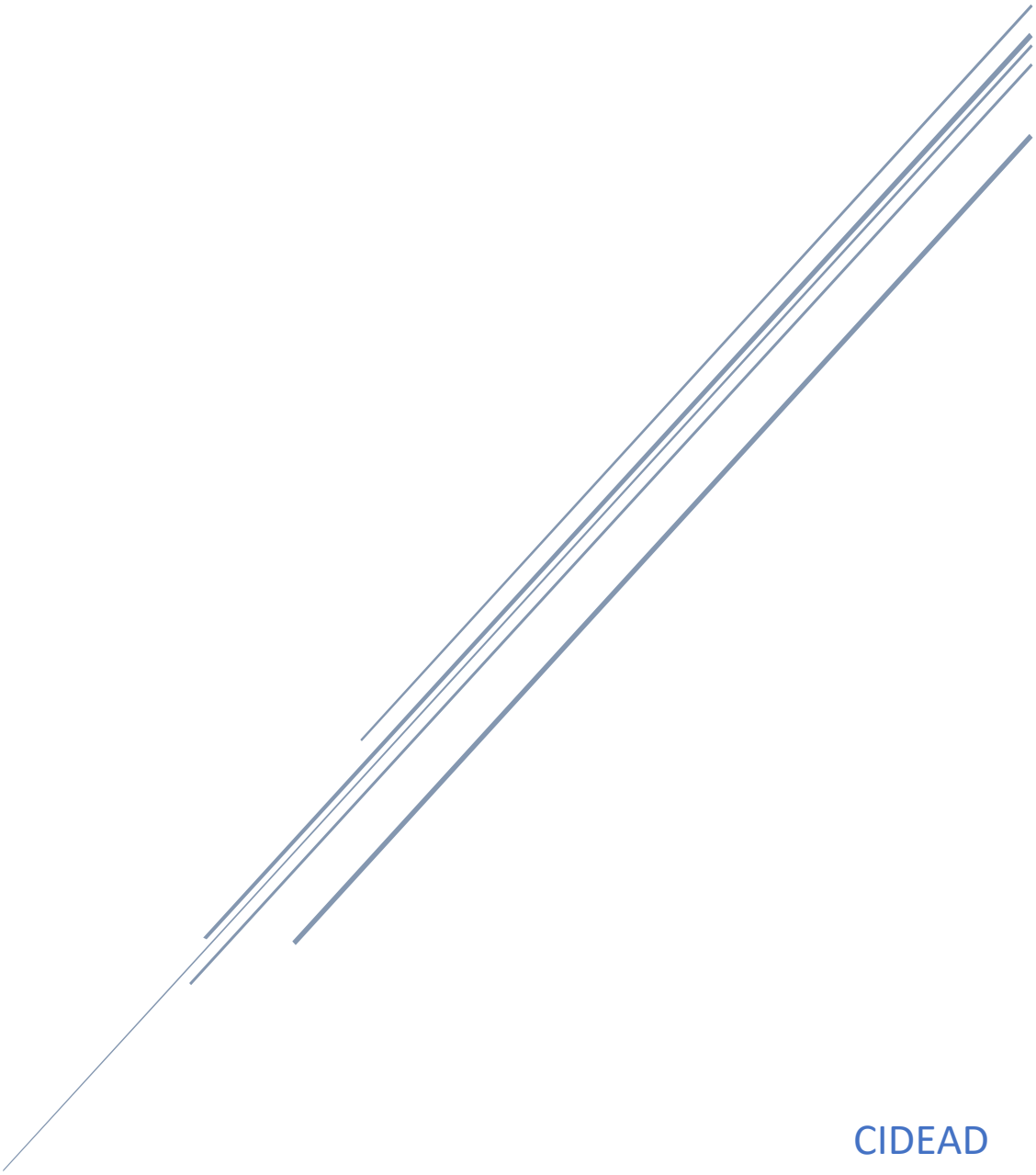
TASK 3 – DEVELOPMENT OF BASIC APPLICATIONS FOR ANDROID

Ismael Juárez García



CIDEAD

HIGHER DEGREE MULTIPLATFORM APPLICATION DEVELOPMENT

TASK UT03.1. ADD NUMBERS WITH PROGRAMMATIC BUTTON

DESCRIPTION

The application performs the sum of two integers received by the user's input in two EditText, the sum is made by pressing a "Sum" button.

LAYOUT STRUCTURE – *activity_main.xml*

The layout is structured as follows:

- RelativeLayout (id/main), elemento contenedor de las demas Views.
  - **LinearLayout** (id/block1), vertically oriented linear layout.
    - **TextView** (id/title), a text element that contains a "Sum" title that is displayed at the top of the screen.
    - **EditText** (id/number1), element to enter the first number. Only the numeric keypad is displayed:
      android:inputType = "number".
    - **EditText** (id/number2), element to enter the second number. Only the numeric keypad is displayed:
      android:inputType = "number".
  - **LinearLayout** (id/block2), horizontally oriented linear layout.
    - **Button** (id/bt_suma), a button that receives the onClick() event, programmed to sum the number1 and number2 when pressed.
    - **TextView** (id/result), text that shows the result of the sum.

PROGRAMMING *MainActivity.kt*

The UI design created for the activity is established:

setContentView(R.layout.activity_main)

We adjusted the margins of the View and adjusted them to the status bar and navigation bar:

configureMargins(findViewById(R.id.main))

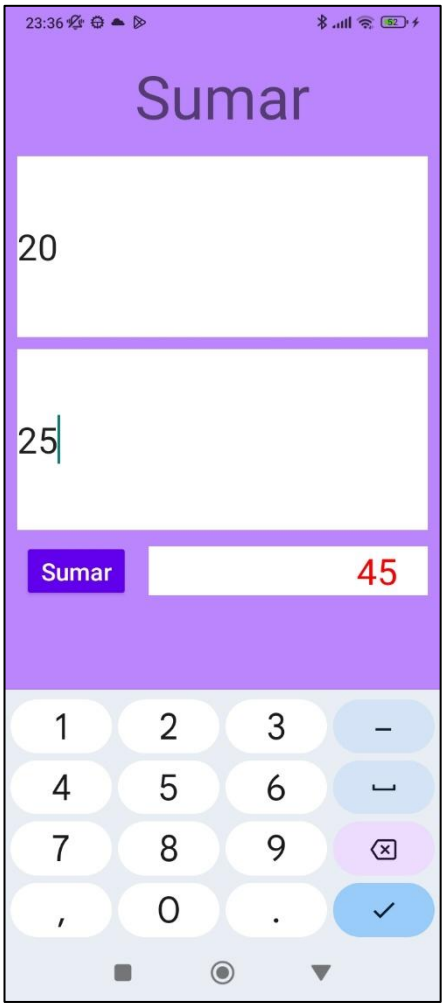
```
private fun configureMargins(view: View) {
    ViewCompat.setOnApplyWindowInsetsListener(view) { v, insets ->
        val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
        val leftPadding=20
        val rightPadding=20
        v.setPadding(
              leftPadding // Margen izquierdo personalizado
            , systemBars.top // Margen superior adaptado a las barras del sistema
            , rightPadding // Margen derecho personalizado
            , systemBars.bottom // Margen inferior adaptado a las barras del sistema
        )
        insets
    }
}
```

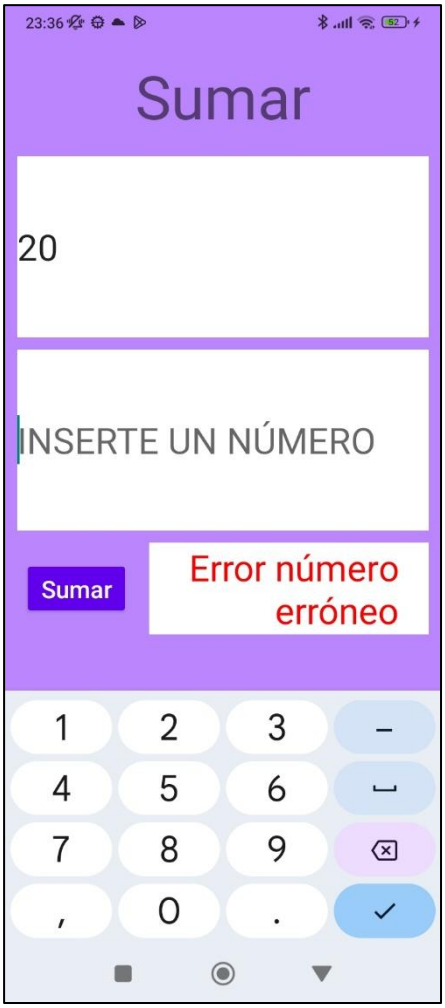
Programmatic button bt_suma, receives the **onClick -> numberAddition event** that performs the following function:

```
fun numberAddition(view: View){
    // Declaración variables textBox1 y textBox2 que recogen los TextViews number1 y number2
    val textBox1: EditText = findViewById(R.id.number1)
    val textBox2: EditText = findViewById(R.id.number2)
    // Declaración variable resultTextView que relacionamos con el TextView que mostrará el resultado
    val resultTextView: TextView = findViewById(R.id.result)
    // Controlamos de errores -> Se controla que se introduzca un número válido, es decir Int.
    try {
        // Declaramos los números que vamos a sumar,
        // transformamos el campo text de los EditText a Int
        val number1 = textBox1.text.toString().toInt()
        val number2 = textBox2.text.toString().toInt()
        // Se realiza la suma y se guarda en la variable addition
        val addition = number1 + number2
        // Guardamos el resultado transformado a string en el campo texto del TextView
        resultTextView.text = addition.toString()
    }catch(e: NumberFormatException ){
        // Si se captura el error de número inválido
        // enviamos el mensaje de error para mostrar en el TextView del resultado
        resultTextView.text= getString(R.string.number_error)
    }
}
```

TESTS

Screenshots of the tests carried out to check the operation of the application are provided:





TASK UT03.2. ADD NUMBERS WITH PROGRAMMATIC BUTTON

DESCRIPTION

The application is a simple calculator with two real type operands and four RadioButtons in which the operation to be performed (addition, subtraction, multiplication and division) will be selected.

The RadioButtons manage and handle the selection event of any of them.

Type and division exceptions are handled by 0, which will show an error using the Toast class.

LAYOUT STRUCTURE – *activity_main.xml*

The layout follows the following structure:

- LinearLayout (id/main), the container element of the other Views.
    - **TextView** (id/title), a text element containing a title "Calculator Pro 3000" displayed at the top of the screen.
    - **EditText** (id/number1), element to enter the first number. Only the numeric keypad is displayed:
      android:inputType = "numberDecimal".
    - **EditText** (id/number2), element to enter the second number. Only the numeric keypad is displayed:
      android:inputType = "numberDecimal".
    - **RadigoGropu** (id/rg_group), a group of radio buttons that controls the operation to be performed by the radio buttons.
        - **RadioButton,** 4 radio buttons (rb_sumar, rb_restar, rb_multiplicar, rb_dividir)
    - **TextView** (id/result), text that shows the result of the operation.

PROGRAMMING *MainActivity.kt*

The UI design created for the activity is established:

setContentView(R.layout.activity_main)

We adjusted the margins of the View and adjusted them to the status bar and navigation bar:

configureMargins(findViewById(R.id.main))

We program the listener to perform operations when a radio button is selected:

```
val rgGroup: RadioGroup = findViewById(R.id.rg_group)
rgGroup.setOnCheckedChangeListener { _, checkedId ->
    when (checkedId) {
        R.id.rb_sumar -> {
            addition()
        }
        R.id.rb_restar -> {
            substract()
        }
        R.id.rb_multiplicar -> {
            multiplication()
        }
        R.id.rb_dividir -> {
            division()
        }
    }
}
```

The functions for operations are as follows:

SUM

```
private fun addition() {
    val textBox1: EditText = findViewById(R.id.number1)
    val textBox2: EditText = findViewById(R.id.number2)
    val resultTextView: TextView = findViewById(R.id.result)
    try {
        val number1 = textBox1.text.toString().toFloat()
        val number2 = textBox2.text.toString().toFloat()
        val result = number1 + number2
        resultTextView.text = "$result"
    } catch (e: NumberFormatException) {
        resultTextView.text = getString(R.string.number_error)
        Toast.makeText( context: this,  text: "Num Error", Toast.LENGTH_LONG).show()
    }
}
```

By means of exceptions we control that the format of the number is correct, if it is not, a Toast is displayed and an error message is displayed in the TextView of the result.

SUBTRACTION

```
private fun substract() {
    val textBox1: EditText = findViewById(R.id.number1)
    val textBox2: EditText = findViewById(R.id.number2)
    val resultTextView: TextView = findViewById(R.id.result)
    try {
        val number1 = textBox1.text.toString().toFloat()
        val number2 = textBox2.text.toString().toFloat()
        val result = number1 - number2
        resultTextView.text = "$result"
    } catch (e: NumberFormatException) {
        resultTextView.text = getString(R.string.number_error)
        Toast.makeText( context: this,  text: "Num Error", Toast.LENGTH_LONG).show()

    }
```

By means of exceptions we control that the format of the number is correct, if it is not, a Toast is displayed and an error message is displayed in the TextView of the result.

MULTIPLICATION

```
private fun multiplication() {
    val textBox1: EditText = findViewById(R.id.number1)
    val textBox2: EditText = findViewById(R.id.number2)
    val resultTextView: TextView = findViewById(R.id.result)
    try {
        val number1 = textBox1.text.toString().toFloat()
        val number2 = textBox2.text.toString().toFloat()
        val result = number1 * number2
        resultTextView.text = "$result"
    } catch (e: NumberFormatException) {
        resultTextView.text = getString(R.string.number_error)
        Toast.makeText( context: this,  text: "Num Error", Toast.LENGTH_LONG).show()

    }
}
```

By means of exceptions we control that the format of the number is correct, if it is not, a Toast is displayed and an error message is displayed in the TextView of the result.

DIVISION

In addition to checking that the number format is correct, the division by 0 is also controlled

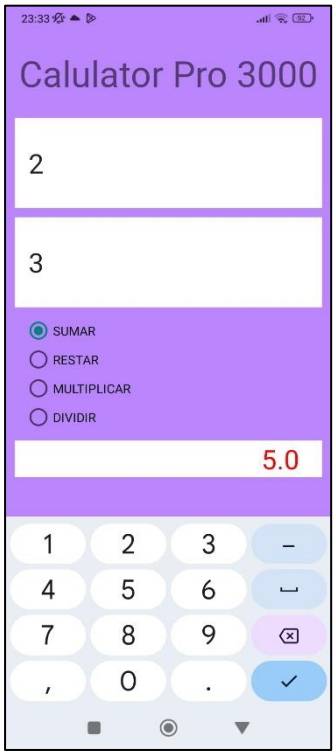
```
private fun division() {
    val textBox1: EditText = findViewById(R.id.number1)
    val textBox2: EditText = findViewById(R.id.number2)
    val resultTextView: TextView = findViewById(R.id.result)
    try {
        val number1 = textBox1.text.toString().toFloat()
        val number2 = textBox2.text.toString().toFloat()
        // Controlamos la división por 0
        if (number2 == 0f) {
            resultTextView.text = "Div. 0 Error"
            Toast.makeText( context: this,  text: "Div. 0 Error", Toast.LENGTH_LONG).show()
            return
        }
        val result = number1 / number2
        resultTextView.text = "$result"
    } catch (e: NumberFormatException) {
        resultTextView.text = getString(R.string.number_error)
        Toast.makeText( context: this,  text: "Num Error", Toast.LENGTH_LONG).show()
    }
}
```

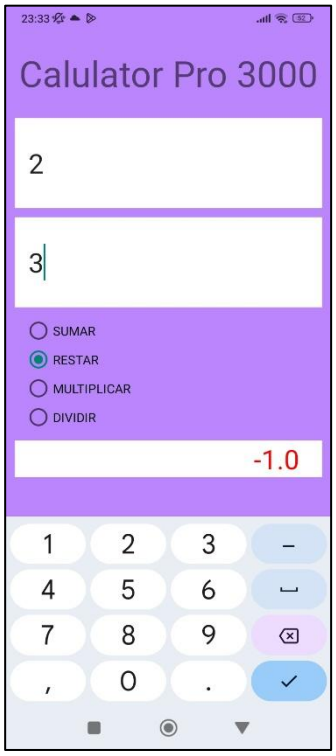
I have also added that when any EditText is selected it will deselect any radio buttons that I may have selected:

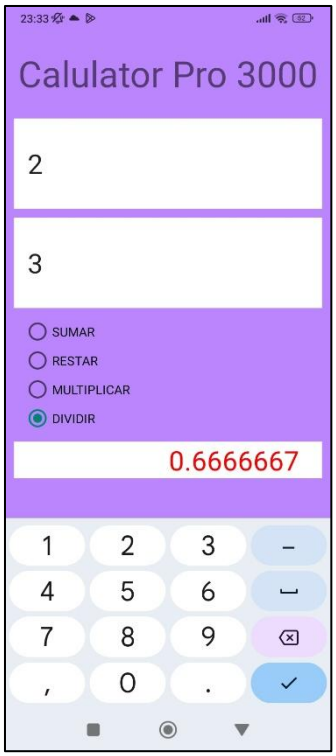
```
// Se referencian los EditText en las variables number1 y number2
val number1: EditText = findViewById(R.id.number1)
val number2: EditText = findViewById(R.id.number2)
// Definimos un listener para escuchar los cambios de foco a los EditText
val focusChangeListener = View.OnFocusChangeListener{ _, hasFocus ->
    // Si obtiene el foco
    if (hasFocus){
        // Quitamos cualquier check que pueda haber en el grupo de botones de radio
        rgGroup.clearCheck()
    }
}
// Asignamos el listener a los EditText number1 y number2
number1.onFocusChangeListener = focusChangeListener
number2.onFocusChangeListener = focusChangeListener
```

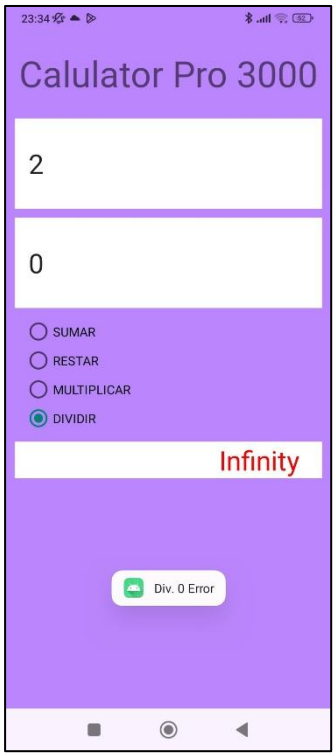
TESTS

Screenshots of the tests carried out to check the operation of the application are provided:









TASK UT03.2. ADD NUMBERS WITH PROGRAMMATIC BUTTON

DESCRIPTION

The task in this section is to implement the application of task 1 using Compose

PROGRAMMING *MainActivity.kt*

The interface is described by the created function MainScreen()

For the construction of the interface I have used the ***Scaffold class*** with the following structure:

Definition of the top bar of the application:

```
Scaffold(
    // topBar in the Scaffold
    topBar = {
        TopAppBar(
            colors = TopAppBarDefaults.topAppBarColors(
                containerColor = MaterialTheme.colorScheme.primary,
                titleContentColor = MaterialTheme.colorScheme.onPrimary
            ),
            title = {
                Text(stringResource("Suma Numeros"))
            }
        )
    },
```

The content is structured as follows:

- **Box**, the box will be the top-tier container that will contain everything.

```
// Content in the Scaffold
content = { paddingValues ->
    // Box that contain All
    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
            .padding(10.dp)
    ) {
```

- **Column**, which contains and sorts all the remaining elements in a centered column.

```
) {
    // Column in the Box,
    Column(
        modifier = Modifier
            .align(Alignment.Center)
            .matchParentSize()
            .padding(bottom = 80.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
```

- **OutlinedTextField**, the first number, using ***KeyboardOption.Number*** we make the numeric keyboard be displayed and only numbers are allowed as input.
  The change of focus to the next is controlled using ***ImeAction.Next.***

```
) {
    OutlinedTextField(
        value = numberOne,
        label = { Text( text: "Sumando 1") },
        onValueChange = { numberOne = it },
        modifier = Modifier
            .fillMaxWidth()
            .background(color = Purple80),
        colors = TextFieldDefaults.outlinedTextFieldColors(
            focusedBorderColor = Color.Transparent,
            unfocusedBorderColor = Color.Transparent,
            focusedLabelColor = Purple40,
            unfocusedLabelColor = Purple40,
        ),
        // Se muestra solo el teclado numérico
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Number,
            //Cuando pulsamos intro se pasa al siguiente campo
            imeAction = ImeAction.Next
        ),
        keyboardActions = KeyboardActions(
            // Cuando pulsamos enter ponemos el foco en el focusRequesterTwo,
            // que corresponde al siguiente OutlinedTextField
            onNext = {
                focusRequesterTwo.requestFocus()
            }
        ),
        placeholder = { Text(text = stringResource(id = R.string.edit_message_one)) }
    )
```

- **HorizontalDivider,** a horizontal divider that serves as the base of the OutlineTextField, since I've removed the borders and so it looks more aesthetic:

```
HorizontalDivider(
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 10.dp),
    thickness = 2.dp, // Grosor del borde
    color = colorResource(id=R.color.purple_500) // Color del borde
)
```

- **OutlinedTextField**, the second number, the keyboard is also controlled, this time when pressing enter we end up since it is the last number to be entered ***ImeAction.Done***.

```
OutlinedTextField(
    value = numberTwo,
    label = { Text( text: "Sumando 2") },
    onValueChange = { numberTwo = it },
    modifier = Modifier
        .fillMaxWidth()
        .background(color = Purple80)
        // Le asignamos el focusRequesterTwo
        .focusRequester(focusRequesterTwo),
    colors = TextFieldDefaults.outlinedTextFieldColors(
        focusedBorderColor = Color.Transparent,
        unfocusedBorderColor = Color.Transparent,
        focusedLabelColor = Purple40,
        unfocusedLabelColor = Purple40,
    ),
    // Se muestra solo el teclado numérico
    keyboardOptions = KeyboardOptions(
        keyboardType = KeyboardType.Number,
        // Cuando pulsamos intro se termina la entrada en el OutlinedTextField
        imeAction = ImeAction.Done
    ),
    placeholder = { Text(text = stringResource(id = R.string.edit_message_two)) }
)
```

- **HorizontalDivider,** another divider that fulfills the same function as the previous one.

- **Button**, pressing it performs the sum, the action is controlled by the onClick event.

```
Button(
    modifier = Modifier
        .height(50.dp)
        .padding(bottom = 10.dp),
    colors = ButtonDefaults.buttonColors(containerColor = Purple40),
    onClick = {
        // Result de la suma
        // Se transforman los TextFields en Int
        // se suman y se tranforman en String para que se mustren ene el resultado
        try {
            result = ((numberOne.toInt()) + (numberTwo.toInt())).toString()
        }catch (e: Exception){
            result = "Número erroneo"
        }
    },
) {
    Text(
        text = stringResource(id = R.string.bt_suma),
        fontSize = 20.sp,

        )
}
```

We can see that the onClick event transforms the numberOne and numberTwo fields into integers, sums them and stores them as estring in the textView result to display them on the screen.

Possible errors are also controlled.

- **Spacer**, a space between the button and the text that shows the final result

```
// Se añade un espacio entre el Boton y el texto
Spacer(modifier = Modifier.width(20.dp))
```

- **Text**, the text that displays the result.

```
Text(
    modifier = Modifier
        .background(Color( color: 0xFFE0E0E0))
        .fillMaxWidth()
        .height(50.dp)
        .wrapContentHeight(Alignment.CenterVertically)
        .padding(start = 40.dp),
    fontSize = 20.sp,
    textAlign = TextAlign.Left,
    text = "Resultado: " + result,
    style = MaterialTheme.typography.bodyMedium
)
```

TESTS

Screenshots of the tests carried out to check the operation of the application are provided: