Unioeste – Universidade Estadual Do Oeste do Paraná

Bacharelado de Ciências da Computação

Professor: Leonardo Medeiros

Aplicações Mobile

Eric Klaus Brenner Melo e Santos

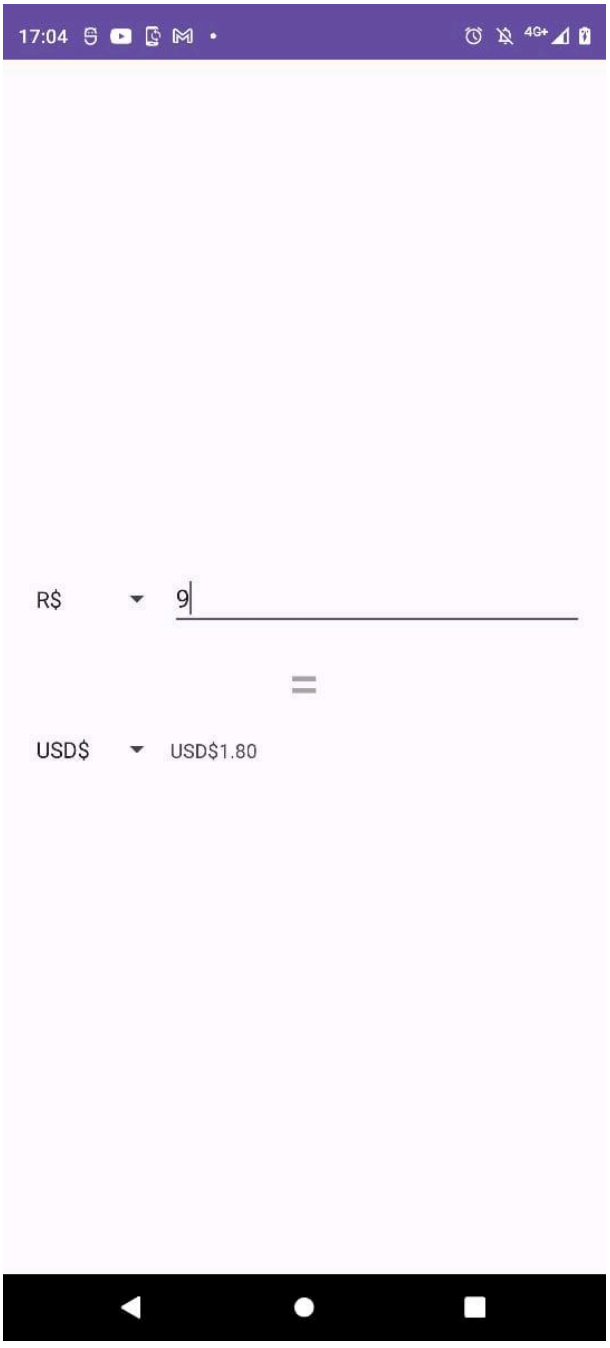
Matheus Rogério Pesarini

Ruan Rubino de Carvalho

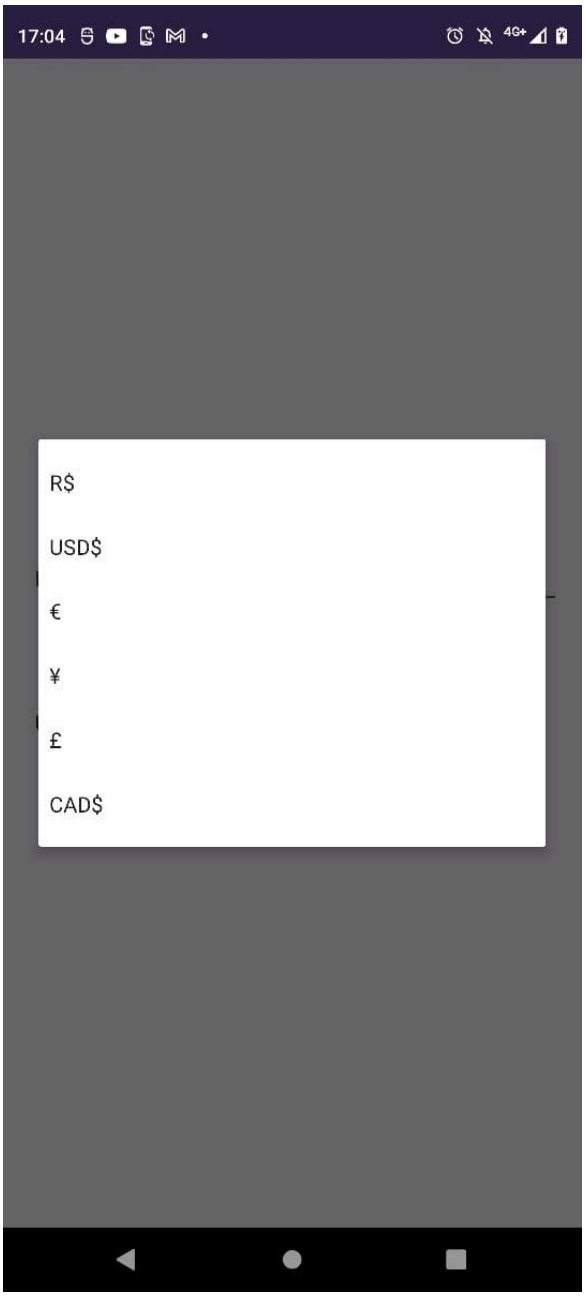
Desenvolvimento Android Studio

Com o Android Studio instalado, criei um novo projeto com o template Empty Activity, utilizando API 21 Lollipop para poder atingir uma compatibilidade de 99.6% com dispositivos Android.

Feito isso, abri o activity_main.xml e comecei a implementar a interface. A interface é extremamente simplista, possuindo apenas 1 EditText, 1 Button, 1 TextView e 2 Spinners para a seleção da moeda de conversão. Além disso, “por trás dos panos” Existem 3 LinearLayouts, em geral contendo toda a aplicação, outro contendo a parte que lida com a entrada do usuário e outro que será utilizado para a saída do valor convertido. A interface geral do projeto podem ser vistas abaixo:



O valor 9 representa o valor em reais (R$) que será convertido para o valor 1.80 em dólares (USD$). Além disso, os Spinners contém 6 moedas disponíveis para a conversão: BRL, USD, EUR, YEN, GBP, CAD.



Embaixo, mostro como foi codificado toda a parte do Activity, não utilizei a interface gráfica para a implementação, apenas o arquivo activity_main.xml.

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/container"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
```

```
android:paddingRight="16dp"
tools:context=".MainActivity"
android:orientation="vertical"
android:gravity="center">

<LinearLayout
    android:id="@+id/linear_layout_user_input"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Spinner
        android:id="@+id/currencies"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:spinnerMode="dialog" />
    <EditText
        android:id="@+id/currency_input"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"
        android:autofillHints="moneyValue"
        android:hint="@string/inputPlaceholder" />
</LinearLayout>

<Button
    android:id="@+id/convert_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:hint="@string/input_equal_output"
    android:textSize="40sp" />

<LinearLayout
    android:id="@+id/linear_layout_user_output"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Spinner
        android:id="@+id/currencies_output"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:spinnerMode="dialog" />
    <TextView
        android:id="@+id/currency_output"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content" />
```

```
  </LinearLayout>
</LinearLayout>
```

Tendo explicado como a parte visual foi construída e implementada, agora será explicado como a lógica de conversão do programa funciona. O EditText principal do programa possui um atributo id com valor “currency_input” atribuído a ele, isso faz com que nós possamos processar o dado na MainActivity. Caso o usuário tente converter sem ter nada no EditText, o programa retornará um alerta Toast pedindo para que ele insira corretamente.

Após a obtenção do valor que o usuário deseja converter, será procurado os valores atuais contidos nos Spinners (tanto o de entrada quanto o de saída). Se o usuário tentar converter de R$ para USD$ é esperado que o programa entre na função handleConversion com essas 2 strings como parâmetro, além é claro do valor que será convertido.

Feito isso, a base da função handleConversion está em um Array chamado currenciesValues que possui as relações de uma unidade monetária para outra na mesma ordem em que estão organizados os Spinners.

```
val currCurrencies: Array<String> = arrayOf("R$", "USD$", "€", "¥", "£", "CAD$")
val currenciesValues: Array<BigDecimal> = arrayOf(BigDecimal(1), BigDecimal(0.2),
BigDecimal(0.19), BigDecimal(30.12), BigDecimal(0.16), BigDecimal(0.27)
```

R$1 = USD$0.2 = €0.19 = ¥30.12 = £0.16 = CAD$0.27

Assim que o usuário tenta converter, é encontrado a posição das unidades monetárias no Array curCurrencies para depois, na mesma posição no Array currenciesValues retirar esses valores e utilizar em uma regra de 3 para a obtenção do valor convertido.

Exemplificando melhor, se o usuário deseja converter R$5 para USD$, a posição no Array é 0 e 1. Com isso, no currenciesValues as posições 0 e 1 retornam os valores 1 e 0.2. Assim, R$1 = USD$0.2 é a base que será utilizada para calcular o valor final. Como R$1 = USD$0.2 e desejamos descobrir quantos USD$ equivalem a R$5, é utilizado a relação matemática:

R$1 - USD$0.2
R$5 - USD$x

Realizando as multiplicações e divisões obtém se por fim, USD$1.00 (o valor é retornado com precisão de 2 casas).. O aplicativo é capaz de realizar todas as conversões dentro das moedas disponíveis. Feita a explicação, abaixo estará disponível a lógica mencionada acima:

```
val convertingIndex = currCurrencies.indexOf(convertingCurrency)
val convertedIndex = currCurrencies.indexOf(convertedCurrency)

if (convertingIndex != -1 && convertedIndex != -1) {
  val convertingValueReference: BigDecimal = currenciesValues[convertingIndex]
  val convertedValueReference: BigDecimal = currenciesValues[convertedIndex]

  val convertedValue: BigDecimal = convertingValue * convertedValueReference /
convertingValueReference
```

```
  val convertedValueToFixed = convertedValue.setScale(2,
RoundingMode.HALF_UP).toString()
  return convertedValueToFixed
}
```

A última feature a ser mencionada é que caso o usuário já tenha convertido algum valor e troque a unidade monetária, seja de entrada ou de saída, o valor atualizará automaticamente. Para realizá-la foi simples. Bastou adicionar ClickListeners nos dois Spinners que chamariam a função novamente após um item ser selecionado.

```
spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
  override fun onNothingSelected(p0: AdapterView<*>?) {
  }

  override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long){
    if(userCurrencyInput.text.toString() != ""){
      val output: String = spinnerOutput.selectedItem.toString() +
handleConversion(spinner.selectedItem.toString(), spinnerOutput.selectedItem.toString(),
userCurrencyInput.text.toString().toBigDecimal())
      currencyOutput.text = output.toString()
    } else {
      Toast.makeText(this@MainActivity, "Por-favor, informe algum valor no campo de
entrada para ser convertido!", Toast.LENGTH_LONG).show()
    }
  }
}

spinnerOutput.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
  override fun onNothingSelected(p0: AdapterView<*>?) {
  }

  override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long){
    if(userCurrencyInput.text.toString() != ""){
      val output: String = spinnerOutput.selectedItem.toString() +
handleConversion(spinner.selectedItem.toString(), spinnerOutput.selectedItem.toString(),
userCurrencyInput.text.toString().toBigDecimal())
      currencyOutput.text = output.toString()
    } else {
      Toast.makeText(this@MainActivity, "Por-favor, informe algum valor no campo de
entrada para ser convertido!", Toast.LENGTH_LONG).show()
    }
  }
}
```

Desenvolvimento Terminal

Para desenvolver um aplicativo android utilizando o terminal, a primeira coisa que foi necessário fazer foi baixar o JDK pelo site oficial da Oracle. Em seguida precisei rodar alguns comandos no terminal para extrair o arquivo compactado e então criar variáveis de sessão do terminal que serão úteis posteriormente para o build do aplicativo.

```
$ tar zxf jdk-8u112-linux-x64.tar.gz
$ export JAVA_HOME=${HOME}/jdk1.8.0_112
$ export PATH=${JAVA_HOME}/bin:$PATH
```

Logo em seguida, tendo como referência a documentação disponibilizada no próprio moodle, foram baixados os arquivos sdk, build_tools, android e platform_tools. Todos eles foram extraídos na raiz do usuário. E posteriormente, com exceção do sdk, movidos para a pasta android-sdk-linux.

```
$ curl -O https://dl.google.com/android/android-sdk_r24.4.1-linux.tgz
$ tar zxf android-sdk_r24.4.1-linux.tgz

$ curl -O https://dl.google.com/android/repository/build-tools_r25-linux.zip
$ unzip build-tools_r25-linux.zip
$ mkdir android-sdk-linux/build-tools $ mv android-7.1.1
android-sdk-linux/build-tools/25.0.0

$ curl -O https://dl.google.com/android/repository/android-16_r05.zip
$ unzip android-16_r05.zip
$ mv android-4.1.2 android-sdk-linux/platforms/android-16

$ curl -O https://dl.google.com/android/repository/platform-tools_r25-linux.zip
$ unzip platform-tools_r25-linux.zip -d android-sdk-linux/
```

Agora uma parte extremamente importante, que é a criação do AndroidManifest. Esse arquivo contém informações valiosas como nome do aplicativo, qual API do android é a mínima para poder rodar o aplicativo entre outras coisas.

```
$ nano AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="net.hanshq.hello"
versionCode="1"
versionName="0.1">
 <uses-sdk android:minSdkVersion="16"/>
 <application android:label="Conversor Temperatura">
 <activity android:name=".MainActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN"/>
 <category android:name="android.intent.category.LAUNCHER"/>
 </intent-filter>
```

```
 </activity>
 </application>
</manifest>
```

De forma básica, aqui foi definido o nome do pacote seguindo a documentação disponibilizada no moodle, o nome do aplicativo para “Conversor Temperatura” e o **nome da activity principal** do aplicativo.

O próximo passo do desenvolvimento foi a criação da interface visual do aplicativo por meio do **activity_main.xml**.

```
$ mkdir res
$ mkdir res/layout
$ cd res/layout
$ nano activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextTemperature"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter temperature in Celsius"
        android:inputType="numberDecimal" />

    <RadioGroup
        android:id="@+id/radioGroupConversion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <RadioButton
            android:id="@+id/radioButtonToFahrenheit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="To Fahrenheit"
            android:checked="true" />

        <RadioButton
            android:id="@+id/radioButtonToKelvin"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="To Kelvin" />
```

```
    </RadioGroup>

    <Button
        android:id="@+id/buttonConvert"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Convert" />

    <TextView
        android:id="@+id/textViewResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Result will be shown here"
        android:textSize="18sp"
        android:textStyle="bold" />

</LinearLayout>
```

Esse layout foi extremamente simples, contendo um campo de entrada onde o usuário digitará a temperatura que deseja converter, dois botões selecionáveis do tipo rádio para informar se deseja o resultado em Kelvin ou em graus Fahrenheit, o botão para converter e por fim o texto onde será exibido o valor da conversão.

Em penúltimo lugar, vem a criação do arquivo que definirá como a lógica do aplicativo funcionará, ou de forma mais simples, é aqui onde a entrada do usuário será modificada para atingir o valor desejado final.

```
$ cd ..
$ cd ..
$ mkdir java
$ mkdir java/net
$ mkdir java/net/hanshq
$ mkdir java/net/hanshq/hello
$ cd java/net/hanshq/hello
$ nano MainActivity.java

package net.hanshq.hello;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;
```

```
public class MainActivity extends Activity {
    EditText editTextTemperature;
    RadioGroup radioGroupConversion;
    Button buttonConvert;
    TextView textViewResult;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        editTextTemperature = (EditText)findViewById(R.id.editTextTemperature);
        radioGroupConversion =
(RadioGroup)findViewById(R.id.radioGroupConversion);
        buttonConvert = (Button)findViewById(R.id.buttonConvert);
        textViewResult = (TextView)findViewById(R.id.textViewResult);

        buttonConvert.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                convertTemperature();
            }
        });
    }

    private void convertTemperature(){
        String temperatureText = editTextTemperature.getText().toString();

        if(temperatureText.isEmpty()){
            Toast.makeText(getApplicationContext(), "Por favor, insira uma temperatura.",
Toast.LENGTH_SHORT).show();
            return;

        }
        double celsius = Double.parseDouble(editTextTemperature.getText().toString());
        double result;

        int radioButtonId = radioGroupConversion.getCheckedRadioButtonId();
        RadioButton radioButton = (RadioButton)findViewById(radioButtonId);

        switch(radioButton.getId()){
            case R.id.radioButtonToFahrenheit:
                result = (celsius * 9 / 5) + 32;
                textViewResult.setText(String.format("%.2f°C é %.2f°F", celsius, result));
                break;
            case R.id.radioButtonToKelvin:
                result = celsius + 273.15;
                textViewResult.setText(String.format("%.2f°C é %.2fK", celsius, result));
```

```
            break;
        default:
            break;
    }
  }
}
```

De forma sucinta, existem 4 váriaveis no código que correspondem aos 4 elementos que estão sendo exibidos na tela: o texto para o usuário digitar, o grupo de botões radio (kelvin ou fahrenheit), o botão para converter e o texto para exibir o resultado final. Foi criada uma lógica inicialmente para adicionar um evento de clique no botão da tela, para que assim quando o usuário clique a conversão seja realizada propriamente.

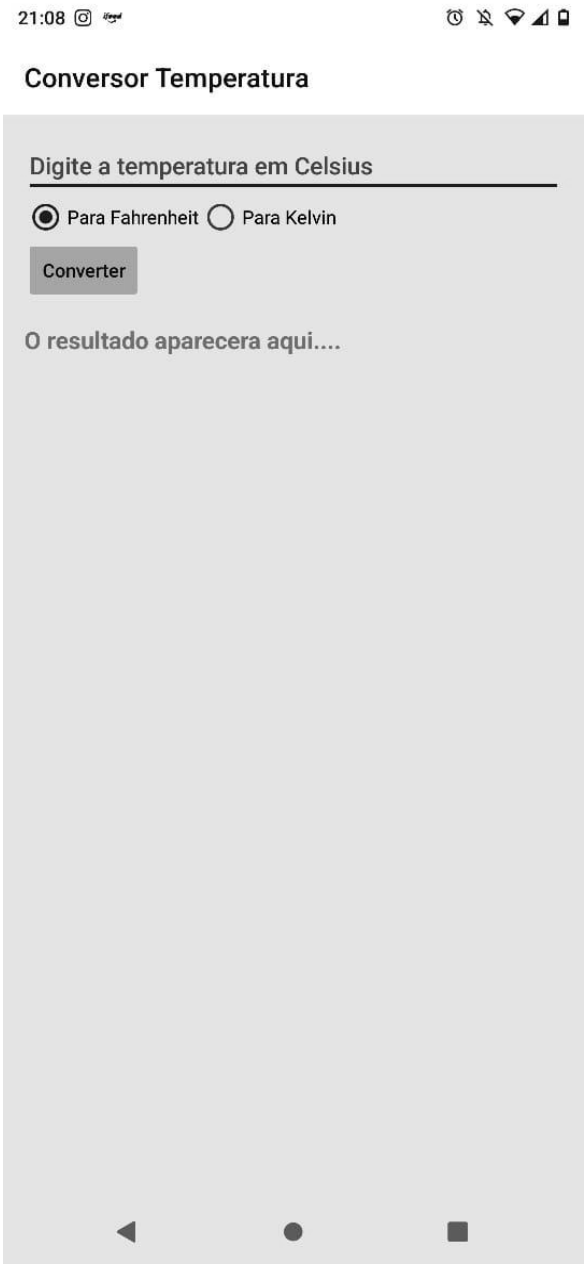
A função de conversão encontra o valor digitado pelo usuário no campo de texto, se o mesmo for vazio, então sai da função exibindo um Toast pedindo para que o usuário digite um valor válido. Caso o valor seja válido, então uma lógica para encontrar qual botão do RadioGroup está ativo é feita, com isso descobrimos se o usuário deseja uma conversão para Kelvin ou para graus Fahrenheit e aqui é feito propriamente a conversão e armazenado o valor em uma váriavel chamada result.

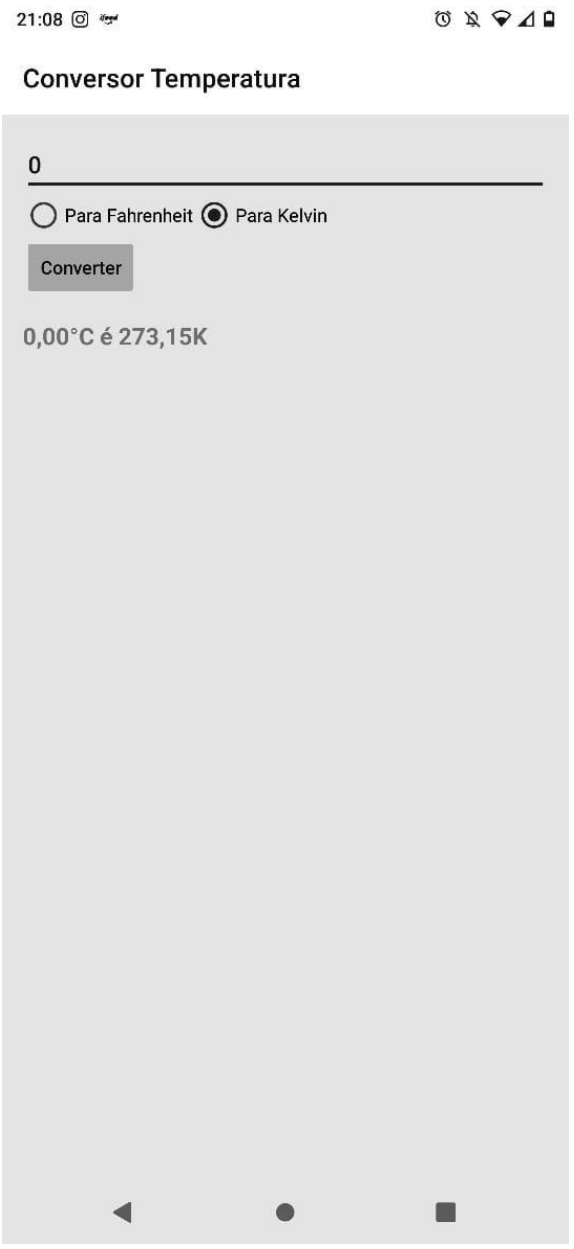
Essa váriavel result então será por fim colocada no TextView de resultado, encerrando o ciclo de vida do programa. Agora começa a última etapa do desenvolvimento que é buildar o aplicativo para poder disponibilizá-los em dispositivos celulares.

```
$ cd ..
$ cd ..
$ cd ..
$ cd ..
```

```
$ "${BUILD_TOOLS}/aapt" package -f -m -J build/gen/ -S res -M AndroidManifest.xml
-I "${PLATFORM}/android.jar"
$ javac -source 1.7 -target 1.7 -bootclasspath "${JAVA_HOME}/jre/lib/rt.jar"
-classpath "${PLATFORM}/android.jar" -d build/obj build/gen/net/hanshq/hello/R.java
java/net/hanshq/hello/MainActivity.java
$ "${BUILD_TOOLS}/dx" --dex --output=build/apk/classes.dex build/obj/
$ "${BUILD_TOOLS}/aapt" package -f -M AndroidManifest.xml -S res/  -I
"${PLATFORM}/android.jar" \ -F build/Temperatura.unsigned.apk build/apk/
$ "${BUILD_TOOLS}/zipalign" -f -p 4 build/Hello.unsigned.apk
build/Temperatura.aligned.apk
$ keytool -genkeypair -keystore keystore.jks -alias androidkey \ -validity 10000
-keyalg RSA -keysize 2048 \ -storepass android -keypass android
$ "${BUILD_TOOLS}/apksigner" sign --ks keystore.jks \ --ks-key-alias androidkey
--ks-pass pass:android \ --key-pass pass:android --out build/Temperatura.apk \
build/Temperatura.aligned.apk
```

Agora sim, o aplicativo estava pronto para uso.
Interface geral do aplicativo e exemplo de conversão para Kelvin:





Desenvolvimento Flutter + Dart

Pacotes necessários para o trabalho:

- Java SDK
- Android Studio
- Flutter + Dart SDK

Após instalado o Java SDK, no Android Studio instalei os plugins do Dart e Flutter, e baixei o Flutter pelo site oficial.
Fiz a configuração inicial do comando:

flutter doctor

Onde precisei baixar o Google Chrome e aceitar os termos com:

flutter doctor –android-license

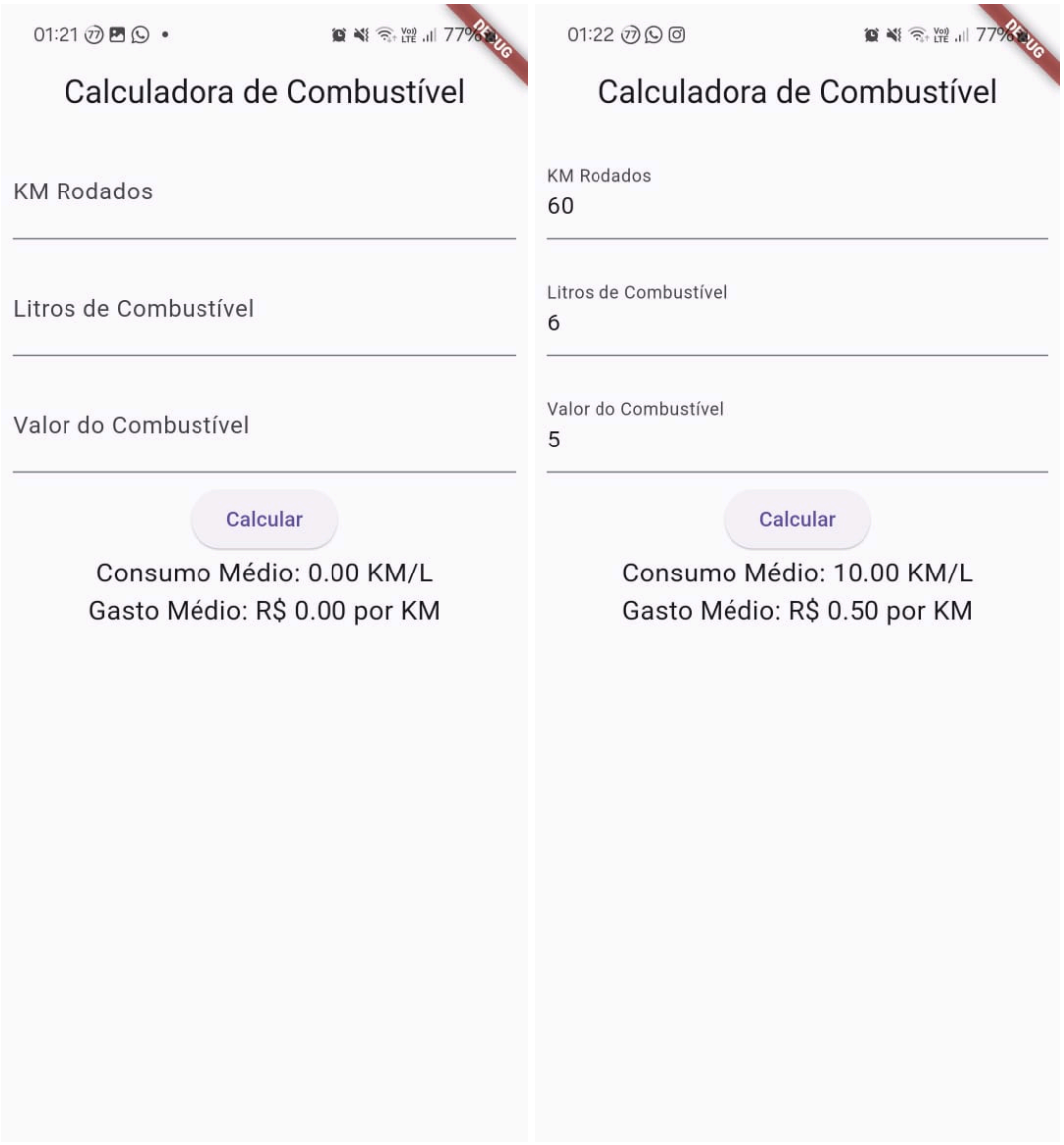
Precisei selecionar a pasta do dart-sdk dentro da pasta do flutter no Android Studio

Após isso foi criado o projeto com o comando:

flutter create tds

Utilizei meu celular para emular a aplicação.

Foi feita uma calculadora de consumo de combustível e de gasto por quilômetro rodado, onde o usuário deve inserir os quilômetros, litros de combustível e o valor, após inserido e clicar no botão de cálculo, será exibido o consumo médio e o gasto médio.



Todo o código foi feito em apenas um arquivo .dart

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

void main() {
  runApp(MaterialApp(
    home: MyApp(),
  ));  // MaterialApp
}
```

O aplicativo começa com a importação da biblioteca de design do Flutter, com a função main de ponto de entrada que chama a função runApp com o widget MaterialApp que tem MyApp como página principal.

```dart
class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

```

MyApp é um StatefulWidget, que pode trocar de estado ao longo do tempo dependendo do tipo de aplicativo e do usuário, a classe privada _MyAppState é criada para gerenciar o estado da aplicação.

```dart
class _MyAppState extends State<MyApp> {
  final _formKey = GlobalKey<FormState>();
  final _kmRodadosController = TextEditingController();
  final _litrosCombustivelController = TextEditingController();
  final _valorCombustivelController = TextEditingController();

  double _consumoMedio = 0.0;
  double _gastoMedio = 0.0;

```

Dentro de _MyAppState, várias variáveis são declaradas. _formKey é uma GlobalKey que identifica de forma única o Form widget. _kmRodadosController, _litrosCombustivelController e _valorCombustivelController são controladores de texto que gerenciam a interação entre o usuário e os campos de texto. _consumoMedio e _gastoMedio são variáveis que armazenam o consumo médio de combustível e o gasto médio por quilômetro.
Todas as variáveis são privadas dessa classe.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Calculadora de Combustível'),
      centerTitle: true,
    ),  // AppBar
    body: Form(
      key: _formKey,
      child: Column(
        children: <Widget>[
          Padding(
            padding: const EdgeInsets.all(8.0), // adiciona margem
            child: TextFormField(
              controller: _kmRodadosController,
              decoration: InputDecoration(labelText: 'KM Rodados'),
              keyboardType: TextInputType.number,
            ),  // TextFormField
          ),  // Padding
          Padding(
            padding: const EdgeInsets.all(8.0), // adiciona margem
            child: TextFormField(
              controller: _litrosCombustivelController,
              decoration: InputDecoration(labelText: 'Litros de Combustível'),
              keyboardType: TextInputType.number,
            ),  // TextFormField
          ),  // Padding
          Padding(
            padding: const EdgeInsets.all(8.0), // adiciona margem
            child: TextFormField(
              controller: _valorCombustivelController,
              decoration: InputDecoration(labelText: 'Valor do Combustível'),
              keyboardType: TextInputType.number,
            ),  // TextFormField
          ),  // Padding
```

O método build retorna um widget Scaffold, que fornece a estrutura básica visual para o aplicativo. Dentro do Scaffold, há um AppBar com o título do aplicativo e um Form que contém os três inputs e um botão.

Cada input contém um controlador, nome, estilo de teclado numérico para inserção e um padding para ficar com um espaçamento melhor.

```
58                ElevatedButton(
59                  onPressed: () {
60                    final kmRodados = double.parse(_kmRodadosController.text);
61                    final litrosCombustivel = double.parse(_litrosCombustivelController.text);
62                    final valorCombustivel = double.parse(_valorCombustivelController.text);
63                    setState(() {
64                      _consumoMedio = kmRodados / litrosCombustivel;
65                      _gastoMedio = valorCombustivel / _consumoMedio;
66                    });
67                  },
68                  child: Text('Calcular'),
69                ),  // ElevatedButton
70                Text(
71                  'Consumo Médio: ${_consumoMedio.toStringAsFixed(2)} KM/L',
72                  style: TextStyle(fontSize: 18.0),
73                ),  // Text
74                Text(
75                  'Gasto Médio: R\$ ${_gastoMedio.toStringAsFixed(2)} por KM',
76                  style: TextStyle(fontSize: 18.0),
77                ),  // Text
78              ],  // <Widget>[]
79            ),  // Column
80          ),  // Form
81        );  // Scaffold
82      }
83    }
```

Nessa parte do código se encontra o botão que ao ser pressionado será chamada a função onPressed, dentro dessa função os valores inseridos são convertido em números para serem usados no cálculo de kmRodados/litrosCombustivel e valorCombustivel/_consumoMedio, ao ser realizado esse cálculo será chamado o setState que irá atualizar somente o estado de exibição dos resultados.

Os últimos dois widgets text, o primeiro exibe o consumo médio de combustível e o segundo exibe o gasto médio por quilômetro. O método toStringAsFixed é usado para formatar os números com duas casas decimais. A fonte de texto é definida para um tamanho de 18.0.