.

Grid Layout

jQuery Mobile has built in flexible grid layout. The grids are 100% in width, completely invisible and do not have padding or margins. Hence they did not interfere with the styling of components placed inside them. jQuery Mobile grid layout can be used to create layout with two to five columns. The columns widths are adjusted according to the screen width and number of columns. There are predefined layouts defined for creating grid. These are named as ui-grid-a (2 columns grid), ui- grid-b (3 columns grid), ui-grid-c (4 columns grid) and ui-grid-d (5 columns grid).

Within the grid container, child elements are assigned ui-block-a/b/c/d/e in a sequential manner which makes each "block" element float side-by-side, forming the grid.

The default behavior of grids in jQuery Mobile is to lay the columns side by side. But for the application to respond to screen sizes we require the columns to be stacked on smaller screens. jQuery Mobile provides a CSS style to make this happen. The style name is ui- responsive. When this style is added to the grid container the grid columns would be stacked if the screen width is below 560px and on bigger screens the grid would be represented in multi-column layout.

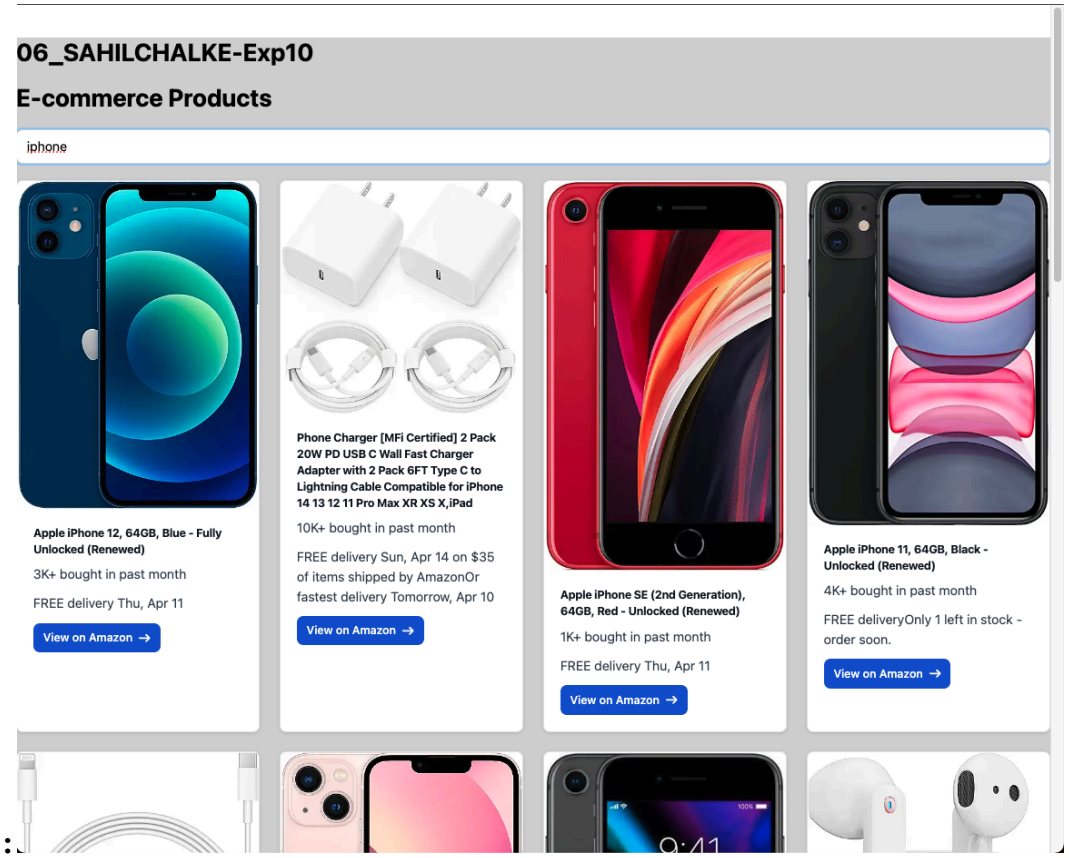
Responsive Tables

This is a newly added feature to jQuery Mobile. This allows us to display large amount of tabular data in a way that looks good on both mobiles and desktops. There are two modes of responsive tables:

Reflow: This mode vertically stacks the cells in rows by default so that the data could be easily readable on mobile phones. Additional style needs to be applied to make the table display in traditional rowcolumn format. This is the default mode for tables defined using data-role=’table’.

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import $ from 'jquery'; // Import jQuery

const App = () => {
  const [products, setProducts] = useState([]);
  const [search, setSearch] = useState('');

  useEffect(() => {
    const fetchData = async () => {
      const options = {
        method: 'GET',
        url:
'https://real-time-amazon-data.p.rapidapi.com/search',
        params: {
          query: search || 'cloths',
          page: '1',
          country: 'US',
          category_id: 'aps'
        },
        headers: {
          'X-RapidAPI-Key':
'03efd5a285mshe7331c9d611b7f7p143fd2jsn5b7e913e6da3',
          'X-RapidAPI-Host':
'real-time-amazon-data.p.rapidapi.com'
        }
      };

      try {
        const response = await
axios.request(options);
        console.log(response.data.data)
        setProducts(response.data.data.products);


        $('.container').css('background-color',
'lightgray');
      } catch (error) {
        console.error(error);
      }
    };

    fetchData();
  }, [search]);

  return (
    <div className="container mx-auto mt-10">
      <h1 className='text-3xl font-bold
mb-5'>06_SAHILCHALKE-Exp10</h1>
      <h1 className="text-3xl font-bold mb-5">E-commerce
Products</h1>
      <div className="mb-5">
        <input type="text" className="w-full px-3 py-2
border border-gray-300 rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-300" placeholder="Search
products" onChange={(e)=>setSearch(e.target.value)} />
      </div>
      <div className="grid grid-cols-1 sm:grid-cols-2
md:grid-cols-3 lg:grid-cols-4 gap-6">
        {products.map(product => (
          <Card key={product.id} product={product} />
        ))}
      </div>
    </div>
  );
};

const Card = ({ product }) => {
  return (
    <div className="bg-white border border-gray-200
rounded-lg shadow">
      <a href={product.product_url}>
        <img className="rounded-t-lg"
src={product.product_photo} alt={product.product_title} />
      </a>
      <div className="p-5">
        <a href={product.product_url}>
          <h5 className="mb-2 text-sm font-bold
tracking-tight text-gray-900">{product.product_title}</h5>
        </a>
        <p className="mb-3 font-normal
text-gray-700">{product.sales_volume}</p>
        <p className="mb-3 font-normal
text-gray-700">{product.delivery}</p>
        <a href={product.product_url} className="inline-flex
items-center px-3 py-2 text-sm font-medium text-center
text-white bg-blue-700 rounded-lg hover:bg-blue-800
focus:ring-4 focus:outline-none focus:ring-blue-300">
          View on Amazon
          <svg className="rtl:rotate-180 w-3.5 h-3.5 ms-2"
aria-hidden="true" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 14 10">
            <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M1 5h12m0
0L9 1m4 4L9 9" />
          </svg>
        </a>
      </div>
    </div>
  );
};

export default App;
```

Output:



:

Conclusion

As we can see jQuery Mobile is built keeping responsive design in mind. It provides a lot of out of box features to support responsive design. One of the advantages of jQuery Mobile is that it provides a base framework which is well tested to be working fine on various screen sizes and browsers. This makes it easier to develop applications using jQuery Mobile as one can concentrate on the application features without worrying about the cross browser compatibility and screen size issues.

EXPERIMENT NO: 11

Aim - To code and register a service worker & complete the install and activation process for a new service for the ecommerce PWA.

Theory –

Progressive Web Apps are a new type of web app that offers native-like capabilities, yet their reach and performance are indistinguishable from a website. This means that, unlike a native app, a Progressive Web App's UI won't be limited by the user's device or by the system.

E-commerce PWAs can also access the device's offline capabilities and can use system-level notification channels to meet the needs of users across platforms. They are optimized for all platforms, and they are fast, reliable, engaging, and usable in every major browser.

Progressive web apps in e-commerce are like native apps with a better build on the web. There is no need to download and install a separate app. PWA lives in a browser, providing a fast, more reliable, and engaging user experience. Progressive web apps have an installation experience that makes users feel like they are getting an app for the first time.

In addition to providing a smooth, native-like experience, progressive web apps are more secure. They use an application shell, called a service worker, to cache data and resources and serve it to the user. This allows the app to be installed on a new device without affecting the user's experience.

This runs the default task in gulpfile.babel.js which copies the project filesto the appropriatefolder and starts a server. Open your browser and navigate to localhost:8080. The app is a mock furniture website, "Modern Furniture Store". Several furniture items should display onthe front page.

When the app opens, confirm that a service worker is not registered at local host by checkingdeveloper tools. If there is a service worker at localhost, unregister it so it doesn't interfere with the lab.

Register the service worker
create a file **service-worker.js**

```
const CACHE_NAME =
'ecommerce-pwa-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/manifest.json',

];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => cache.addAll(urlsToCache))
  );
});
```

```
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.filter(cacheName => {
          return cacheName.startsWith('ecommerce-pwa-cache-') &&
cacheName !== CACHE_NAME;
        }).map(cacheName => caches.delete(cacheName))
      );
    })
  );
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        if (response) {
          return response;
        }

        return fetch(event.request);
      })
  );
});
```

Cache the application shell
call the worker in the index.js

// index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then(registration => {
        console.log('Service Worker registered with scope:', registration.scope);
      })
      .catch(error => {
        console.error('Service Worker registration failed:', error);
      });
  });
}

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

1. Use the cache-first strategy

   To complete in app/service-worker.js, write the code to respond to fetch requestswith the " cache, falling back to network" strategy. First, look for the response in the cache and if it exists, respond with the matching file. If the file does not exist, request the file from the network and cache a clone of the response. Save the file when you have completed this step

2. Delete outdated caches

   To complete in app/service-worker.js, write the code to delete unused caches inthe activate event handler. You should create an "allowlist" of caches currently in use that should not be deleted (such as the e-commerce-v1 cache). Use caches.keys() to get a list of the cache names. Then, inside Promise.all, map the array containing the cache names to a function that deletes each cache not in the allowlist. Save the file when you have completed this step.

3. Test it out

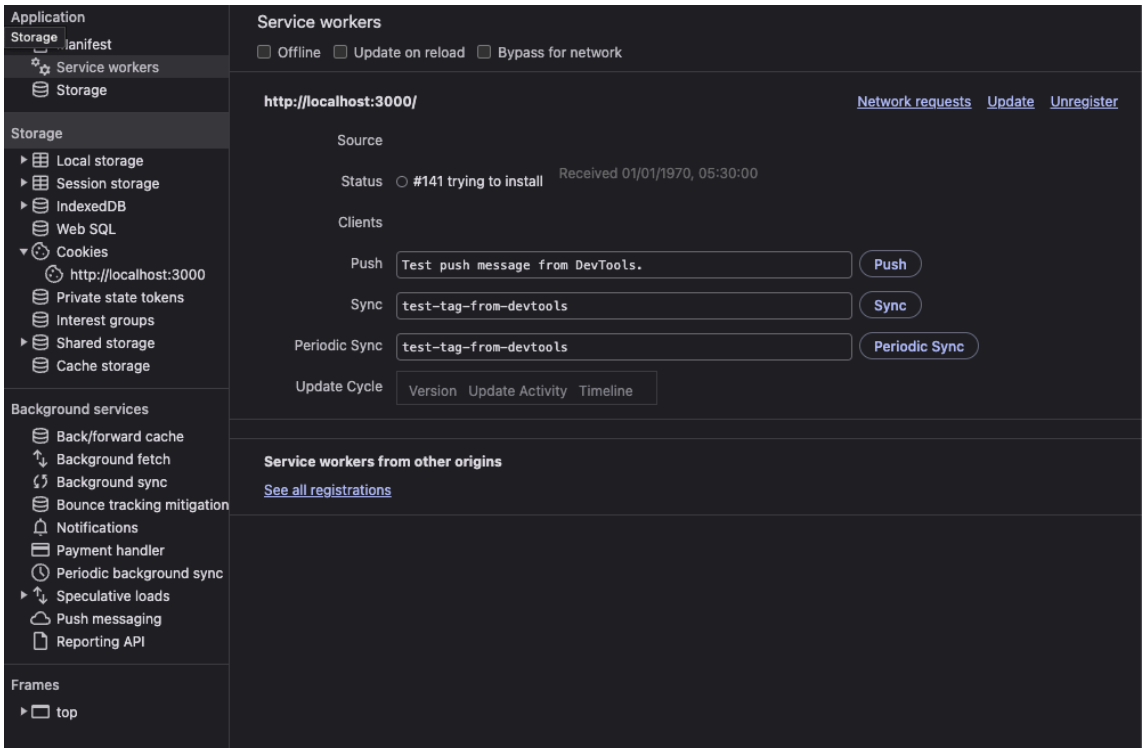
   To test the app, close any open instances of the app in your browser and stop the local server(ctrl+c).

   Run the following in the command line to clean out the old files in the dist folder, rebuild it, and serve the app:

```
npm run serve
```

Open the browser and navigate to localhost:4000. Inspect the cache to make sure that the specified files are cached when the service worker is installed. Take the app offline and refreshthe page. The app should load normally!

OUTPUT:-



Conclusion: hence,we have implemented service-workers in the PWA application

EXPERIMENT NO: 12

Aim:-To deploy an Ecommerce PWA using SSL enabled static hosting solution.

Theory:-

Importance of SSL Enabled Static Hosting:

SSL (Secure Sockets Layer) encryption ensures that data transmitted between the user's device and the server is secure and cannot be intercepted by malicious third parties. Hosting a website on a static hosting solution with SSL enabled provides several benefits, including:

Security: SSL encryption protects sensitive data such as login credentials, payment information, and personal details from being compromised during transmission.

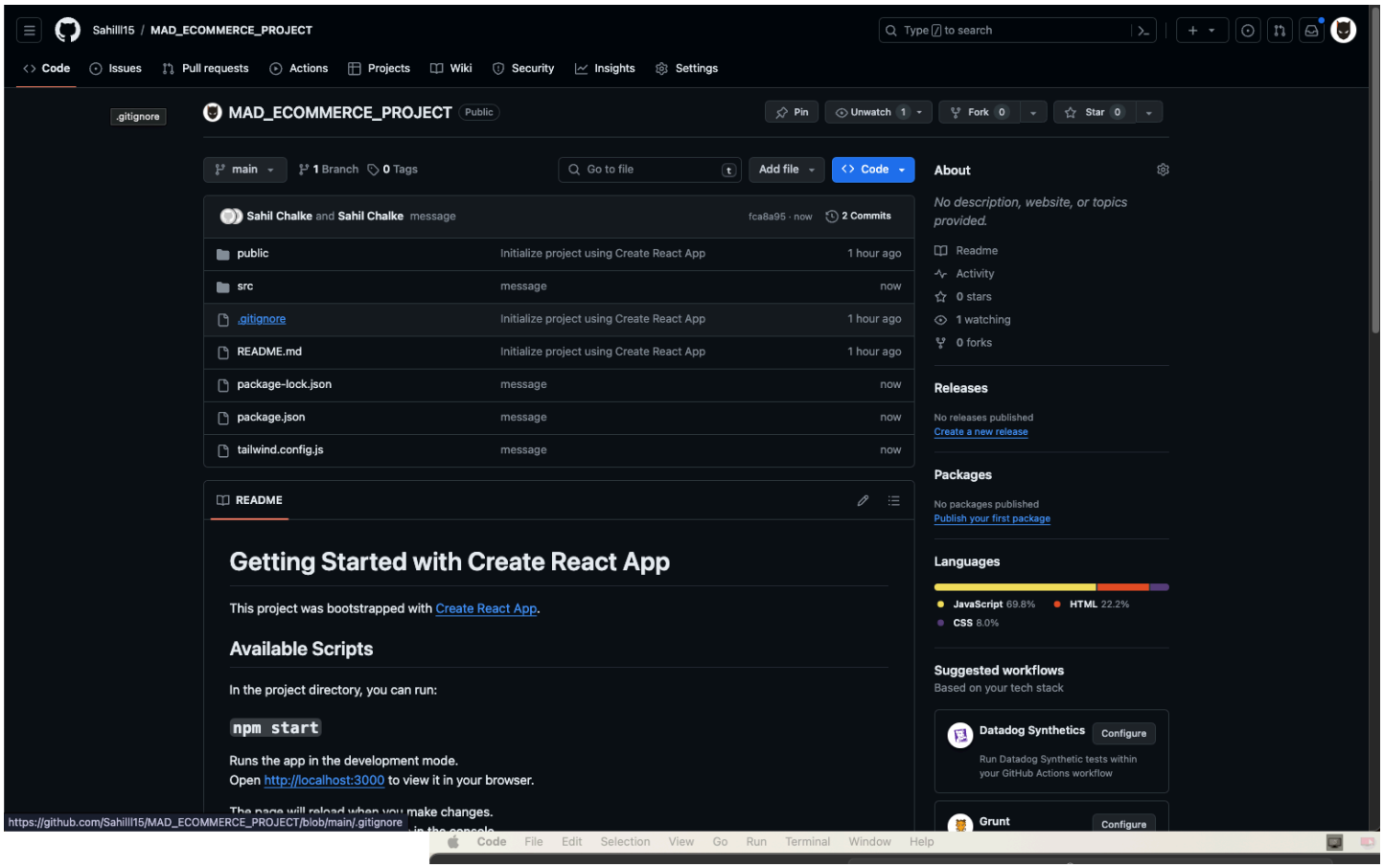
Trustworthiness: Websites with SSL certificates are considered more trustworthy by users and search engines. They display a padlock icon in the browser's address bar, indicating that the connection is secure.

SEO Benefits: Search engines like Google prioritize websites with SSL certificates in their search results, leading to better visibility and higher rankings.

Improved Performance: Static hosting solutions typically offer faster load times and better performance compared to traditional hosting environments, resulting in a smoother user experience.

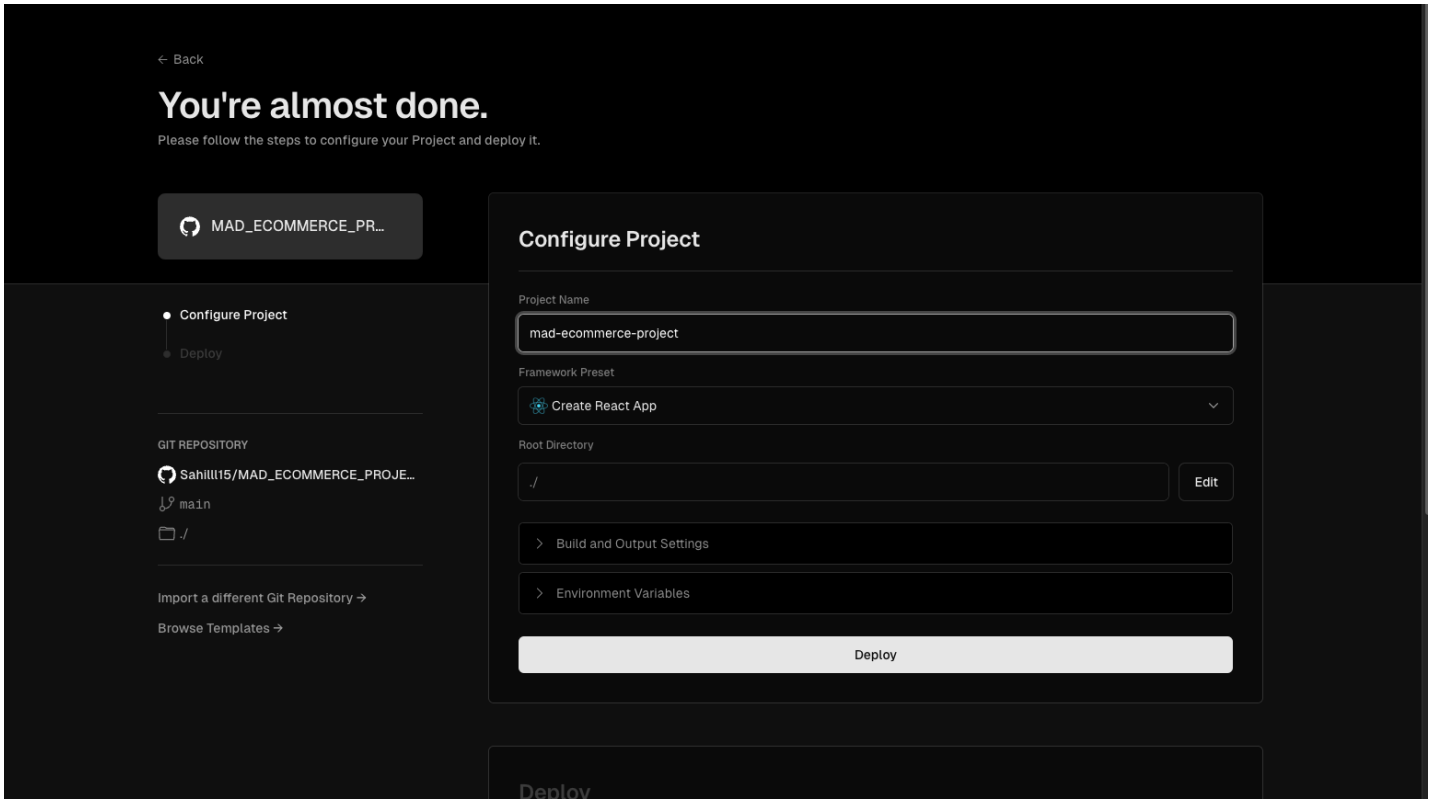
Steps:

step 1)Add the project to the github

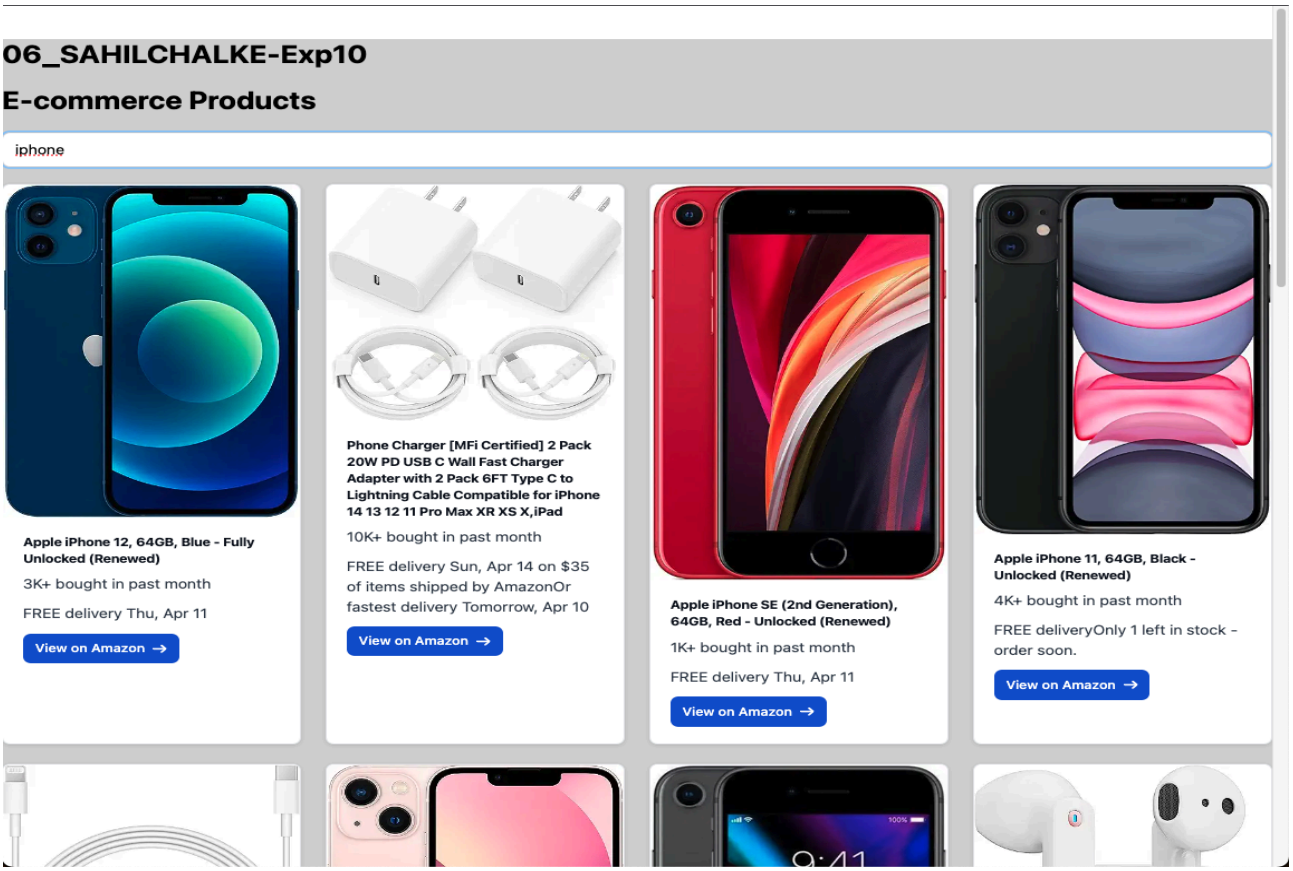


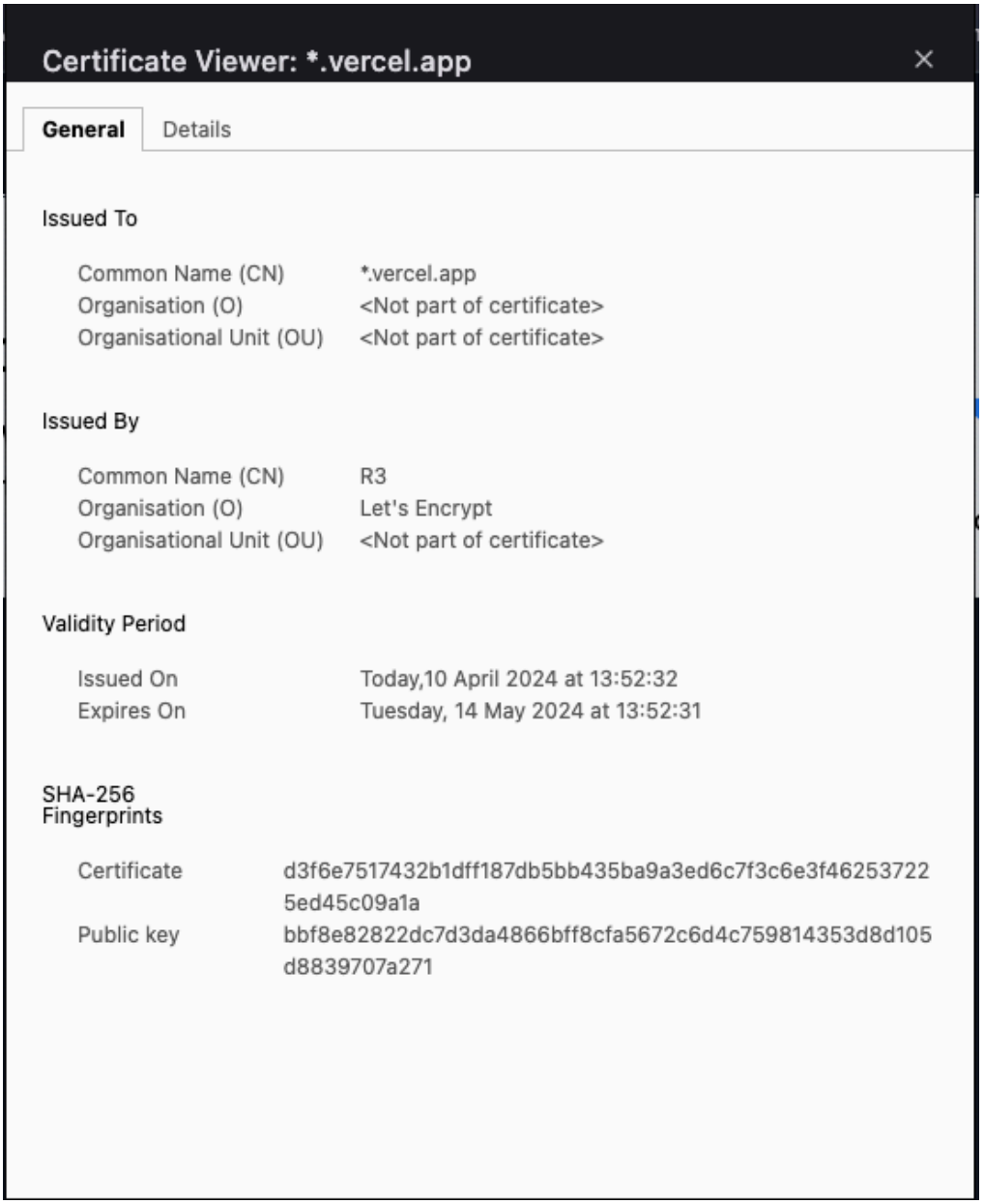
step 2) go to vercel and link the repo and deploy

vercel provides us with ssl enabled deployment for static as well as dynamic websites



step 3)**deployed project**





conclusion:-hence we have deployed are PWA application to an ssl enabled hosting service

MAD MINI PROJECT REPORT

Problem statement:-APP for C3 committee. All the programs will be advertised there. It will be a forum where students can ask technical queries.

Features:-

1)**Forum**:-

- This feature allows users of the application to participate in discussions by creating posts with titles and descriptions.
- The forum functionality can be implemented using Firebase, a backend service provided by Google. Firebase's Realtime Database or Cloud Firestore can be utilized to store and retrieve forum posts.
- Users can create new threads by providing a title and description, and they can also comment on existing threads to engage in discussions.
- Firebase Authentication can be integrated to ensure that only authenticated users can create new posts or comment on existing ones.

2)**Authentication**:-

- Authentication is a crucial aspect of any application to ensure that users' data is secure and that only authorized individuals can access certain features.
- Firebase Authentication provides ready-to-use authentication services, allowing users to sign up, sign in, and manage their accounts securely.
- Users can register for an account using their email address and password, or they can sign in using their Google or other social media accounts.
- Once authenticated, users can access the forum, chat, and other features of the application.

3)**Chat**:

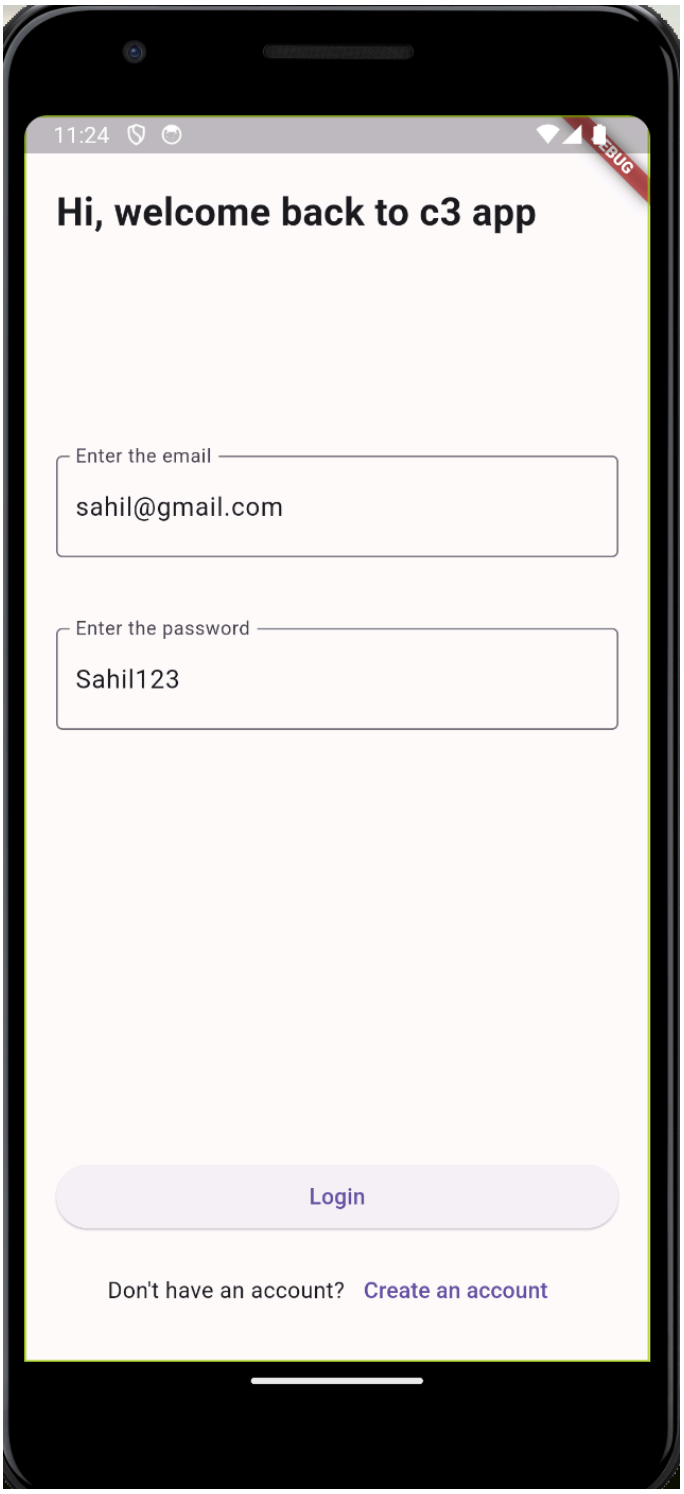
- The chat feature enables real-time communication between users within the application.
- Firebase can be used to implement real-time messaging functionality, either through Firebase Realtime Database or Firebase Cloud Firestore.
- Users can send text messages, emojis, images, and other media in their conversations.
- Chat rooms or channels can be created to organize discussions based on different topics or groups.
- Firebase Authentication ensures that users can only participate in chats if they are authenticated and authorized to do so.

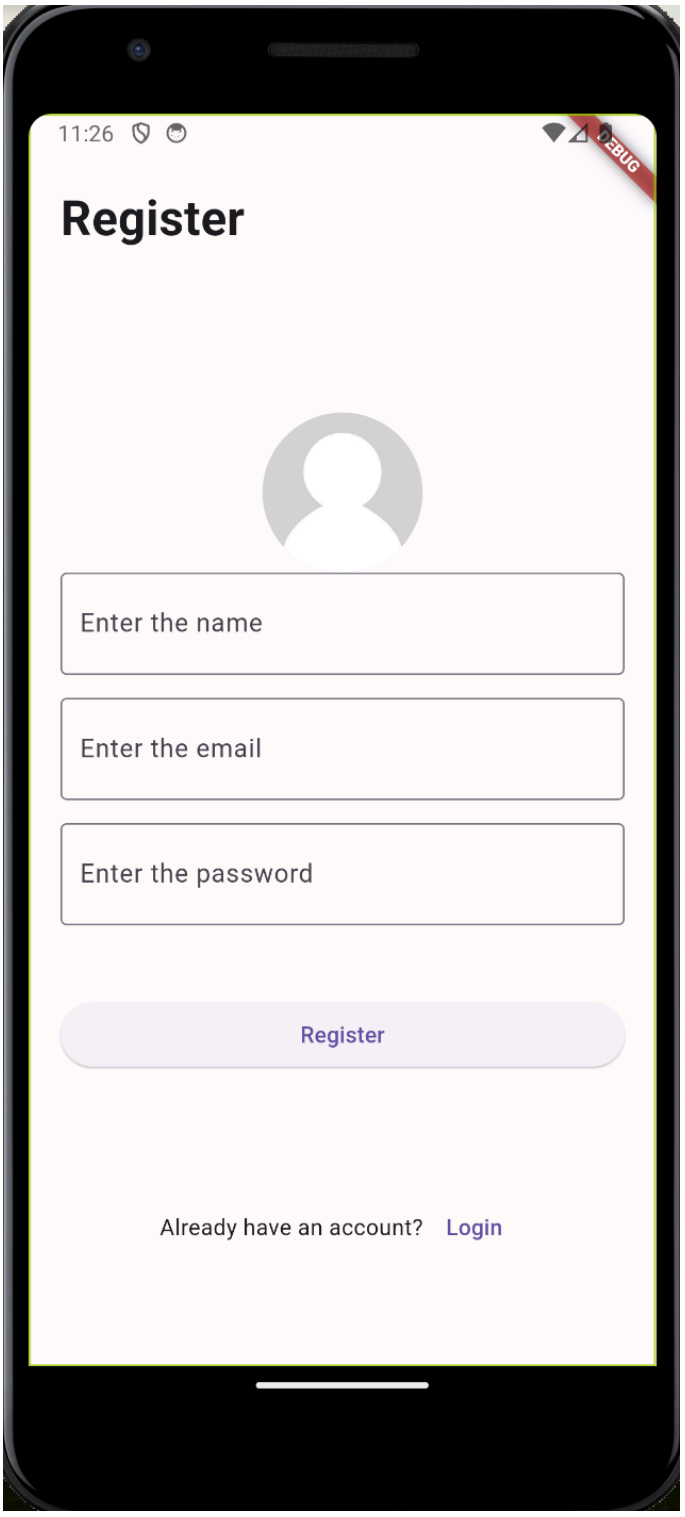
4)**Profile** :-

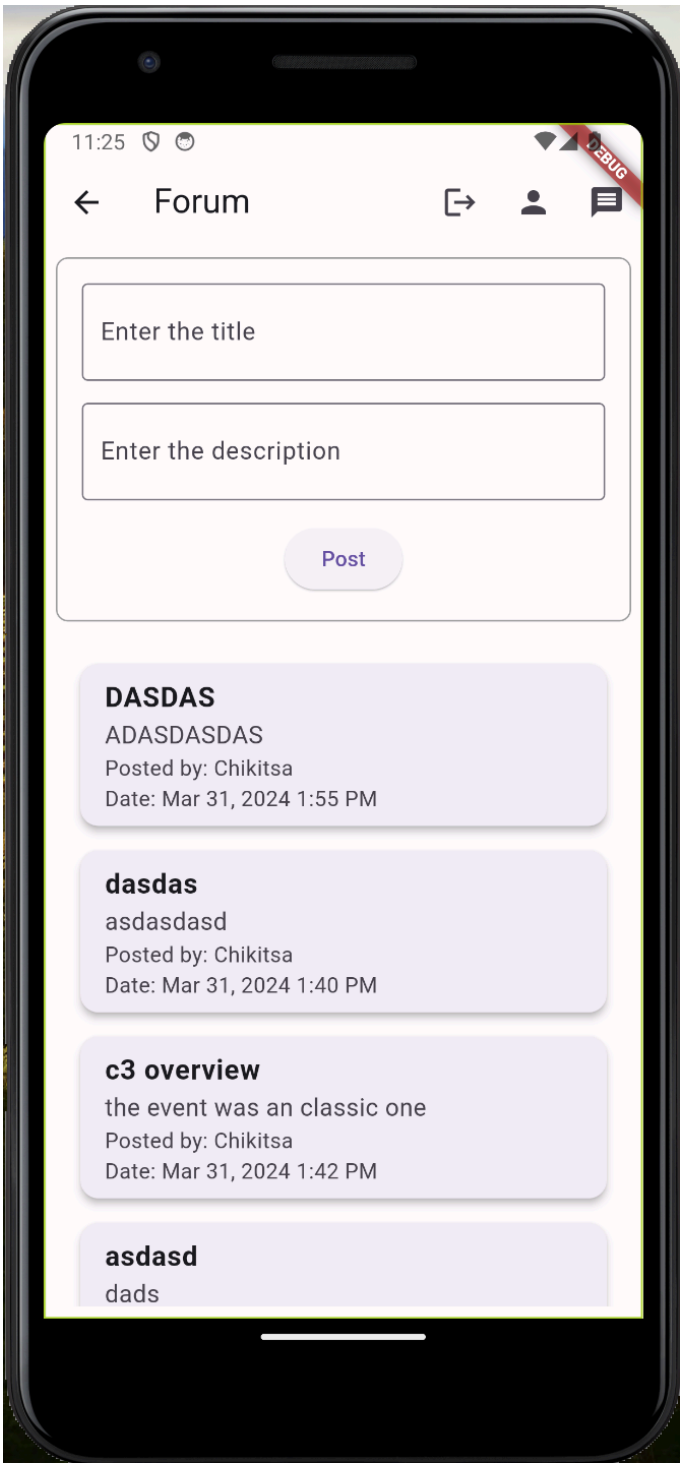
- The profile feature provides users with a dedicated space to view and manage their personal information within the application.
- Users can update their profile picture, edit their username, bio, and other details.
- The profile page may also display additional information such as user activity, posts/comments made by the user, and any badges or achievements earned.
- Firebase can be used to store user profile data securely, and Firebase Authentication ensures that users can only access and update their own profiles.

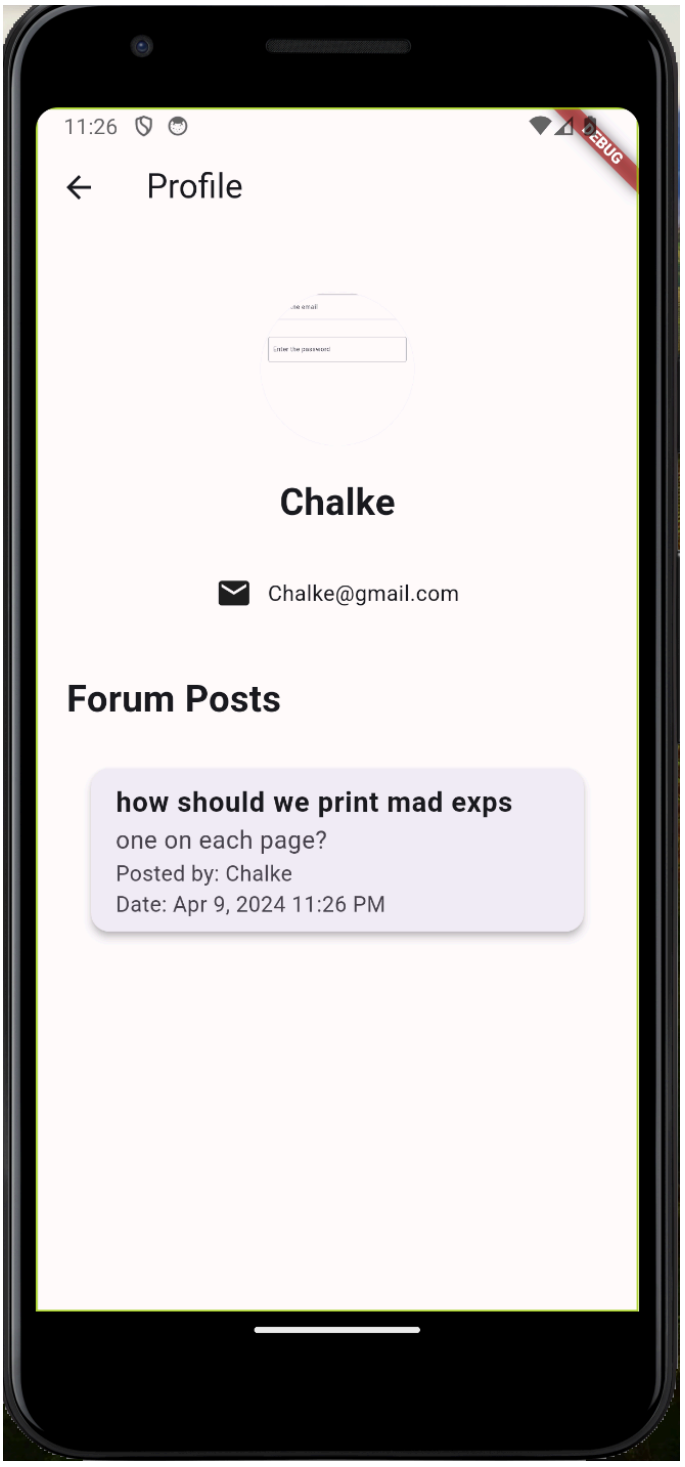
Overall, these features combine to create a comprehensive educational application where students can engage in discussions, communicate with each other, and manage their profiles securely. Firebase provides a robust backend infrastructure to support these features effectively.

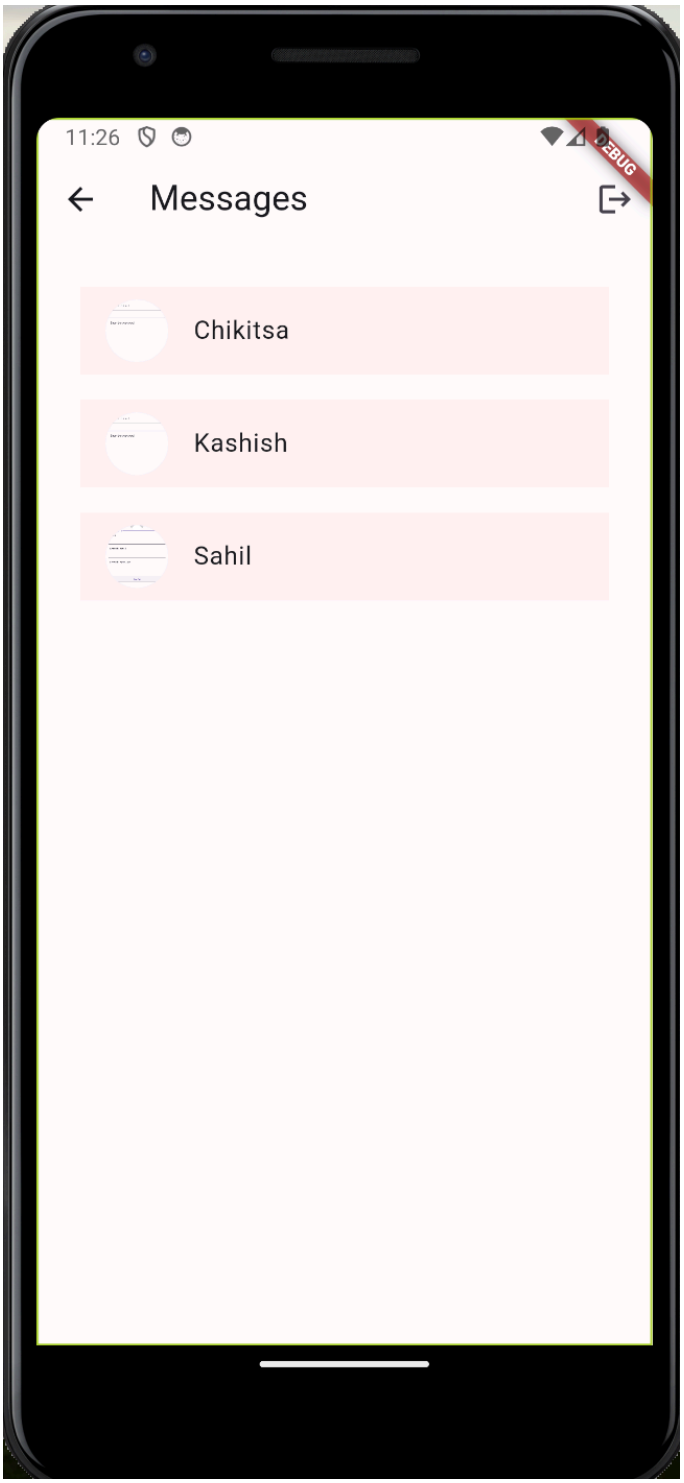
OUTPUT:-

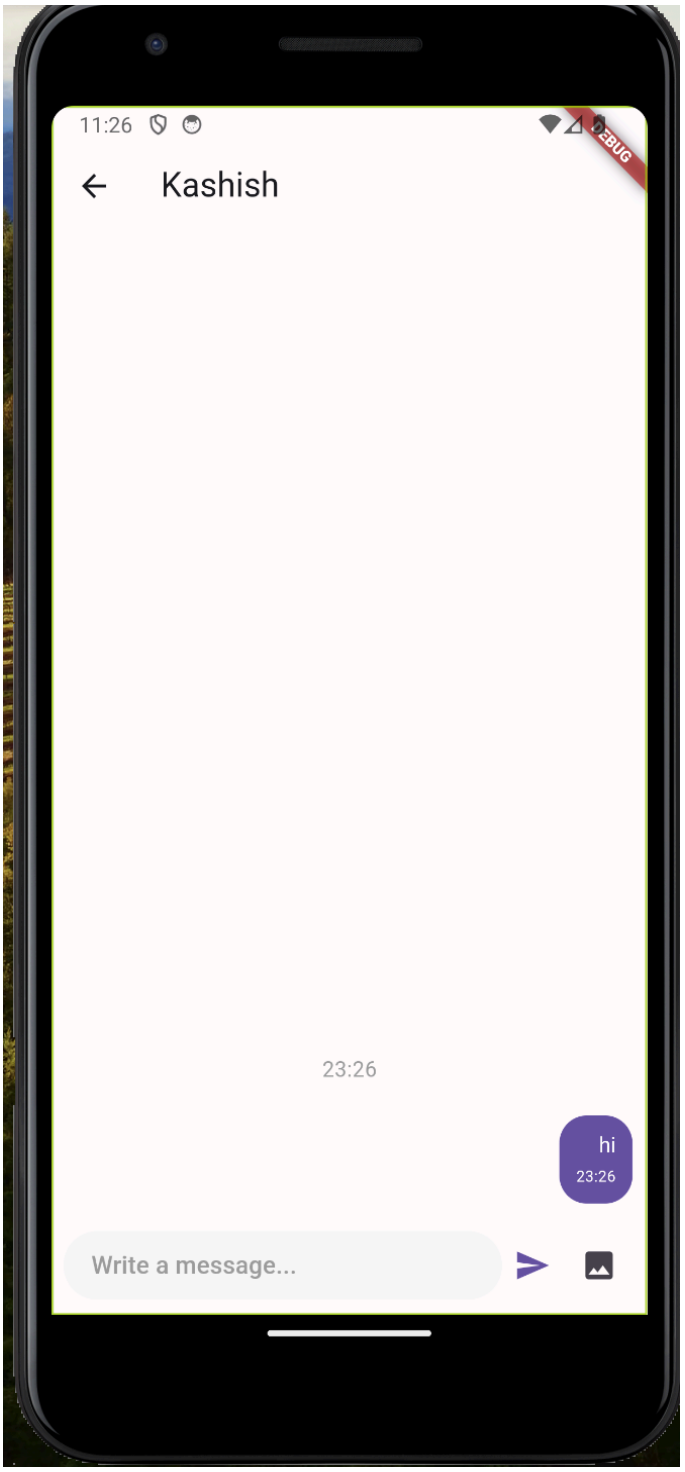

11:24
DEBUG
Hi, welcome back to c3 app
Enter the email
sahil@gmail.com
Enter the password
Sahil123
Login
Don't have an account? Create an account


11:26
DEBUG
Register
Enter the name
Enter the email
Enter the password
Register
Already have an account? Login


11:25
DEBUG
Forum
Enter the title
Enter the description
Post
DASDAS
ADASDASDAS
Posted by: Chikitsa
Date: Mar 31, 2024 1:55 PM
dasdas
asdasdasd
Posted by: Chikitsa
Date: Mar 31, 2024 1:40 PM
c3 overview
the event was an classic one
Posted by: Chikitsa
Date: Mar 31, 2024 1:42 PM
asdasd
dads


11:26
DEBUG
Profile
Chalke
Chalke@gmail.com
Forum Posts
how should we print mad exps
one on each page?
Posted by: Chalke
Date: Apr 9, 2024 11:26 PM


11:26
DEBUG
Messages
Chikitsa
Kashish
Sahil


11:26
DEBUG
Kashish
23:26
hi
23:26
Write a message...