**ASSIGNMENT 01: - Date:**

Develop a Flutter app using Dart programming that allows users to convert temperatures between Celsius, Fahrenheit, and Kelvin. The app should include an input field for the temperature value, drop-down menus to select the input and output units, and a button to perform the conversion. Display the converted temperature result within the app's interface.

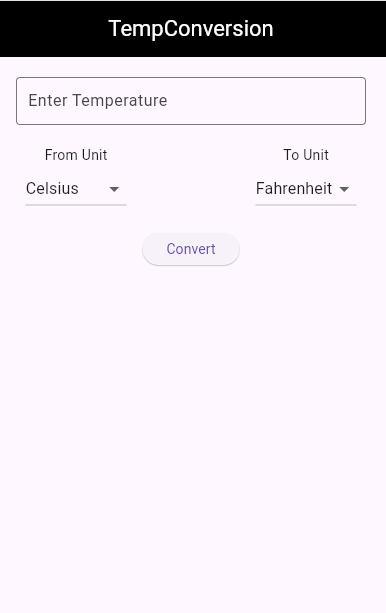
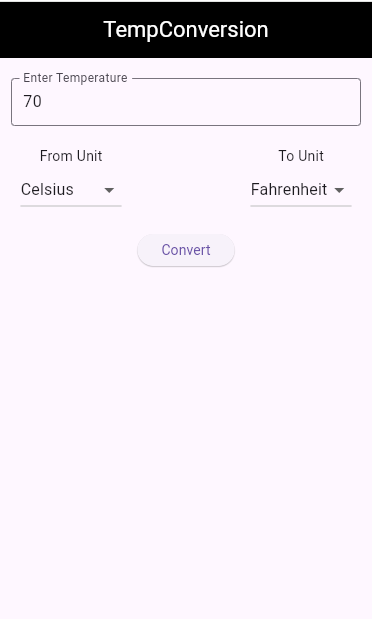
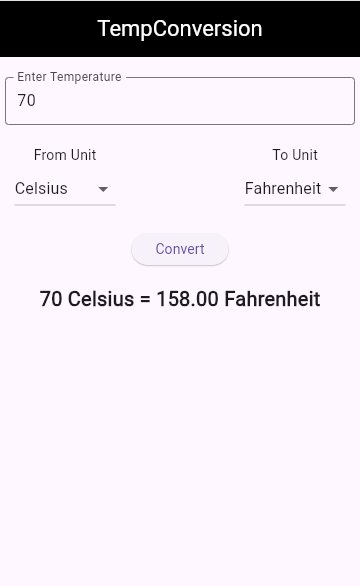
### **Design Procedure:**

1. **Project Setup**:
   * A Flutter project was created with the main.dart file.
   * Necessary packages like material.dart were imported.
2. **Main Functionality**:
   * The main() function runs the app using runApp().
   * A StatelessWidget class MyApp is created to build the MaterialApp and remove the debug banner.
3. **UI Design**:
   * A StatefulWidget TempConversion is created to manage dynamic UI changes (like showing the result).
   * The screen includes:
     + A **TextField** for entering the temperature value.
     + Two **DropdownButton** widgets to select the "From Unit" and "To Unit".
     + A **Convert Button** to trigger the conversion.
     + A **Text Widget** to display the result.
4. **Temperature Conversion Logic**:
   * A function \_conversion() is defined.
   * This function reads the input temperature, checks selected units, and applies appropriate formulas:
     + Celsius ↔ Fahrenheit ↔ Kelvin conversions handled.
   * Result is displayed after rounding to two decimal places.
5. **User Interaction**:
   * When the "Convert" button is pressed:
     + The app reads the input.
     + Converts the value according to selected units.
     + Displays the formatted result on the screen.
6. **Styling and Layout**:
   * Padding and SizedBox widgets are used for proper spacing.
   * AppBar is designed with black background and white text for a neat look.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: TempConversion(),  
 );  
 }  
}  
class TempConversion extends StatefulWidget {  
 const TempConversion({super.key});  
  
 @override  
 State<TempConversion> createState() => \_TempConversionState();  
}  
class \_TempConversionState extends State<TempConversion> {  
 final TextEditingController \_controller = TextEditingController();  
 String \_fromUnit = 'Celsius';  
 String \_toUnit = 'Fahrenheit';  
 String \_result = '';  
  
 final List<String> \_units = ['Celsius', 'Fahrenheit', 'Kelvin'];  
  
 *// Conversion Logic* void \_conversion() {  
 double input = double.*tryParse*(\_controller.text) ?? 0;  
 double output = 0;  
  
 if (\_fromUnit == \_toUnit) {  
 output = input;  
 } else if (\_fromUnit == 'Celsius') {  
 if (\_toUnit == 'Fahrenheit') {  
 output = input \* 9 / 5 + 32;  
 } else if (\_toUnit == 'Kelvin') {  
 output = input + 273.15;  
 }  
 } else if (\_fromUnit == 'Fahrenheit') {  
 if (\_toUnit == 'Celsius') {  
 output = (input - 32) \* 5 / 9;  
 } else if (\_toUnit == 'Kelvin') {  
 output = (input - 32) \* 5 / 9 + 273.15;  
 }  
 } else if (\_fromUnit == 'Kelvin') {  
 if (\_toUnit == 'Celsius') {  
 output = input - 273.15;  
 } else if (\_toUnit == 'Fahrenheit') {  
 output = (input - 273.15) \* 9 / 5 + 32;  
 }  
 }  
 setState(() {  
 \_result = '$input $\_fromUnit = ${output.toStringAsFixed(2)} $\_toUnit';  
 });  
 }  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('TempConversion'),  
 centerTitle: true,  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(20.0),  
 child: Column(  
 children: <Widget>[  
 SizedBox(  
 width: 350,  
 child: TextField(  
 controller: \_controller,  
 keyboardType: TextInputType.*number*,  
 decoration: InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Enter Temperature'),  
 ),  
 ),  
 const SizedBox(height: 20),  
 Row(  
 children: [  
 Expanded(  
 child: Column(  
 children: [  
 const Text('From Unit'),  
 DropdownButton<String>(  
 value: \_fromUnit,  
 items: \_units.map((String value) {  
 return DropdownMenuItem(  
 value: value,  
 child: Text(value),  
 );  
 }).toList(),  
 onChanged: (String? newValue) {  
 setState(() {  
 \_fromUnit = newValue!;  
 });  
 },  
 ),  
 ],  
 ),  
 ),  
 Expanded(  
 child: Column(  
 children: [  
 const Text('To Unit'),  
 DropdownButton<String>(  
 value: \_toUnit,  
 items: \_units.map((String value) {  
 return DropdownMenuItem(  
 value: value,  
 child: Text(value),  
 );  
 }).toList(),  
 onChanged: (String? newValue) {  
 setState(() {  
 \_toUnit = newValue!;  
 });  
 },  
 ),  
 ],  
 ),  
 ),  
 ],  
 ),  
 const SizedBox(height: 20),  
 ElevatedButton(  
 onPressed: \_conversion,  
 child: const Text('Convert'),  
 ),  
 const SizedBox(height: 20),  
 Text(  
 \_result,  
 style: const TextStyle(fontSize: 20, fontWeight: FontWeight.*bold*),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

In this project, a simple yet functional **Temperature Conversion App** was successfully developed using Flutter.  
It takes a temperature input, allows the user to select the source and destination units (Celsius, Fahrenheit, or Kelvin), and displays the correctly converted result.  
The app uses basic Flutter widgets like TextField, DropdownButton, ElevatedButton, and Text, demonstrating the power of Flutter for building interactive and dynamic user interfaces.  
The project helped understand important concepts like **State Management**, **User Input Handling**, **Dropdown Menus**, and **Mathematical Logic Implementation** in Flutter.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 02:- Date:**

Develop a Flutter app using Dart programming to calculate the Body Mass Index (BMI) based on user input for weight and height. The app should include input fields for users to enter their weight (in kilograms) and height (in centmetres), a button to trigger the BMI calculaton, and a display area for the BMI result along with its classificaton (e.g., Underweight, Normal weight, Overweight, Obese). Ensure that the app runs in Android Studio and provides a user-friendly interface. If BMI < 18.5 show underweight, BMI

18.5<=24.9 show normal weight, and BMI 24.9<= 29.9 show overweight, else obese.

### **Design Procedure**

1. **App Initialization**:  
   The application starts from the main() function, where runApp() launches the root widget MyApp. This MyApp is a StatelessWidget that returns a MaterialApp.
2. **Theme and Title Setup**:  
   In MaterialApp, the title is set as "Assignment Two", and a color theme is created using ColorScheme.fromSeed(). The debug banner is disabled.
3. **Home Page Widget**:  
   The home screen is handled by a StatefulWidget named MyHomePage, which receives a title as a parameter. All logic and UI interactions are managed in its state class \_MyHomePageState.
4. **Input Fields**:  
   Two TextEditingControllers are used:
   * t1\_w for entering **Weight in Kg**.
   * t2\_h for entering **Height in cm**.  
     These inputs are collected using TextField widgets.
5. **BMI Calculation Logic**:  
   The function \_calculateBMI() performs the BMI calculation using the formula:

BMI=Weight(Height×0.01)2BMI = \frac{Weight}{(Height \times 0.01)^2}

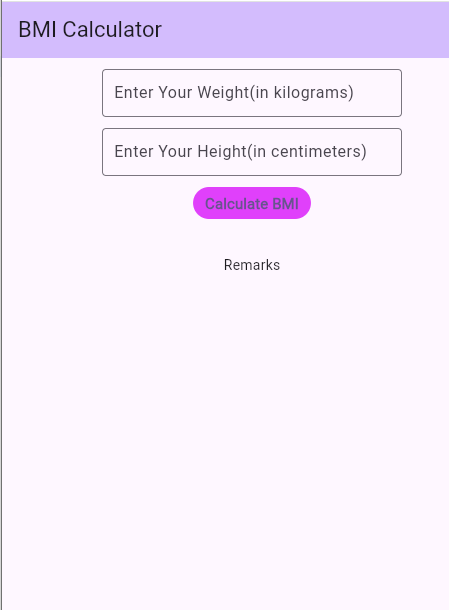
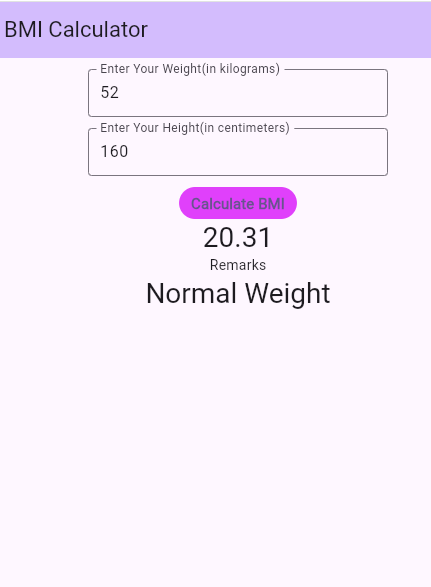
* + The result is stored in the variable res with 2 decimal places.
  + Based on the BMI value, the app assigns a remark to the stat variable:
    - BMI < 18.5 → "Under Weight"
    - 18.5 to 24.9 → "Normal Weight"
    - 25 to 29.9 → "Over Weight"
    - 30 and above → "Obese"

1. **User Interface Layout**:  
   The layout is built using a Scaffold with an AppBar and a Center widget containing a vertical Column.
   * Two input fields collect user data.
   * A TextButton triggers the BMI calculation.
   * The result (res) and the remarks (stat) are displayed using Text widgets.

**Source Code**:

import 'package:flutter/material.dart';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 *// This widget is the root of your application.* @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Assignment Two',  
 debugShowCheckedModeBanner: false,  
 theme: ThemeData(  
 colorScheme: ColorScheme.fromSeed(seedColor: Colors.*deepPurple*),  
 useMaterial3: true,  
 ),  
 home: const MyHomePage(title: 'BMI Calculator'),  
 );  
 }  
}  
  
class MyHomePage extends StatefulWidget {  
 const MyHomePage({super.key, required this.title});  
 final String title;  
  
 @override  
 State<MyHomePage> createState() => \_MyHomePageState();  
}  
  
class \_MyHomePageState extends State<MyHomePage> {  
 TextEditingController t1\_w = TextEditingController();  
 TextEditingController t2\_h = TextEditingController();  
 String res = '';  
 String stat = '';  
 void \_calculateBMI() {  
 setState(() {  
 double weight = double.*parse*(t1\_w.text);  
 double height = double.*parse*(t2\_h.text);  
 double bmi = weight / (height \* 0.01 \* height \* 0.01);  
 res = bmi.toStringAsFixed(2);  
 if (bmi < 18.5) {  
 stat = 'Under Weight';  
 } else if (bmi <= 24.9) {  
 stat = 'Normal Weight';  
 } else if (bmi <= 29.9) {  
 stat = 'Over Weight';  
 } else {  
 stat = 'Obese';  
 }  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 backgroundColor: Theme.*of*(context).colorScheme.inversePrimary,  
 title: Text(widget.title),  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: <Widget>[  
 SizedBox(  
 width: 300,  
 child: TextField(  
 decoration: InputDecoration(  
 border: OutlineInputBorder(),  
 label: Text("Enter Your Weight(in kilograms)")),  
 controller: t1\_w)),  
 SizedBox(height: 11,),  
 SizedBox(  
 width: 300,  
 child: TextField(  
 decoration: InputDecoration(  
 border: OutlineInputBorder(),  
 label: Text("Enter Your Height(in centimeters)")),  
 controller: t2\_h)),  
 SizedBox(height: 11,),  
 TextButton(  
 onPressed: \_calculateBMI,  
 style:  
 TextButton.*styleFrom*(backgroundColor: Colors.*purpleAccent*),  
 child: Text(  
 'Calculate BMI',  
 style: TextStyle(fontSize: 15, fontWeight: FontWeight.*w600*),  
 )),  
 Text(  
 res,  
 style: Theme.*of*(context).textTheme.headlineMedium,  
 ),  
 const Text('Remarks'),  
 Text(  
 stat,  
 style: Theme.*of*(context).textTheme.headlineMedium,  
 ),  
 ],  
 ),  
 ),  
 *// This trailing comma makes auto-formatting nicer for build methods.* );  
 }  
}

**Code Output**:

**Conclusion**

In this Flutter project, we developed a simple **BMI Calculator** application that takes the user's weight and height, calculates the BMI, and displays a health remark accordingly.  
This app uses basic Flutter components such as TextField, TextButton, StatefulWidget, and setState() to manage inputs and state updates.  
The main goal of the project was to understand user input handling, UI design, and dynamic data display using Flutter. The app is beginner-friendly, user-friendly, and performs the calculation accurately based on standard BMI rules.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 03: - Date:**

Create a Flutter app using Dart programming that calculates a person's age based on their date of birth. The app should include an input field for users to enter their date of birth in the format (yyyy-mm-dd), a button to trigger the age calculaton, and a display area to show the calculated age. Ensure that the app runs in Android Studio and provides a user-friendly interface

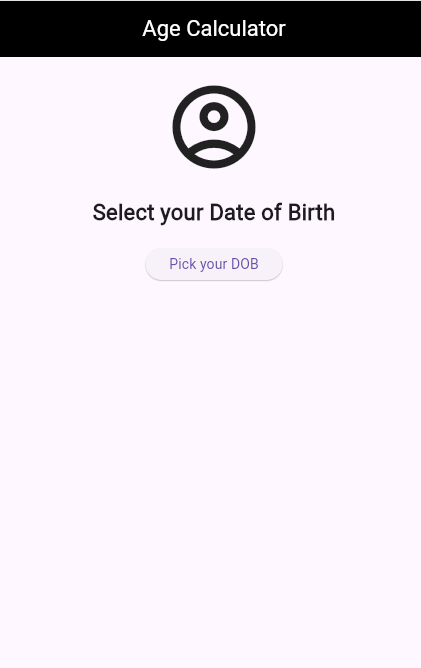
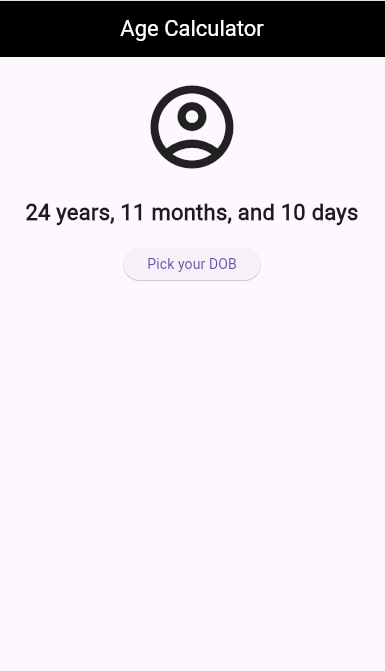
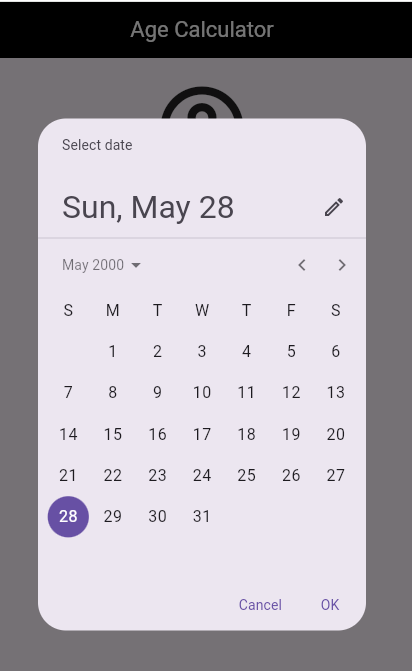
### **Design Procedure:**

1. **Import Flutter Package**:  
   First, the flutter/material.dart package is imported to access Material Design components for the app.
2. **Main Function**:  
   The main() function is the entry point of the application. It runs the app by calling runApp() with the MyApp widget.
3. **MyApp Widget**:
   * MyApp is a **StatelessWidget**.
   * It returns a MaterialApp with debugShowCheckedModeBanner set to false to remove the debug label.
   * The home property is set to the AgeCalculator widget.
4. **AgeCalculator Widget**:
   * AgeCalculator is a **StatefulWidget**.
   * It is responsible for allowing the user to pick their date of birth and calculate their age.
5. **State Management (\_AgeCalculatorState)**:
   * A String variable named myAge is created to store and display the age or a default message ("Select your Date of Birth").
   * The pickDob() function opens a **DatePicker** where the user can select their date of birth.
   * After the user selects a date, the calculateAge() function is called.
6. **Age Calculation**:
   * In the calculateAge() function, the difference between the current date and the selected birth date is calculated.
   * The result is converted into **years**, **months**, and **days**.
   * The setState() function is used to update the myAge variable and refresh the UI.
7. **User Interface (UI) Design**:
   * An **AppBar** is created with a black background and white text, centered in the app.
   * An icon (account\_circle\_outlined) is displayed at the center.
   * Below the icon, the calculated age is shown.
   * An **ElevatedButton** is provided to allow users to open the DatePicker and select their DOB.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: AgeCalculator(),  
 );  
 }  
}  
class AgeCalculator extends StatefulWidget {  
 const AgeCalculator({super.key});  
 @override  
 State<AgeCalculator> createState() => \_AgeCalculatorState();  
}  
class \_AgeCalculatorState extends State<AgeCalculator> {  
 String myAge = 'Select your Date of Birth';  
 void pickDob(BuildContext context) {  
 showDatePicker(  
 context: context,  
 initialDate: DateTime.now(),  
 firstDate: DateTime(1900),  
 lastDate: DateTime.now(),  
 ).then((pickedDate) {  
 if (pickedDate != null) {  
 calculateAge(pickedDate);  
 }  
 });  
 }  
 void calculateAge(DateTime birth) {  
 DateTime now = DateTime.now();  
 Duration age = now.difference(birth);  
 int years = age.inDays ~/ 365;  
 int months = (age.inDays % 365) ~/ 30;  
 int days = ((age.inDays % 365) % 30);  
  
 setState(() {  
 myAge = '$years years, $months months, and $days days';  
 });  
 }  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.white,  
 backgroundColor: Colors.black,  
 title: const Text('Age Calculator'),  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 const SizedBox(height: 20),  
 const Icon(  
 Icons.account\_circle\_outlined,  
 size: 100,  
 ),  
 const SizedBox(height: 20),  
 Text(  
 myAge,  
 style: const TextStyle(  
 fontSize: 22,  
 fontWeight: FontWeight.bold,  
 ),  
 textAlign: TextAlign.center,  
 ),  
 const SizedBox(height: 20),  
 ElevatedButton(  
 onPressed: () => pickDob(context),  
 child: const Text('Pick your DOB'),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**In this project, a simple and user-friendly **Age Calculator App** has been created using Flutter.  
The user can select their Date of Birth, and the app automatically calculates and displays their age in years, months, and days.  
This project demonstrates the use of **DatePicker**, **state management** (setState()), and basic **Flutter UI components**.  
It serves as a good example for beginners to understand **date handling**, **state updates**, and **basic app structure** in Flutter.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 04: - Date:**

Develop a Flutter app using Dart programming that functons as a stopwatch with start, stop, and reset functionalites. The app should include a display area for the elapsed tme, buttons to start, stop, and reset the stopwatch, and should provide a user-friendly interface. Ensure that the app runs in Android Studio and updates the elapsed time in real-time while the stopwatch is running.

### **Design Procedure:**

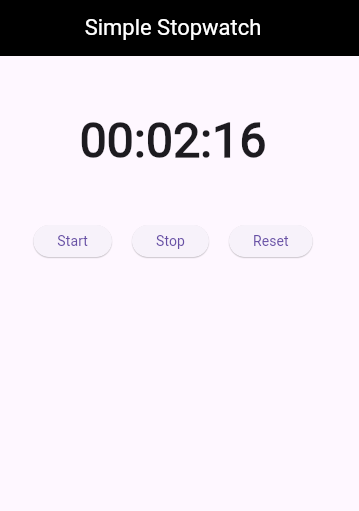
1. **App Initialization**:
   * The main() function runs the app using the MyApp widget.
   * MyApp is a StatelessWidget that returns a MaterialApp with the StopWatch widget as the home screen.
2. **State Management**:
   * StopWatch is a StatefulWidget because the time needs to update every second.
   * Inside the \_StopWatchState class, three important variables are used:
     + seconds: keeps track of total elapsed time in seconds.
     + timer: a Timer object that triggers every second.
     + isRunning: a boolean to indicate whether the stopwatch is currently running.
3. **Timer Control Functions**:
   * \_start(): starts the timer using Timer.periodic if it is not already running.
   * \_stop(): stops the timer using timer.cancel() if it is running.
   * \_reset(): stops the timer and resets the seconds to 0.
4. **Time Formatting**:
   * The timeText getter converts the total seconds into HH:MM:SS format using simple integer math and the padLeft method.
5. **User Interface**:
   * The UI is built using a Scaffold with an AppBar and a Column layout.
   * A Text widget displays the formatted time.
   * Three ElevatedButtons allow the user to Start, Stop, and Reset the stopwatch.

**Source Code**:

import 'dart:async'; import 'package:flutter/material.dart';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: StopWatch(),  
 );  
 }  
}  
  
class StopWatch extends StatefulWidget {  
 const StopWatch({super.key});  
  
 @override  
 State<StopWatch> createState() => \_StopWatchState();  
}  
  
class \_StopWatchState extends State<StopWatch> {  
 int seconds = 0; Timer? timer; bool isRunning = false; void \_start() {  
 if (!isRunning) {  
 timer = Timer.periodic(Duration(seconds: 1), (Timer t) {  
 setState(() {  
 seconds = seconds + 1;  
 });  
 });  
 setState(() {  
 isRunning = true;  
 });  
 }  
 }  
  
 void \_stop() {  
 if (isRunning) {  
 timer?.cancel(); setState(() {  
 isRunning = false;  
 });  
 }  
 }  
  
 void \_reset() {  
 timer?.cancel();

setState(() {  
 seconds = 0;  
 isRunning = false;  
 });  
 }  
  
 String get timeText {  
 int hrs = seconds ~/ 3600;  
 int mins = (seconds % 3600) ~/ 60;  
 int secs = seconds % 60;  
  
 String twoDigits(int n) => n.toString().padLeft(2, '0');  
 return "${twoDigits(hrs)}:${twoDigits(mins)}:${twoDigits(secs)}";  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 backgroundColor: Colors.*black*,  
 foregroundColor: Colors.*white*,  
 title: const Text('Simple Stopwatch'),  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 SizedBox(height: 50,),  
 Text(  
 timeText, style: const TextStyle(fontSize: 48, fontWeight: FontWeight.*bold*),  
 ),  
 const SizedBox(height: 50),  
 Row(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 ElevatedButton(  
 onPressed: \_start,  
 child: const Text('Start'),  
 ),  
 const SizedBox(width: 20),  
 ElevatedButton(  
 onPressed: \_stop,  
 child: const Text('Stop'),  
 ),  
 const SizedBox(width: 20),  
 ElevatedButton(  
 onPressed: \_reset,  
 child: const Text('Reset'),  
 ),  
 ],  
 )  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

This Flutter project demonstrates the creation of a **basic stopwatch application** that allows users to start, stop, and reset time tracking. It features a clean and minimal user interface and handles real-time updates using a Timer.

The app effectively utilizes:

* **Stateful widgets** for dynamic updates,
* **Timer class** for time control,
* and **Flutter UI widgets** for interaction and layout.

This project is ideal for beginners learning Flutter, as it introduces important concepts such as **state management**, **event handling**, and **UI design** in a simple and understandable way.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 05: - Date:**

Develop a Flutter app using Dart programming that creates a list of names for individuals eligible to vote. The app should take input from the user, including names and ages, and use toList and map to filter and display two separate lists: one for those who are eligible to vote and one for those who are not eligible. Ensure that the app runs in Android Studio and provides a clear and user-friendly interface to show both lists based on the eligibility criteria.

### **Design Procedure:**

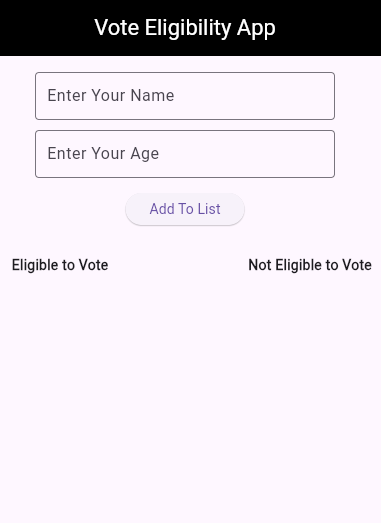
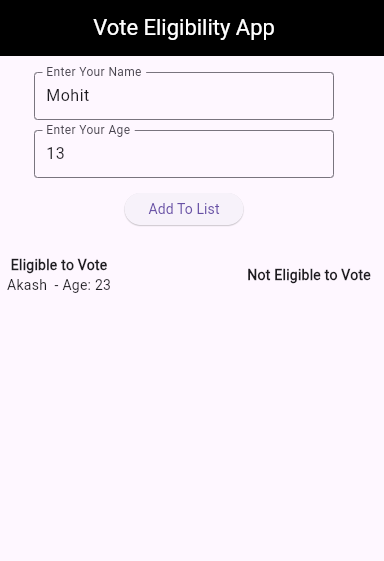
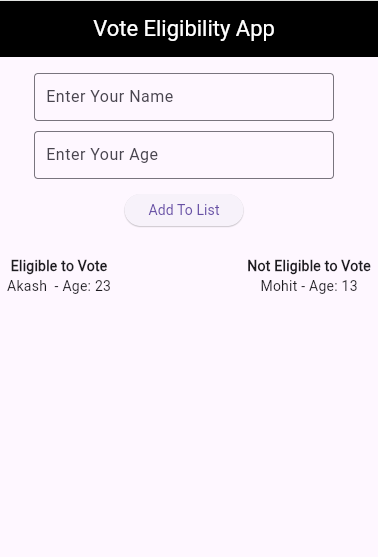
The design procedure of the Vote Eligibility App is based on the Flutter framework, with a focus on creating a user interface that allows users to input personal data and checks for their eligibility to vote based on their age. The design includes several steps and features:

1. **Creating the Flutter Application**:
   * The app is created using the Flutter framework. It has a StatefulWidget to manage dynamic changes (adding people to the list and checking eligibility).
2. **Setting up the User Interface**:
   * The app consists of two main input fields: one for entering the user’s name and another for entering the user’s age.
   * These input fields are created using the TextField widget, with the TextEditingController to manage the data input.
   * A button labeled "Add To List" is used to trigger the addition of the entered data (name and age) into the list.
3. **Data Management**:
   * A list of people is maintained using the List<Map<String, dynamic>> data structure, where each entry is a dictionary containing the name and age of the person.
   * The user’s input is validated: if either the name or age is empty, it does not proceed. The age input is parsed as an integer to check if it’s a valid number.
4. **Eligibility Check**:
   * After adding people to the list, the app categorizes them based on their age. It filters people into two categories:
     + **Eligible to Vote**: People aged 18 or above.
     + **Not Eligible to Vote**: People below the age of 18.
   * The list of eligible and non-eligible people is displayed side by side on the screen.
5. **Updating the UI**:
   * The list of eligible and non-eligible people is dynamically updated whenever the user adds a new person. This is achieved by calling setState() which rebuilds the UI to reflect the changes in the list.
6. **App Styling**:
   * The app uses basic styling such as borders for input fields, padding between widgets, and a black background for the app bar for a simple, clear design.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: VoteEligibilityApp(),  
 );  
 }  
}  
  
class VoteEligibilityApp extends StatefulWidget {  
 const VoteEligibilityApp({super.key});  
  
 @override  
 State<VoteEligibilityApp> createState() => \_VoteEligibilityAppState();  
}  
  
class \_VoteEligibilityAppState extends State<VoteEligibilityApp> {  
TextEditingController nameController = TextEditingController();  
 TextEditingController ageController = TextEditingController();  
List<Map<String, dynamic>> peopleList = [];  
void \_addPerson() {  
 String name = nameController.text;  
 String ageText = ageController.text;  
if (name.isEmpty || ageText.isEmpty) {  
 return;  
 }  
  
 int age = int.*tryParse*(ageText) ?? 0;  
  
 setState(() {  
 peopleList.add({"name": name, "age": age});  
  
nameController.clear();  
 ageController.clear();  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
List<Map<String, dynamic>> eligible = peopleList  
 .where((person) => person["age"] >= 18)  
 .toList();  
  
 List<Map<String, dynamic>> notEligible = peopleList  
 .where((person) => person["age"] < 18)  
 .toList();  
  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Vote Eligibility App'),  
 centerTitle: true,  
 ),  
 body: SingleChildScrollView(  
 child: Column(  
 children: [  
 SizedBox(height: 16),  
 SizedBox(  
 width: 300,  
 child: TextField(  
 controller: nameController,  
 decoration: InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Enter Your Name',  
 ),  
 ),  
 ),  
 SizedBox(height: 10),  
 SizedBox(  
 width: 300,  
 child: TextField(  
 controller: ageController,  
 keyboardType: TextInputType.*number*,  
 decoration: InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Enter Your Age',  
 ),  
 ),  
 ),  
 SizedBox(height: 15),  
 ElevatedButton(onPressed: \_addPerson, child: Text('Add To List')),  
 SizedBox(height: 30),  
 Row(  
 children: [  
 Expanded(  
 child: Column(  
 children: [  
 Text(  
 'Eligible to Vote',  
 style: TextStyle(fontWeight: FontWeight.*bold*),  
 ),  
 ...eligible.map((person) {  
 return Text("${person['name']} - Age: ${person['age']}");  
 }).toList(),  
 ],  
 ),  
 ),  
 Expanded(  
 child: Column(  
 children: [  
 Text(  
 'Not Eligible to Vote',  
 style: TextStyle(fontWeight: FontWeight.*bold*),  
 ),  
 ...notEligible.map((person) {  
 return Text("${person['name']} - Age: ${person['age']}");  
 }).toList(),  
 ],  
 ),  
 ),  
 ],  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

The Vote Eligibility App effectively manages user input for age and name to check voting eligibility. The use of Flutter’s StatefulWidget allows real-time updates to the interface as users input data. This app helps categorize people into two groups based on their age and displays them side by side, making it easy to understand who is eligible to vote and who is not. The app's simple and user-friendly design ensures that it is easy for users to add their information and immediately see the results. Overall, this app is a straightforward implementation for checking vote eligibility based on age, utilizing Flutter’s capabilities for UI building and data handling.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 06: - Date:**

Develop a Flutter app using Dart programming to implement a countdown timer. The app should allow users to set a specific time, start the countdown, and display the remaining time. Include functonalites to start, stop, and reset the timer. Ensure that the app runs seamlessly in Android Studio and provides a user-friendly interface for users to interact with the countdown timer.

### **Design Procedure:**

1. **Project Initialization**:
   * A new Flutter project was created using the Flutter framework.
   * Necessary packages such as material.dart were imported for UI design.
2. **Main App Structure**:
   * The main function calls runApp() and initializes the MyApp widget.
   * MyApp is a stateless widget that returns a MaterialApp with the home screen set to CountdownTimer.
3. **Creating the CountdownTimer Widget**:
   * A StatefulWidget named CountdownTimer was created to handle dynamic changes in the timer.
   * Its state is managed by \_CountdownTimerState.
4. **Timer Logic Implementation**:
   * An integer variable \_timeLeft was initialized with a value of 10 seconds.
   * A Timer object \_timer was used to run the countdown periodically every second.
   * Boolean \_isRunning is used to avoid multiple timer instances from running simultaneously.
5. **Control Functions**:
   * \_startTimer() begins the countdown if the timer is not already running.
   * \_stopTimer() cancels the current timer and sets the running flag to false.
   * \_resetTimer() stops the timer and resets \_timeLeft back to 10 seconds.
6. **User Interface Design**:
   * A Scaffold widget was used with an AppBar titled "Countdown Timer".
   * The countdown is displayed inside a circular container styled using BoxDecoration.
   * A row of three buttons — Start, Stop, and Reset — is added for user interaction.
   * Proper spacing and alignment were applied using SizedBox, EdgeInsets, and MainAxisAlignment.

**Source Code**:

import 'dart:async';  
import 'package:flutter/material.dart';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: CountdownTimer(),  
 );  
 }  
}  
  
class CountdownTimer extends StatefulWidget {  
 const CountdownTimer({super.key});  
  
 @override  
 State<CountdownTimer> createState() => \_CountdownTimerState();  
}  
  
class \_CountdownTimerState extends State<CountdownTimer> {  
 int \_timeLeft = 10; *// Countdown starts from 10 seconds* Timer? \_timer; *// Timer variable* bool \_isRunning = false; *// To check if the timer is running  
  
 // Start the timer* void \_startTimer() {  
 if (\_isRunning) return; *// If already running, do nothing* \_isRunning = true;  
 \_timer = Timer.periodic(Duration(seconds: 1), (timer) {  
 setState(() {  
 if (\_timeLeft > 0) {  
 \_timeLeft--; *// Reduce time* } else {  
 \_timer?.cancel(); *// Stop when time reaches 0* \_isRunning = false;  
 }  
 });  
 });  
 }  
  
 *// Stop the timer* void \_stopTimer() {  
 \_timer?.cancel();  
 \_isRunning = false;  
 }  
  
 *// Reset the timer* void \_resetTimer() {  
 \_timer?.cancel();  
 \_isRunning = false;  
 setState(() {  
 \_timeLeft = 10;  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text('Countdown Timer'),  
 backgroundColor: Colors.*black*,  
 foregroundColor: Colors.*white*,  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 Container(  
 decoration: BoxDecoration(  
 shape: BoxShape.circle,  
 border: Border.all(width: 4)  
 ),  
 margin: EdgeInsets.all(24),width: 200,height: 200,  
 child: Center(  
 child:Text(  
 'Time Left: $\_timeLeft',  
 style: TextStyle(fontSize: 30),  
 ),  
 ),  
 ),  
  
 SizedBox(height: 20),  
 Row(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 ElevatedButton(onPressed: \_startTimer, child: Text('Start')),  
 SizedBox(width: 10),  
 ElevatedButton(onPressed: \_stopTimer, child: Text('Stop')),  
 SizedBox(width: 10),  
 ElevatedButton(onPressed: \_resetTimer, child: Text('Reset')),  
 ],  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**The countdown timer app was successfully developed using Flutter. It demonstrates the use of StatefulWidget, Dart’s Timer class, and Flutter's UI components such as Scaffold, AppBar, Container, and ElevatedButton. The app allows users to start, stop, and reset a 10-second countdown, with real-time updates on the screen. This project serves as a practical implementation of timer control logic and state management in Flutter, providing a solid foundation for building more advanced time-based applications.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 07: - Date:**

Develop a Flutter app using Dart programming that converts one currency to another based on user input and predefined exchange rates. The app should include input fields for users to enter the amount, drop-down menus to select the input and output currencies, and a buyton to perform the conversion. Ensure that the app displays the converted amount and provides a user-friendly interface. The app should run seamlessly in Android Studio.

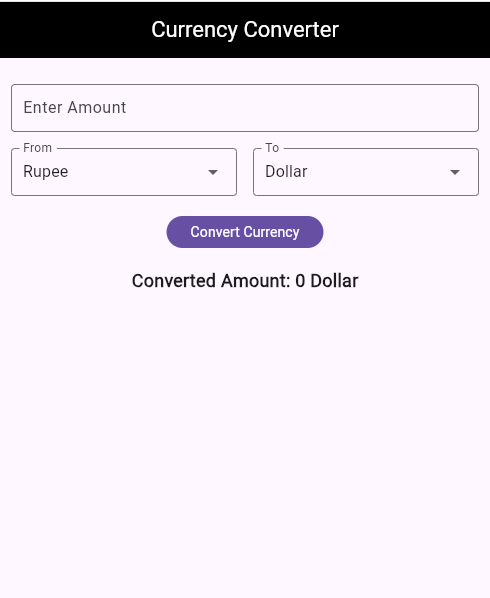
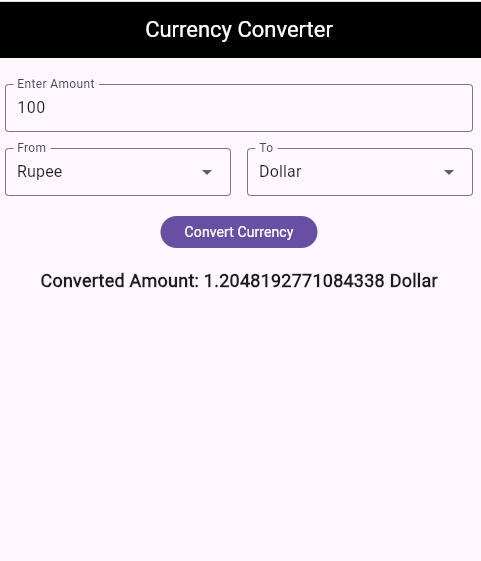
### **Design Procedure:**

1. **Objective**:  
   To build a simple Flutter application that converts currency from one type to another using predefined exchange rates.
2. **Project Setup**:
   * A new Flutter project was created.
   * The main entry point is defined in the main() function which runs the MyApp widget.
3. **UI Structure**:
   * The MaterialApp widget is used as the root of the app with debugShowCheckedModeBanner set to false.
   * The home screen is set to ConCurrency, a StatefulWidget that maintains the state of the currency conversion process.
4. **Widgets Used**:
   * TextField: Allows users to input the amount to convert.
   * DropdownButtonFormField: Lets users select the "From" and "To" currencies from a list.
   * FilledButton: Triggers the currency conversion logic.
   * Text: Displays the converted amount.
   * AppBar, Padding, SizedBox, and Column are used for layout and styling.
5. **Conversion Logic**:
   * A map \_rates contains hardcoded exchange rates relative to 1 USD.
   * When the user presses the "Convert Currency" button, the convert() method:
     + Reads the input amount.
     + Converts it to USD.
     + Then converts the USD amount to the desired target currency.
     + Updates the UI using setState() to display the result.
6. **State Management**:  
   The setState() method is used to update the selected currencies and the converted amount when user input changes.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: ConCurrency(),  
 );  
 }  
}  
  
class ConCurrency extends StatefulWidget {  
 const ConCurrency({super.key});  
  
 @override  
 State<ConCurrency> createState() => \_ConCurrencyState();  
}  
  
class \_ConCurrencyState extends State<ConCurrency> {  
 final TextEditingController \_amountController = TextEditingController();  
  
 String \_fromCurrency = 'Rupee';  
 String \_toCurrency = 'Dollar';  
 double \_convertedAmount = 0.0;  
  
 *// Exchange rates relative to 1 USD* final Map<String, double> \_rates = {  
 'Rupee': 83.0,  
 'Dollar': 1.0,  
 'Euro': 0.92,  
 'Yen': 155.0,  
 };  
  
 void convert() {  
 double input = double.*tryParse*(\_amountController.text) ?? 0.0;  
  
 double fromRate = \_rates[\_fromCurrency]!;  
 double toRate = \_rates[\_toCurrency]!;  
  
 double inUSD = input / fromRate;  
 double result = inUSD \* toRate;  
  
 setState(() {  
 \_convertedAmount = result;  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Currency Converter'),  
 centerTitle: true,  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(16),  
 child: Column(  
 children: [  
 const SizedBox(height: 10),  
 TextField(  
 controller: \_amountController,  
 keyboardType: TextInputType.*number*,  
 decoration: const InputDecoration(  
 labelText: 'Enter Amount',  
 border: OutlineInputBorder(),  
 ),  
 ),  
 const SizedBox(height: 16),  
 Row(  
 children: [  
 Expanded(  
 child: DropdownButtonFormField<String>(  
 value: \_fromCurrency,  
 decoration: const InputDecoration(  
 labelText: 'From',  
 border: OutlineInputBorder(),  
 ),  
 items: \_rates.keys  
 .map((String currency) => DropdownMenuItem<String>(  
 value: currency,  
 child: Text(currency),  
 ))  
 .toList(),  
 onChanged: (val) {  
 setState(() {  
 \_fromCurrency = val!;  
 });  
 },  
 ),  
 ),  
 const SizedBox(width: 16),  
 Expanded(  
 child: DropdownButtonFormField<String>(  
 value: \_toCurrency,  
 decoration: const InputDecoration(  
 labelText: 'To',  
 border: OutlineInputBorder(),  
 ),  
 items: \_rates.keys  
 .map((String currency) => DropdownMenuItem<String>(  
 value: currency,  
 child: Text(currency),  
 ))  
 .toList(),  
 onChanged: (val) {  
 setState(() {  
 \_toCurrency = val!;  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 const SizedBox(height: 20),  
 FilledButton(  
 onPressed: convert,  
 child: const Text('Convert Currency'),  
 ),  
 const SizedBox(height: 20),  
 Text(  
 'Converted Amount: $\_convertedAmount $\_toCurrency',  
 style: const TextStyle(fontSize: 18, fontWeight: FontWeight.*bold*),  
 )  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

This Flutter application successfully demonstrates the implementation of a basic currency converter. The app features a clean UI that allows users to input an amount, choose source and target currencies, and view the converted result instantly. It uses fundamental Flutter widgets and state management through StatefulWidget and setState(). The hardcoded exchange rates simplify the conversion logic, making it suitable for educational or demonstrational purposes. This project also strengthens understanding of widget trees, form input, dropdown menus, and state handling in Flutter.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 08: - Date:**

Create a Flutter app using Dart programming that calculates the tip amount based on the bill total and the selected tip percentage. The app should include input fields for users to enter the bill amount, optons to select the tip percentage, and butyons to calculate the tip and reset the data. Ensure that the app displays the calculated tip amount and the total bill, uses Indian currency (INR), and runs seamlessly in Android Studio, providing a user-friendly interface.

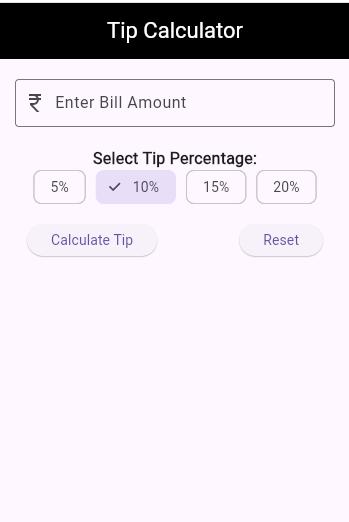
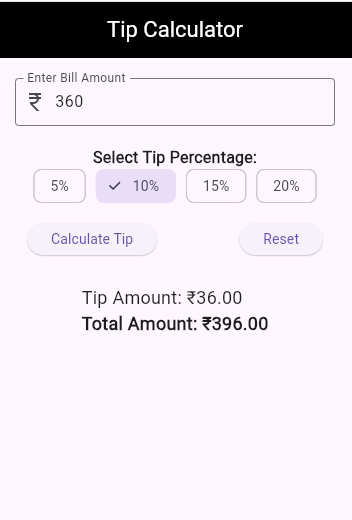
### **Design Procedure:**

1. **Objective Definition:**
   * The goal is to create a Tip Calculator application using Flutter that allows users to input a bill amount, select a tip percentage, and then calculates and displays the tip and total amount.
2. **Widget Structure Planning:**
   * The application is structured using StatelessWidget for the app container (MyApp) and a StatefulWidget (CalculateTipCalculator) for the interactive UI and logic.
3. **Input Field Implementation:**
   * A TextField is used to accept the bill amount from the user, with input validation handled by double.tryParse to ensure valid decimal values are entered.
4. **Tip Selection:**
   * Tip percentages are presented using ChoiceChip widgets within a Wrap layout. The selected tip is visually indicated, and the state updates on user selection.
5. **Logic Implementation:**
   * When the "Calculate Tip" button is pressed, the app calculates the tip based on the selected percentage and updates the UI with the tip and total amounts.
   * The calculation is done using the formula:
   * Tip Amount = Bill × Tip Percentage
   * Total Amount = Bill + Tip Amount
6. **Formatting Output:**
   * The intl package is used to format the currency in Indian Rupee (₹) format using NumberFormat.currency.
7. **Reset Functionality:**
   * A "Reset" button is included to clear all inputs and reset the state to initial values.
8. **User Feedback:**
   * An error SnackBar is shown if the user enters an invalid amount, improving user experience and error handling.
9. **UI Design Considerations:**
   * The UI includes padding, spacing, and clear labels for better usability.
   * Material Design components like AppBar, TextField, ChoiceChip, and ElevatedButton are used to maintain consistency and a professional appearance.

**Source Code**:

import 'package:flutter/material.dart';  
import 'package:intl/intl.dart';   
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: CalculateTipCalculator(),  
 );  
 }  
}  
  
class CalculateTipCalculator extends StatefulWidget {  
 const CalculateTipCalculator({super.key});  
  
 @override  
 State<CalculateTipCalculator> createState() =>  
 \_CalculateTipCalculatorState();  
}  
  
class \_CalculateTipCalculatorState extends State<CalculateTipCalculator> {  
 final TextEditingController \_billController = TextEditingController();  
   
 double \_selectedTip = 0.1;  
   
 double \_tipAmount = 0.0;  
 double \_totalAmount = 0.0;  
   
 final currencyFormatter = NumberFormat.currency(locale: 'en\_IN', symbol: '₹');  
   
 void calculateTip() {  
 final billText = \_billController.text;  
   
 final double? bill = double.*tryParse*(billText);  
 if (bill != null && bill >= 0) {  
 setState(() {  
 \_tipAmount = bill \* \_selectedTip;  
 \_totalAmount = bill + \_tipAmount;  
 });  
 } else {  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 SnackBar(content: Text('Please enter a valid bill amount.')),  
 );  
 }  
 }  
 void reset() {  
 setState(() {  
 \_billController.clear();  
 \_selectedTip = 0.1;  
 \_tipAmount = 0.0;  
 \_totalAmount = 0.0;  
 });  
 }  
   
 Widget \_buildTipChip(double percent) {  
 return ChoiceChip(  
 label: Text('${(percent \* 100).toInt()}%'),  
 selected: \_selectedTip == percent,  
 onSelected: (bool selected) {  
 setState(() {  
 \_selectedTip = percent;  
 });  
 },  
 );  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Tip Calculator'),  
 centerTitle: true,  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(20),  
 child: Column(  
 children: [  
 SizedBox(  
 width:320 ,  
 child: TextField(  
 controller: \_billController,  
 keyboardType: TextInputType.numberWithOptions(decimal: true),  
 decoration: const InputDecoration(  
 labelText: 'Enter Bill Amount',  
 border: OutlineInputBorder(),  
 prefixIcon: Icon(Icons.*currency\_rupee*),  
 ),  
 ),  
 ),  
 const SizedBox(height: 20),  
 const Text(  
 'Select Tip Percentage:',  
 style: TextStyle(fontSize: 16, fontWeight: FontWeight.*bold*),  
 ),  
 Wrap(  
 spacing: 10,  
 children: [  
 \_buildTipChip(0.05),  
 \_buildTipChip(0.1),  
 \_buildTipChip(0.15),  
 \_buildTipChip(0.2),  
 ],  
 ),  
 const SizedBox(height: 20),  
   
 Row(  
 mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
 children: [  
 ElevatedButton(  
 onPressed: calculateTip, child: const Text('Calculate Tip')),  
 ElevatedButton(onPressed: reset, child: const Text('Reset')),  
 ],  
 ),  
 const SizedBox(height: 30),  
   
 if (\_tipAmount > 0)  
 Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Text(  
 'Tip Amount: ${currencyFormatter.format(\_tipAmount)}',  
 style: const TextStyle(fontSize: 18),  
 ),  
 Text(  
 'Total Amount: ${currencyFormatter.format(\_totalAmount)}',  
 style: const TextStyle(fontSize: 18, fontWeight: FontWeight.*bold*),  
 ),  
 ],  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**The Tip Calculator app was successfully developed using Flutter with a clean and responsive user interface. It allows users to easily calculate tips based on selected percentages and provides accurate and formatted results. The app effectively demonstrates Flutter’s capabilities in building interactive, real-time UI applications. Features like input validation, state management, and user feedback ensure reliability and a good user experience. This project showcases practical use of basic Flutter widgets and stateful logic in a real-world application.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 09: - Date:**

Develop a Flutter App using Dart programming to calculate the area and circumference of a circle based on the input radius. The app should provide a dropdown list for users to select either the area or circumference calculation. Ensure that the app displays the calculated result and runs seamlessly in Android Studio, providing a user-friendly interface.

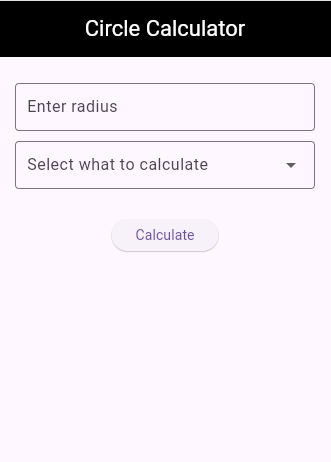
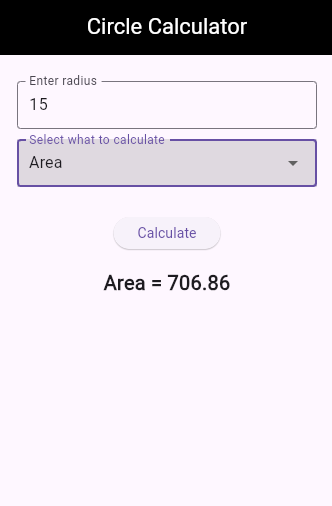
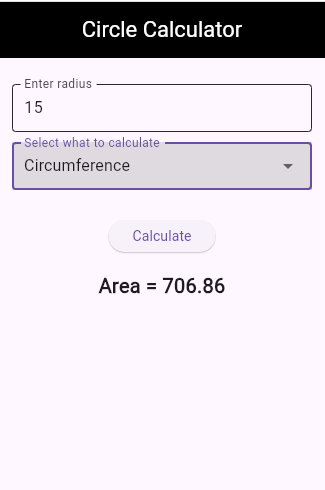
### **Design Procedure:**

1. **Main Application Structure:**
   * The MyApp class is the entry point of the application, initializing the main screen of the app, which is the CircleCalculator widget.
2. **CircleCalculator Widget:**
   * This is a stateful widget where the actual logic for the circle calculations happens.
   * It contains a TextEditingController to manage the input field for the radius, a DropdownButtonFormField to allow the user to select either "Area" or "Circumference", and a button to trigger the calculation.
3. **User Input Handling:**
   * The user enters the radius in the TextField. The value is retrieved using \_radiusController.text.
   * The user also selects the desired calculation type ("Area" or "Circumference") from the dropdown using \_selectedOption.
4. **Validation:**
   * Before performing any calculations, the app checks if the radius is a valid positive number.
   * If the radius is invalid or if no option is selected in the dropdown, an appropriate error message is displayed.
5. **Calculation Logic:**
   * The app uses the formula for area Area = π \* radius² and circumference Circumference = 2 \* π \* radius based on the user's selection.
   * The result is then displayed on the screen with two decimal precision.
6. **UI Components:**
   * The app uses common Flutter widgets like Scaffold, AppBar, TextField, DropdownButtonFormField, ElevatedButton, and Text to build the UI.
   * The UI is simple and clean, with a consistent theme using the MaterialApp widget.
7. **State Management:**
   * The app uses setState to update the UI whenever the user enters the radius or selects an option to calculate.

**Source Code**:

import 'package:flutter/material.dart';  
import 'dart:math';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: CircleCalculator(),  
 );  
 }  
}  
  
class CircleCalculator extends StatefulWidget {  
 const CircleCalculator({super.key});  
  
 @override  
 State<CircleCalculator> createState() => \_CircleCalculatorState();  
}  
  
class \_CircleCalculatorState extends State<CircleCalculator> {  
 final TextEditingController \_radiusController = TextEditingController();  
 String? \_selectedOption;  
 String \_result = '';  
  
 void \_calculate() {  
 final input = \_radiusController.text;  
 final radius = double.*tryParse*(input);  
  
 if (radius == null || radius <= 0) {  
 setState(() {  
 \_result = 'Please enter a valid positive number';  
 });  
 return;  
 }  
  
 if (\_selectedOption == null) {  
 setState(() {  
 \_result = 'Please select an option';  
 });  
 return;  
 }  
  
 double output;  
 if (\_selectedOption == 'Area') {  
 output = pi \* radius \* radius;  
 \_result = 'Area = ${output.toStringAsFixed(2)}';  
 } else if (\_selectedOption == 'Circumference') {  
 output = 2 \* pi \* radius;  
 \_result = 'Circumference = ${output.toStringAsFixed(2)}';  
 }  
  
 setState(() {});  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Circle Calculator'),  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 SingleChildScrollView(  
 padding: const EdgeInsets.all(16),  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 const SizedBox(height: 10),  
 SizedBox(  
 width: 300,  
 child: TextField(  
 controller: \_radiusController,  
 decoration: const InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Enter radius',  
 ),  
 keyboardType: TextInputType.numberWithOptions(decimal: true),  
 ),  
 ),  
 const SizedBox(height: 10),  
 SizedBox(  
 width: 300,  
 child: DropdownButtonFormField<String>(  
 decoration: const InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Select what to calculate',  
 ),  
 value: \_selectedOption,  
 items: const [  
 DropdownMenuItem(  
 value: 'Area',  
 child: Text('Area'),  
 ),  
 DropdownMenuItem(  
 value: 'Circumference',  
 child: Text('Circumference'),  
 ),  
 ],  
 onChanged: (value) {  
 setState(() {  
 \_selectedOption = value;  
 });  
 },  
 ),  
 ),  
 const SizedBox(height: 30),  
 ElevatedButton(  
 onPressed: \_calculate,  
 child: const Text('Calculate'),  
 ),  
 const SizedBox(height: 20),  
 Text(  
 \_result,  
 style: const TextStyle(fontSize: 20, fontWeight: FontWeight.*bold*),  
 ),  
 ],  
 ),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

The **Circle Calculator** app is an interactive and user-friendly tool designed to help users easily calculate the area or circumference of a circle based on the given radius. The design focuses on simplicity and ease of use, ensuring that users can input their data and receive the results without difficulty. The app incorporates essential features such as input validation, clear error messages, and dynamic UI updates based on user input. The code is organized in a way that makes it easy to extend in the future with more geometric calculations or features. This application demonstrates the practical use of Flutter for building simple yet functional applications with minimalistic design and effective state management.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 10: - Date:**

Develop a flashcards app using Flutter and Dart programming that allows users to add questions and answers. The app should enable users to review the flashcards they have created. Ensure that the app runs seamlessly in Android Studio and provides a user-friendly interface for managing and reviewing flashcards.

### **Design Procedure:**

**Source Code**:

**Code Output**:

**Conclusion:**

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 11: - Date:**

Develop a single-page app using Flutter with the following features:

* 1. An AppBar displaying the app name as "Barbecue Station."
  2. A side drawer with the name "Cuisine List."
  3. A floating action button that, when clicked, navigates to a page displaying pictures of pizza. Ensure the app runs seamlessly in Android Studio and provides a user-friendly interface.

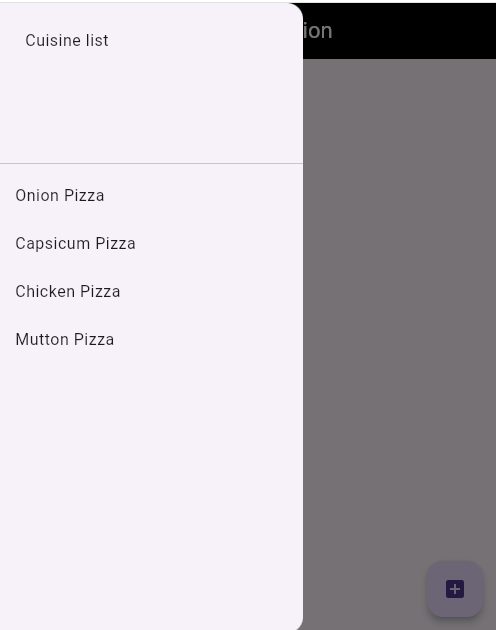
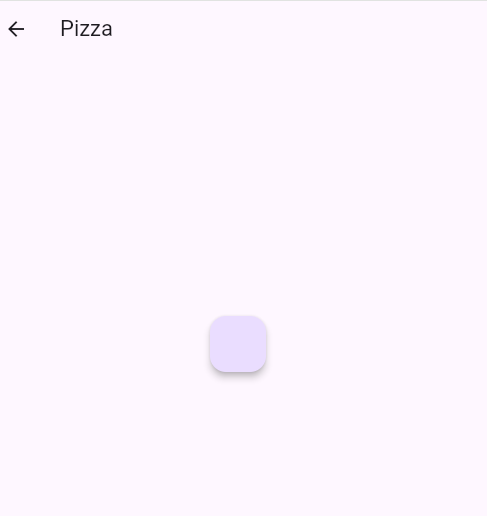
### **Design Procedure:**

1. **Create Main Entry Point**
   * The application starts with the main() function.
   * runApp(const MyApp()) is called to launch the MyApp widget.
2. **Design the Main Application (MyApp)**
   * MyApp is a StatelessWidget that returns a MaterialApp.
   * debugShowCheckedModeBanner is set to false to remove the debug label from the app.
   * The home property is assigned to the BarbecueStation widget.
3. **Create Home Screen (BarbecueStation Widget)**
   * BarbecueStation is a StatefulWidget that manages its state using \_BarbecueStationState.
   * A Scaffold is used to design the basic structure of the screen, including AppBar, Drawer, Body, and FloatingActionButton.
4. **AppBar Design**
   * The AppBar is given a black background and white text/icon color.
   * The title "Barbecue Station" is set and aligned at the center.
5. **Body Design**
   * The body contains a Column.
   * Inside the Center widget, another Column is used (currently it is empty but structured for future content).
6. **Drawer Design**
   * A Drawer widget is added to the Scaffold.
   * Inside the Drawer, a ListView is used.
   * A DrawerHeader displays the text "Cuisine list".
   * Below the header, four ListTile widgets list different pizza options: Onion Pizza, Capsicum Pizza, Chicken Pizza, and Mutton Pizza.
7. **Floating Action Button (FAB) Design**
   * When the FAB is pressed, it navigates to a new screen PizzaPage.
   * Navigator.push() is used for page transition.
8. **PizzaPage Design**
   * PizzaPage is a StatelessWidget.
   * It has an AppBar with the title "Pizza".
   * In the body, a FloatingActionButton is provided, and when pressed, it brings the user back to the previous screen using Navigator.pop().

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: BarbecueStation(),  
 );  
 }  
}  
class BarbecueStation extends StatefulWidget {  
 const BarbecueStation({super.key});  
  
 @override  
 State<BarbecueStation> createState() => \_BarbecueStationState();  
}  
class \_BarbecueStationState extends State<BarbecueStation> {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Barbecue Station'),  
 centerTitle: true,  
 ),  
 body: Column(  
 children: [  
 Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: <Widget>[  
 const Text('',), *// Text(// '$\_counter',// style: Theme.of(context).textTheme.headlineMedium,// ),* ],  
 ),  
 ),  
 ],  
 ),  
 drawer: Drawer(  
 child: ListView(  
 children: [  
 DrawerHeader(  
 child: Padding(  
 padding: const EdgeInsets.all(10),  
 child: Text('Cuisine list'),  
 )),  
 ListTile(  
 title: Text('Onion Pizza'),  
 ),  
 ListTile(  
 title: Text('Capsicum Pizza'),  
 ),  
 ListTile(  
 title: Text('Chicken Pizza'),  
 ),  
 ListTile(  
 title: Text('Mutton Pizza'),  
 ),  
 ],  
 ),  
 ),  
 floatingActionButton: FloatingActionButton(  
 onPressed: () {  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(builder: (context) => PizzaPage()),  
 );  
 },  
 tooltip: 'Increment',  
 child: const Icon(Icons.*add\_box*,),  
 ),  
 );  
 }  
}  
class PizzaPage extends StatelessWidget {  
 const PizzaPage({super.key});  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Pizza'),  
 ),  
 body: Center(  
 child: FloatingActionButton(  
 onPressed: () {  
 Navigator.*pop*(context);  
 },  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

In this project, a simple **multi-screen Flutter application** has been developed where navigation between two screens is implemented.

* A **Drawer** is used to display a list of cuisine options.
* A **FloatingActionButton** helps in navigating to another page.
* Basic components like AppBar, Drawer, FloatingActionButton, and Scaffold are used effectively.
* The structure of the application is clean and user-friendly.
* The project successfully demonstrates important Flutter concepts such as navigation, UI design, and widget management.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 12: - Date:**

Develop a Flutter app using Dart programming that displays 10 hardcoded random names upon clicking a button. Include another button that navigates to a different page. Ensure the app runs seamlessly in Android Studio and provides a user-friendly interface.

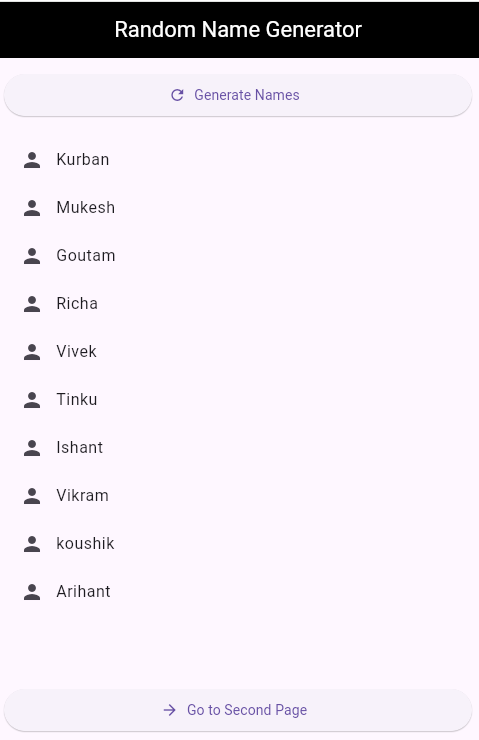
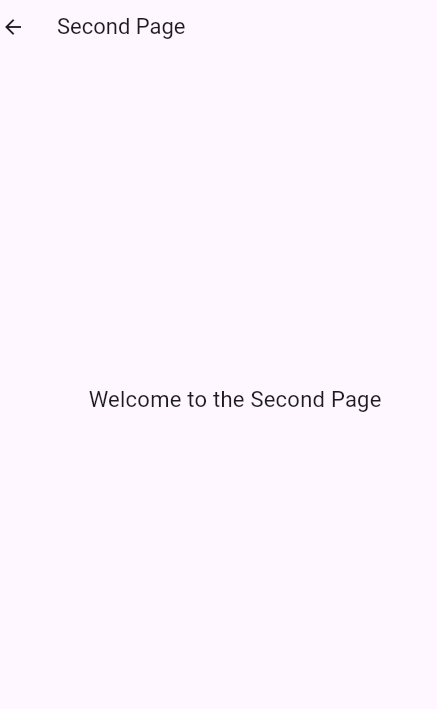
### **Design Procedure:**

1. **Project Initialization:**
   * A Flutter project was created using the flutter create command.
   * The main application logic was written in the main.dart file.
   * The root of the application is defined using the MaterialApp widget with routing and theme configurations.
2. **Main Widgets:**
   * MyApp is a StatelessWidget that sets up the app's theme, title, and routes.
   * HomePage is a StatefulWidget where the core functionality (random name generation) is implemented.
3. **Random Name Generator Logic:**
   * A predefined list allNames stores various names.
   * When the user presses the "Generate Names" button, the list is shuffled and the first 10 names are selected and displayed.
   * The ListView.builder widget is used to dynamically create list tiles for each generated name.
4. **Navigation:**
   * Navigation to another screen (SecondPage) is handled using named routes.
   * The /second route is mapped in the MaterialApp and triggered using Navigator.pushNamed().
5. **Second Page Layout:**
   * SecondPage is a simple StatelessWidget that displays a welcome message in the center of the screen.
   * An AppBar with a back button is provided for returning to the home page.
6. **UI Design:**
   * A clean and user-friendly layout is created using ElevatedButton.icon, ListTile, and proper spacing.
   * Consistent color schemes and AppBar styling enhance the visual appearance.

**Source Code**:

import 'package:flutter/material.dart';  
  
void main() {  
 runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Random Names App',  
 debugShowCheckedModeBanner: false,  
 theme: ThemeData(primarySwatch: Colors.*indigo*),  
 home: HomePage(),  
 routes: {  
 '/second': (context) => SecondPage(),  
 },  
 );  
 }  
}  
  
class HomePage extends StatefulWidget {  
 @override  
 \_HomePageState createState() => \_HomePageState();  
}  
  
class \_HomePageState extends State<HomePage> {  
 List<String> names = [];  
 final List<String> allNames = [  
 'Kurban',  
 'koushik',  
 'Ramesh',  
 'Tinku',  
 'Satyendra',  
 'Ashok',  
 'Ishant',  
 'Mukesh',  
 'Richa',  
 'Suresh',  
 'Rohit',  
 'Vikram',  
 'Goutam',  
 'Vivek',  
 'Arihant'  
 ];  
  
 void generateRandomNames() {  
 allNames.shuffle();  
 setState(() {  
 names = allNames.take(10).toList();  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: Text('Random Name Generator'),  
 centerTitle: true,  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: Column(  
 children: [  
 ElevatedButton.icon(  
 onPressed: generateRandomNames,  
 icon: Icon(Icons.*refresh*),  
 label: Text('Generate Names'),  
 style: ElevatedButton.*styleFrom*(minimumSize: Size.fromHeight(50)),  
 ),  
 SizedBox(height: 20),  
 Expanded(  
 child: ListView.builder(  
 itemCount: names.length,  
 itemBuilder: (context, index) {  
 return ListTile(  
 leading: Icon(Icons.*person*),  
 title: Text(names[index]),  
 );  
 },  
 ),  
 ),  
 ElevatedButton.icon(  
 onPressed: () {  
 Navigator.*pushNamed*(context, '/second');  
 },  
 icon: Icon(Icons.*arrow\_forward*),  
 label: Text('Go to Second Page'),  
 style: ElevatedButton.*styleFrom*(  
 *//backgroundColor: Colors.purpleAccent,* minimumSize: Size.fromHeight(50)),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}  
  
class SecondPage extends StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text('Second Page'),  
 leading: BackButton(),  
 ),  
 body: Center(  
 child: Text(  
 'Welcome to the Second Page',  
 style: TextStyle(fontSize: 22),  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

This Flutter project successfully demonstrates the use of basic widgets and functionalities such as StatelessWidget, StatefulWidget, ListView, Navigation, and random list generation using the shuffle() method. The app provides an interactive experience where users can generate random names and navigate between screens. Overall, this project is an excellent exercise for understanding core Flutter concepts including UI layout, state management, and routing.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 13: - Date:**

Develop a Flutter app using a stateful widget that changes the AppBar name when the user enters a name. The app should include an input field for the user to enter their name, and the AppBar title should update accordingly. Ensure the app runs seamlessly in Android Studio and provides a user-friendly interface.

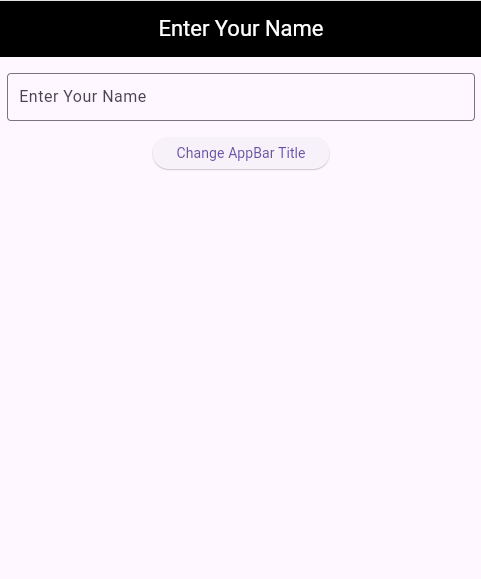
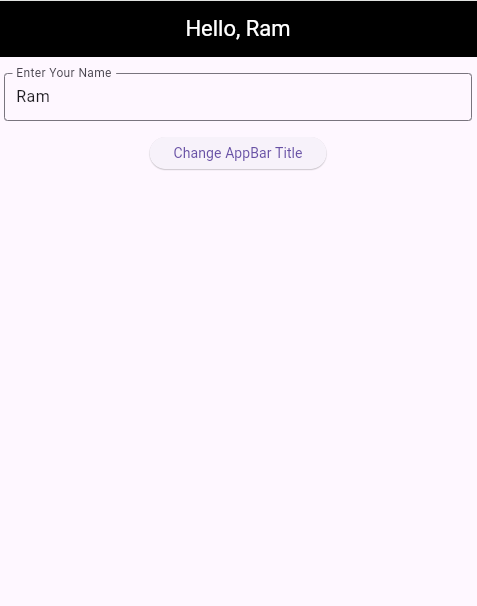
### **Design Procedure:**

1. **Project Initialization**:
   * A Flutter project is created, and inside the main() function, the runApp() method is called to launch the MyApp widget.
2. **MyApp Widget**:
   * MyApp is a **StatelessWidget**.
   * It returns a MaterialApp with:
     + A title: 'Name AppBar'
     + debugShowCheckedModeBanner set to false.
     + The home property set to ChangeAppbar widget.
3. **ChangeAppbar Widget**:
   * ChangeAppbar is a **StatefulWidget** because we need to change the AppBar title dynamically based on user input.
   * A String variable userName is used to store the entered name.
   * A TextEditingController named \_controller is created to manage the text input.
4. **Scaffold Design**:
   * The screen layout uses a Scaffold containing:
     + **AppBar**:
       - Black background and white foreground.
       - Title displays 'Enter Your Name' if no input is given, otherwise displays 'Hello, <userName>'.
       - Title is centered.
     + **Body**:
       - A TextField with an OutlineInputBorder is provided for the user to input their name.
       - An ElevatedButton is placed below the TextField:
         * When pressed, it updates the userName variable using setState().
         * This automatically refreshes the AppBar title.
5. **State Management**:
   * The setState() function is used to update the user interface whenever the user enters a name and presses the button.

**Source 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 const MaterialApp(  
 title: 'Name AppBar',  
 debugShowCheckedModeBanner: false,  
 home: ChangeAppbar(),  
 );  
 }  
}  
  
class ChangeAppbar extends StatefulWidget {  
 const ChangeAppbar({super.key});  
  
 @override  
 State<ChangeAppbar> createState() => \_ChangeAppbarState();  
}  
  
class \_ChangeAppbarState extends State<ChangeAppbar> {  
 String userName = '';  
 final TextEditingController \_controller=TextEditingController();  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 backgroundColor: Colors.*black*,  
 foregroundColor: Colors.*white*,  
 title:  
 Text(userName.isEmpty ? 'Enter Your Name' : 'Hello, $userName'),  
 centerTitle: true,  
 ),  
 body: Column(  
 children: [  
 Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: TextField(  
 controller: \_controller,  
 decoration: InputDecoration(  
 border: OutlineInputBorder(), labelText: 'Enter Your Name'),  
 ),  
 ),  
 ElevatedButton(onPressed: (){  
 setState(() {  
 userName = \_controller.text ;  
 });  
 }, child: Text('Change AppBar Title'))  
 ],  
 )  
   
 );  
 }  
}

**Code Output**:

**Conclusion:**

In this project, we developed a simple Flutter application where the AppBar title changes dynamically based on user input.  
We demonstrated the use of a **StatefulWidget** to update the UI in real-time.  
When the user types their name and presses the button, the AppBar immediately updates to show "Hello, ".  
Through this application, we learned how to handle **TextField input**, use a **TextEditingController**, manage UI updates with **setState()**, and design a basic interactive app layout in Flutter.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 14: - Date:**

Develop a Flutter app with a login page that includes the following validation rules:

* 1. The username must be at least 6 characters long.
  2. The email should be valid.
  3. The password must contain: i. An uppercase letter ii. A lowercase letter iii. Digits iv. A special character v. More than 6 characters

The login button should be displayed, and when clicked, it should show any validation errors if present. Ensure the app runs seamlessly in Android Studio and provides a user-friendly interface.

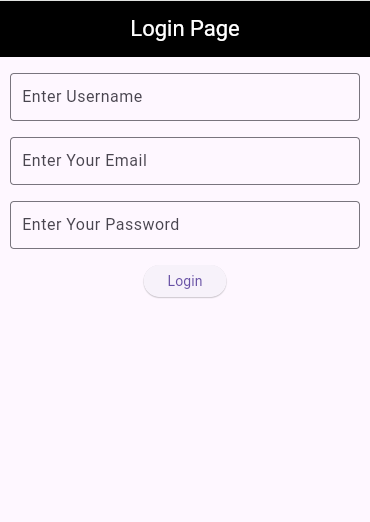
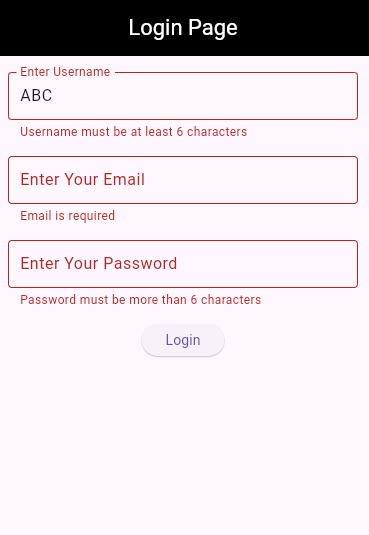
### **Design Procedure:**

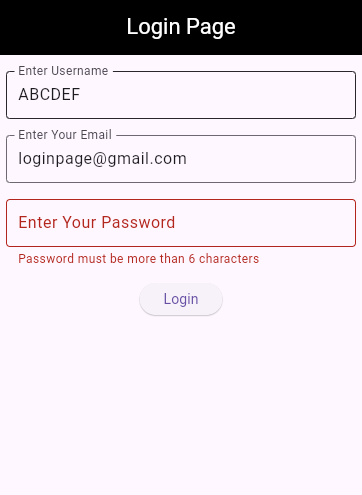
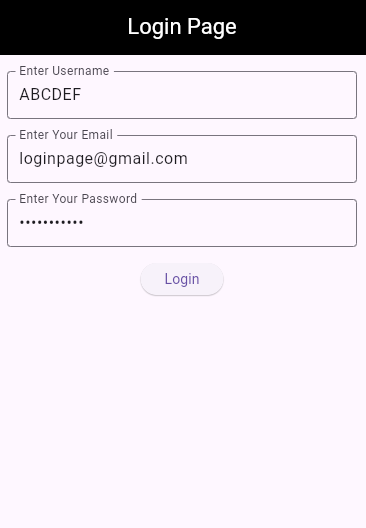
1. **Project Initialization**:
   * A new Flutter project was created using flutter create login\_app.
   * The main Dart file (main.dart) was modified to implement a user login page with input validation.
2. **Basic Structure Setup**:
   * The main() function runs the MyApp widget, which returns a MaterialApp with the LoginPage as the home screen.
   * debugShowCheckedModeBanner is set to false for a cleaner UI.
3. **Stateful Widget for Login Page**:
   * LoginPage is a StatefulWidget so that UI can be updated dynamically (e.g., for autovalidation).
   * A GlobalKey<FormState> is used to manage and validate the form.
4. **Form UI Design**:
   * The form contains three TextFormField widgets:
     + **Username**: Validates for a minimum length of 6 characters.
     + **Email**: Validates for presence of '@' and '.'.
     + **Password**: Checks for multiple conditions like length, uppercase, lowercase, digit, and special character.
   * Each field has an OutlineInputBorder for clear UI separation.
5. **Validation Handling**:
   * Initially, form autovalidation is off.
   * When the "Login" button is pressed:
     + AutovalidateMode is changed to always to show validation messages.
     + If all validations pass, a SnackBar is shown with "Login Successful".
6. **UI Styling**:
   * An AppBar with a black background and white text is used.
   * Padding is applied using EdgeInsets.all(16.0) to space the form properly.
   * SizedBox is used between fields for spacing.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: LoginPage(),  
 );  
 }  
}  
  
class LoginPage extends StatefulWidget {  
 const LoginPage({super.key});  
  
 @override  
 State<LoginPage> createState() => \_LoginPageState();  
}  
  
class \_LoginPageState extends State<LoginPage> {  
 final \_formKey = GlobalKey<FormState>();  
 AutovalidateMode validate = AutovalidateMode.disabled;  
  
 String? validateEmail(String? value) {  
 if (value == null || value.isEmpty) {  
 return 'Email is required';  
 }  
 if (!value.contains('@') || !value.contains('.')) {  
 return 'Enter a valid email';  
 }  
 return null;  
 }  
  
 String? validateUsername(String? value) {  
 if (value == null || value.length < 6) {  
 return 'Username must be at least 6 characters';  
 }  
 return null;  
 }  
  
 String? validatePassword(String? value) {  
 if (value == null || value.length <= 6) {  
 return 'Password must be more than 6 characters';  
 }  
 if (!value.contains(RegExp(r'[A-Z]'))) {  
 return 'Include at least one uppercase letter';  
 }  
 if (!value.contains(RegExp(r'[a-z]'))) {  
 return 'Include at least one lowercase letter';  
 }  
 if (!value.contains(RegExp(r'[0-9]'))) {  
 return 'Include at least one digit';  
 }  
 if (!value.contains(RegExp(r'[!@#\$&\*~]'))) {  
 return 'Include at least one special character';  
 }  
 return null;  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Login Page'),  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: Form(  
 key: \_formKey,  
 autovalidateMode: validate,  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 SizedBox(  
 width: 350,  
 child: TextFormField(  
 decoration: const InputDecoration(  
 border: OutlineInputBorder(), labelText: 'Enter Username'),  
 validator: validateUsername,  
 ),  
 ),  
 const SizedBox(height: 16),  
 SizedBox(  
 width: 350,  
 child: TextFormField(  
 decoration: const InputDecoration(  
 border: OutlineInputBorder(), labelText: 'Enter Your Email'),  
 validator: validateEmail,  
 ),  
 ),  
 const SizedBox(height: 16),  
 SizedBox(  
 width: 350,  
 child: TextFormField(  
 obscureText: true,  
 decoration: const InputDecoration(  
 border: OutlineInputBorder(),  
 labelText: 'Enter Your Password'),  
 validator: validatePassword,  
 ),  
 ),  
 const SizedBox(height: 16),  
 ElevatedButton(  
 onPressed: () {  
 setState(() {  
 validate = AutovalidateMode.always;  
 });  
  
 if (\_formKey.currentState!.validate()) {  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 const SnackBar(content: Text('Login Successful')),  
 );  
 }  
 },  
 child: const Text('Login'))  
 ],  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**In this project, a simple yet effective **Login Page UI** was designed using **Flutter**. The application demonstrates the use of **form validation**, **state management**, and **Flutter widgets** like TextFormField, Form, and Scaffold. It ensures proper validation of **username**, **email**, and **password** with user feedback. This login page can serve as the foundation for any authentication system in a Flutter application.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 15: - Date:**

Develop a Flutter app with a button that animates a Container's width, height, and colour over a period of 1 second when pressed. The container should initially have a width and height of 100.0 and a colour of red. When the button is pressed, the container should animate to a width of 200.0, height of 200.0, and colour of blue. Ensure the app runs seamlessly in Android Studio and provides a visually engaging animation.

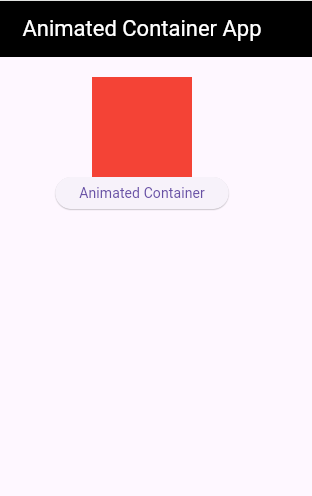
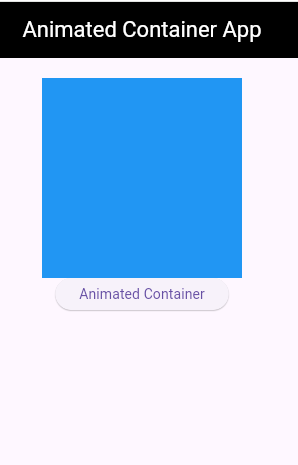
### **Design Procedure:**

1. **Project Setup:**
   * Flutter SDK installed and a new Flutter project created.
   * Main entry point is defined using main() function which runs the MyApp widget.
2. **MyApp Widget:**
   * A StatelessWidget that returns a MaterialApp.
   * The home property is set to AnimatedContainerApp, which is the main UI.
3. **AnimatedContainerApp Widget:**
   * A StatefulWidget is used because the UI will change dynamically using setState().
4. **State Management:**
   * The \_AnimatedContainerAppState class holds mutable state variables: \_width, \_height, and color.
   * Initial values: width = 100, height = 100, color = red.
5. **UI Layout:**
   * A Scaffold widget is used for the screen structure.
   * AppBar is configured with a title and styled with a black background and white text.
   * Column widget is used for vertical layout:
     + SizedBox is used to give spacing at the top.
     + A Center widget wraps the AnimatedContainer to center it horizontally.
     + The AnimatedContainer animates its width, height, and color over a duration of 1 second using the easeInOut curve.
     + An ElevatedButton is placed below, which when pressed, changes the container’s properties using setState().
6. **Animation Trigger:**
   * When the button is pressed, the container smoothly transitions to:
     + Width = 200
     + Height = 200
     + Color = blue

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: AnimatedContainerApp(),  
 );  
 }  
}  
class AnimatedContainerApp extends StatefulWidget {  
 const AnimatedContainerApp({super.key});  
  
 @override  
 State<AnimatedContainerApp> createState() => \_AnimatedContainerAppState();  
}  
  
class \_AnimatedContainerAppState extends State<AnimatedContainerApp> {  
 var \_width=100.0;  
 var \_height=100.0;  
 var color=Colors.*red*;  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Animated Container App'),  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 centerTitle: true,  
 ),  
 body: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 SizedBox(height: 20,),  
 Center(  
 child: AnimatedContainer(duration:Duration(seconds: 1),  
 width: \_width,  
 height: \_height,  
 color: color,  
 curve: Curves.*easeInOut*,  
 ),  
 ),  
 ElevatedButton(onPressed: ()=> setState(() {  
 \_width=200.0;  
 \_height=200.0;  
 color=Colors.*blue*;  
 }), child: Text('Animated Container'))  
 ],  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

* The Flutter application successfully demonstrates the use of the AnimatedContainer widget to create smooth UI animations without manually handling the animation controllers.
* On button press, the container's size and color change with animation, enhancing the user experience.
* The use of setState() ensures that the widget rebuilds and the animation is triggered.
* This project helps in understanding the basics of Flutter animations and state management using StatefulWidget.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 16: - Date:**

Develop a simple Flutter app that includes two images. Utilize the AnimatedCrossFade widget to create a smooth transition between the two images when a button is pressed. Ensure the app runs seamlessly in Android Studio and provides a visually engaging experience.

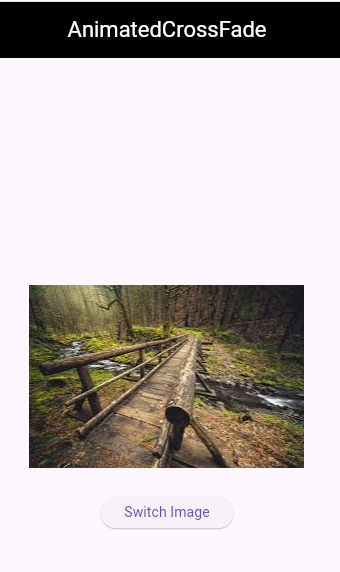
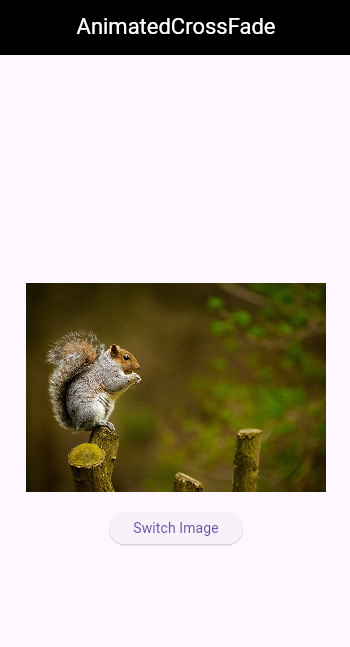
### **Design Procedure:**

1. **Import Flutter Material Package:**
   * import 'package:flutter/material.dart'; is used to access all the UI components provided by Flutter's Material Design library.
2. **Main Function and Root Widget:**
   * The main() function calls runApp() and runs the root widget MyApp.
   * MyApp is a StatelessWidget that returns a MaterialApp with debugShowCheckedModeBanner set to false, and sets the home widget as ImageFadeSwitcher.
3. **Stateful Widget for Image Switching:**
   * ImageFadeSwitcher is a StatefulWidget that allows toggling between two images.
4. **Managing State:**
   * A boolean variable \_showFirst is used to determine which image to show.
   * \_toggleImage() function toggles this variable and calls setState() to update the UI.
5. **UI Design with Scaffold:**
   * A Scaffold widget is used to create the basic material layout structure with an AppBar and a body.
6. **Image Transition Using AnimatedCrossFade:**
   * Two Image.network widgets are used as firstChild and secondChild in AnimatedCrossFade.
   * The images fade between each other based on the value of \_showFirst.
   * The transition duration is set to 1 second using Duration(seconds: 1).
7. **Switch Button:**
   * A simple ElevatedButton is added below the images to call \_toggleImage() and switch between images when pressed.

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: ImageFadeSwitcher(),  
 );  
 }  
}  
  
class ImageFadeSwitcher extends StatefulWidget {  
 const ImageFadeSwitcher({super.key});  
  
 @override  
 State<ImageFadeSwitcher> createState() => \_ImageFadeSwitcherState();  
}  
  
class \_ImageFadeSwitcherState extends State<ImageFadeSwitcher> {  
 bool \_showFirst = true;  
  
 void \_toggleImage() {  
 setState(() {  
 \_showFirst = !\_showFirst;  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('AnimatedCrossFade'),  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: <Widget>[  
 AnimatedCrossFade(  
 duration: const Duration(seconds: 1),  
 firstChild: Image.network(  
 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQXt27d0oVdyA1uFSG4b7SwMDcvaS0frjme3A&s', *// Random image 1* width: 300,  
 ),  
 secondChild: Image.network(  
 'https://shotkit.com/wp-content/uploads/bb-plugin/cache/nature-photography-landscape-a892fa3059994f7f5119d4a53fadada3-zybravgx2q47.jpg', *// Random image 2* width: 300,  
 ),  
 crossFadeState: \_showFirst  
 ? CrossFadeState.showFirst  
 : CrossFadeState.showSecond,  
 ),  
 const SizedBox(height: 20),  
 ElevatedButton(  
 onPressed: \_toggleImage,  
 child: const Text('Switch Image'),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
}

**Code Output**:

**Conclusion:**

In this project, a simple Flutter application was designed to demonstrate image switching using the AnimatedCrossFade widget. Two images from the internet were used, and the user can switch between them using a button. The animation provides a smooth cross-fade effect, enhancing the user experience. This project helped understand how to manage widget state, use network images, and implement built-in animations in Flutter.

----------------------------------------------------------

**Teachers Signature**

**ASSIGNMENT 17: - Date:**

Basic Flutter Mobile App

- Create a simple Flutter app with a single screen that displays "Hello, Flutter!".

- Add a button that changes the text to "Hello, World!" when pressed.

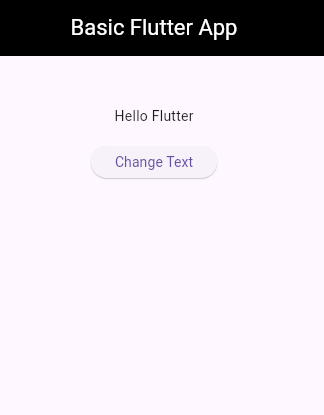
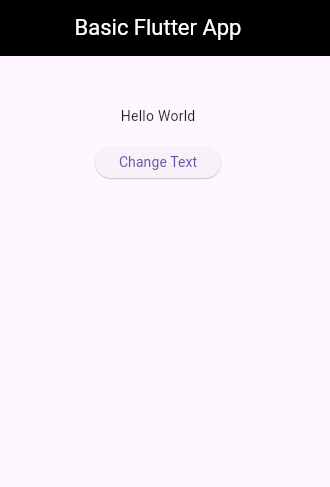
### **Design Procedure:**

1. **Import the Flutter Material Package**
   * import 'package:flutter/material.dart';  
     This provides access to all the UI widgets and material design components.
2. **Create the main() Function**
   * The entry point of the Flutter app.
   * runApp() function is called with the root widget MyApp.
3. **Create a Stateless Widget (MyApp)**
   * It returns a MaterialApp widget.
   * debugShowCheckedModeBanner is set to false to remove the debug banner.
   * The home screen is set to BasicFlutterApp widget.
4. **Create a Stateful Widget (BasicFlutterApp)**
   * Allows UI to change dynamically using setState().
5. **Declare a State Variable (displayText)**
   * Holds the string to be shown on the screen.
   * Initialized with 'Hello Flutter'.
6. **Create a Function changeText()**
   * It updates the displayText variable to 'Hello World'.
   * The UI is refreshed using setState().
7. **Build the UI in build() Method**
   * Scaffold widget provides the basic app layout.
   * AppBar with black background and white text.
   * Center widget aligns content in the middle.
   * A Column widget is used to arrange:
     + A Text widget showing displayText.
     + An ElevatedButton labeled 'Change Text'.
   * When the button is pressed, it calls changeText().

**Source 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 const MaterialApp(  
 debugShowCheckedModeBanner: false,  
 home: BasicFlutterApp(),  
 );  
 }  
}  
  
class BasicFlutterApp extends StatefulWidget {  
 const BasicFlutterApp({super.key});  
  
 @override  
 State<BasicFlutterApp> createState() => \_BasicFlutterAppState();  
}  
  
class \_BasicFlutterAppState extends State<BasicFlutterApp> {  
 String displayText='Hello Flutter';  
 void changeText(){  
 setState(() {  
 displayText='Hello World';  
 });  
 }  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 foregroundColor: Colors.*white*,  
 backgroundColor: Colors.*black*,  
 title: const Text('Basic Flutter App'),  
 centerTitle: true,  
 ),  
 body: Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.start,  
 children: [  
 SizedBox(height: 50,),  
 Text(displayText),SizedBox(height: 20,),  
 ElevatedButton(onPressed: changeText, child: const Text('Change Text'))  
  
 ],  
 ),  
 ),  
   
 );  
 }  
}

**Code Output**:

**Conclusion:**

In this Flutter application, a simple interactive user interface was created using StatelessWidget and StatefulWidget. Initially, the text 'Hello Flutter' is displayed. When the user taps the "Change Text" button, the setState() method updates the text to 'Hello World', demonstrating state management in Flutter. This project helped in understanding widget hierarchy, stateful behavior, and UI rendering in a Flutter app.

----------------------------------------------------------

**Teachers Signature**