import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Calculator(),

    );

  }

}

class Calculator extends StatefulWidget {

  @override

  \_CalculatorState createState() => \_CalculatorState();

}

class \_CalculatorState extends State<Calculator> {

  String displayText = '0'; // To store the current display text

  bool isDecimalUsed = false; // Track whether decimal point has been used

  bool isOperatorLast = false; // Check if the last character is an operator

  // This function is called when a button is pressed

  void \_onButtonPressed(String buttonText) {

    setState(() {

      if (buttonText == 'C') {

        displayText = '0'; // Clear the display

        isDecimalUsed = false; // Reset decimal usage

        isOperatorLast = false; // Reset operator check

      } else if (buttonText == '=') {

        \_calculateResult();

      } else {

        // Handle button press based on context

        if (buttonText == '.') {

          // Only allow '.' if it's not already used in the current number

          if (!isDecimalUsed && !isOperatorLast) {

            displayText += buttonText; // Append decimal point

            isDecimalUsed = true; // Mark decimal as used

          }

        } else if (buttonText == '+' ||

            buttonText == '-' ||

            buttonText == '\*' ||

            buttonText == '/') {

          // Prevent adding operator after operator

          if (!isOperatorLast) {

            displayText += buttonText; // Add operator to the display

            isOperatorLast = true; // Mark operator as last character

            isDecimalUsed = false; // Reset decimal usage

          }

        } else {

          // Append numbers or other valid characters

          if (displayText == '0') {

            displayText =

                buttonText; // Replace '0' with the first pressed number/operator

          } else {

            displayText += buttonText; // Append non-decimal button text

          }

          isOperatorLast = false; // Reset operator check

        }

      }

    });

  }

  // This function performs the actual calculation

  void \_calculateResult() {

    try {

      final result = \_evaluateExpression(displayText);

      setState(() {

        // Format result to avoid showing unnecessary decimals like 3.0

        displayText =

            result % 1 == 0 ? result.toStringAsFixed(0) : result.toString();

        isDecimalUsed = false; // Reset decimal flag after calculation

        isOperatorLast = false; // Reset operator check after calculation

      });

    } catch (e) {

      setState(() {

        displayText = 'Error'; // Show error if invalid expression

      });

    }

  }

  // Basic function for evaluating simple math expressions

  double \_evaluateExpression(String expression) {

    expression = expression.replaceAll('x', '\*').replaceAll('÷', '/');

    // Now handle each operand and operator separately

    List<String> operands = \_splitOperands(expression);

    List<String> operators = \_splitOperators(expression);

    double result = double.parse(operands[0]);

    for (int i = 1; i < operands.length; i++) {

      switch (operators[i - 1]) {

        case '+':

          result += double.parse(operands[i]);

          break;

        case '-':

          result -= double.parse(operands[i]);

          break;

        case '\*':

          result \*= double.parse(operands[i]);

          break;

        case '/':

          if (double.parse(operands[i]) == 0) {

            throw FormatException("Cannot divide by zero");

          }

          result /= double.parse(operands[i]);

          break;

      }

    }

    return result;

  }

  // Split the expression into operands

  List<String> \_splitOperands(String expression) {

    return expression.split(RegExp(r'[\+\-\\*/]')); // Split by operators

  }

  // Split the expression into operators

  List<String> \_splitOperators(String expression) {

    return expression

        .split(RegExp(r'[\d]'))

        .where((s) => s.isNotEmpty)

        .toList();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Color(

          0xffa9f8f8), // Light blue background color for the entire screen

      body: Container(

        margin: EdgeInsets.all(

            0), // Margin is set to 0 so the container stretches to the edge

        decoration: BoxDecoration(

          border: Border.all(

            color: Colors.yellow, // Yellow border

            width: 2, // Thin yellow border

          ),

        ),

        child: Column(

          children: [

            // Display (with rounded corners on all sides)

            Container(

              padding: EdgeInsets.symmetric(horizontal: 10),

              margin: EdgeInsets.only(top: 60, bottom: 60),

              width: MediaQuery.of(context).size.width \* 0.9,

              height: 40, // Make the display thinner

              decoration: BoxDecoration(

                color: Colors.white,

                borderRadius: BorderRadius.all(

                    Radius.circular(0)), // No border for the display

                boxShadow: [

                  BoxShadow(

                    color: Colors.grey.withOpacity(0.3),

                    blurRadius: 10,

                    spreadRadius: 2,

                  ),

                ],

              ),

              child: Align(

                alignment: Alignment.centerLeft,

                child: Text(

                  displayText, // Display the current expression or result

                  style: TextStyle(

                    fontSize: 28,

                    fontWeight:

                        FontWeight.bold, // Bold font for the display text

                  ),

                ),

              ),

            ),

            // Buttons (4 x 4 grid)

            Expanded(

              child: Padding(

                padding: const EdgeInsets.all(20.0), // Adjust button spacing

                child: GridView.builder(

                  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

                    crossAxisCount: 4, // 4 columns

                    crossAxisSpacing: 40, // Smaller space between buttons

                    mainAxisSpacing: 40, // Smaller space between buttons

                    childAspectRatio: 1,

                  ),

                  itemCount: 16, // 16 buttons (including empty one)

                  itemBuilder: (context, index) {

                    // Button labels (added an empty space ' ' button)

                    List<String> buttons = [

                      '1',

                      '2',

                      '3',

                      'C',

                      '4',

                      '5',

                      '6',

                      '+',

                      '7',

                      '8',

                      '9',

                      '-',

                      ' ',

                      '0',

                      '.',

                      '='

                    ];

                    String buttonLabel = buttons[index];

                    // If the button is empty, use a SizedBox to make it invisible

                    if (buttonLabel == ' ') {

                      return SizedBox

                          .shrink(); // Makes the button invisible but maintains the layout

                    }

                    // Get button color

                    Color buttonColor = \_getButtonColor(buttonLabel);

                    Color textColor = \_getTextColor(buttonLabel);

                    return ElevatedButton(

                      onPressed: () =>

                          \_onButtonPressed(buttonLabel), // Handle button press

                      style: ButtonStyle(

                        backgroundColor: MaterialStateProperty.all(

                          Color(0xfff1f4f4), // Buttons are white

                        ),

                        foregroundColor: MaterialStateProperty.all(

                          textColor, // Button text color

                        ),

                        padding: MaterialStateProperty.all(EdgeInsets.all(

                            5)), // Smaller padding for smaller buttons

                        shape: MaterialStateProperty.all(

                          RoundedRectangleBorder(

                            borderRadius: BorderRadius.circular(0),

                            side: BorderSide(

                              color: Color(0xfff4837a), // No border

                              width: 0, // No border width

                            ),

                          ),

                        ),

                        elevation: MaterialStateProperty.all(5),

                      ),

                      child: Text(

                        buttonLabel,

                        style: TextStyle(

                          fontSize:

                              18, // Adjusted font size for smaller buttons

                          color: textColor, // Text color

                          fontWeight: FontWeight.bold, // Make the text bold

                        ),

                      ),

                    );

                  },

                ),

              ),

            ),

          ],

        ),

      ),

    );

  }

  // Get button color based on label

  Color \_getButtonColor(String button) {

    if (button == 'C' ||

        button == '+' ||

        button == '-' ||

        button == '=' ||

        button == '.') {

      return Colors.red; // 'C', '+', '-', '=', '.' buttons are red

    } else {

      return Colors.white70; // Other buttons are light white

    }

  }

  // Get button text color based on label

  Color \_getTextColor(String button) {

    if (button == 'C' ||

        button == '+' ||

        button == '-' ||

        button == '=' ||

        button == '.') {

      return Colors.red;

    } else {

      return Colors.black;

    }

  }

}