**Dicee**

Flutter Preview:



Code Explanation (step by step):

1. Setup Flutter Projects -> go to main.dart files -> delete all values
2. Set up the main Function

void main() {

  runApp(

    const MaterialApp(

      home: Screen(),

    ),

  );

}

1. Setup a screen.dart file to store a stateless widget ‘Screen’ because the screen will never have a function/ purpose to rendering itself more than once except for the dice (details at step ‘’).
2. Make the UI, with an AppBar and the Body
3. To achieve a gradient background, use a container to get the type ‘decoration’ and ‘box decoration’ class. Then set the gradient as shown below.

body: Container(

        decoration: const BoxDecoration(

          gradient: LinearGradient(

              colors: [Colors.deepPurple, Color.fromARGB(255, 67, 37, 118)],

              begin: Alignment.topLeft,

              end: Alignment.bottomRight),

        ),

1. Inside the container, there should be a center widget, since we can to center the widget to the middle, and use column with an attribute ‘mainaxisalignment’ to center it both horizontally and vertically.

mainAxisAlignment: MainAxisAlignment.center,

1. Since the font in the app use poppins, we can use the google fonts extension created by flutter that provides many fonts.

**How use Google Fonts Extension by Flutter**

1. Go to <https://pub.dev/packages/google_fonts> and in the installing section, copy

flutter pub add google\_fonts

and paste it in the VSCode terminal

A computer screen with white text

Description automatically generated

1. Import the extension

import 'package:google\_fonts/google\_fonts.dart';

1. Use, these format to use Google Fonts and obtain the poppins font.

Text(

'How to use:',

style: GoogleFonts.poppins(color: Colors.white, fontSize: 18),

),

1. Make a new file ‘dice.dart’ to show the dice and the digit shown, we make a new file to simplify the code because this .dart file will be using a **Stateful Widget.** Stateful Widget will be responsible for making a UI that will render something, for example, the dice and the digit shown will always be different every time the dice is clicked. But the ‘how to:’ section will fit the best in a stateless widget because it will always be the same word how many times the dice is rolled.
2. Use a shortcut ‘stfl’ to be able to make a Stateful Widget. The widget will consist of 2 class, one is the extended Stateful and another is a class that extends the state of the actual class.

A computer screen with text

Description automatically generated

1. To make a row in the column parent widget, we can use **Row** widget to display the dice and the digit shown and align it to the left with mainaxisalignment. To make the dice tap-able, use the **GestureDetector Widget** and put the Image there using Image.asset
2. We can make the dice randomized by making a random digit number first. Make 2 variables using **var** that will keep the next number and current number.
3. Then, make a void function that yields like this:

var nextNumber = 1;

  var currentNumber = 0;

  void randomize() {

    currentNumber = nextNumber;

    setState(() {

      nextNumber = Random().nextInt(6) + 1;

    });

    if (nextNumber == currentNumber) {

      randomize();

    }

  }

SetState will be responsible for refreshing/ re-rendering the UI, here we will refresh the UI after we randomize because the dice will always be different based on the next number.

1. Set the image like this:

child: Image.asset(

              'images/dice$nextNumber.png',

              width: 100,

            ),

‘$’ means to add the value of a variable inside a string. Here we put it inside the I mage.asset since the name of the dice is incrementing from dice1.png to dice6.png.

1. To be able to show the value of the number in the UI, use a Text widget and type ‘variable\_name.toString()’. toString() will convert the variable to a string so that it can be shown in the UI.
2. Use initState above the build Widget to set something that will be set once everytime the class was called. In the code below, nextNumber will always be randomized first, then show the result to the UI. After it being called once, the function will never be called again until the class function is called again.

@override

  void initState() {

    // TODO: implement initState

    super.initState();

    nextNumber = Random().nextInt(6) + 1;

  }

Github Link: