
AGENDA 3

LAYOUT DO
PROJETO, TELA
DE SPLASH,
IMAGENS
E CORES





Vamos agora desenvolver o aplicativo “Conversor de Medidas” proposto pelo Gustavo. Esse aplicativo conta com alguns recursos básicos de customização da interface do usuário. Essas ferramentas são utilizadas para que a interface seja agradável para quem utiliza o sistema. Em um primeiro momento vamos elencar alguns pontos importantes nesse processo, começamos pelas cores utilizadas.

Dicas para o desenvolvimento da interface do usuário

A escolha das cores é fundamental para o desenvolvimento da interface do usuário. Todos nós já sabemos que a escolha e combinação de cores não deve ocorrer por acaso. É um processo que requer um estudo sobre o significado de cada cor. Você deve se lembrar das agendas de Design Digital, que estudou no módulo II! O vermelho, por exemplo, é considerado um tom quente, porque transmite energia e coragem, facilitando até mesmo uma ação do usuário. Já o verde, é considerado uma cor fria, por transmitir segurança.

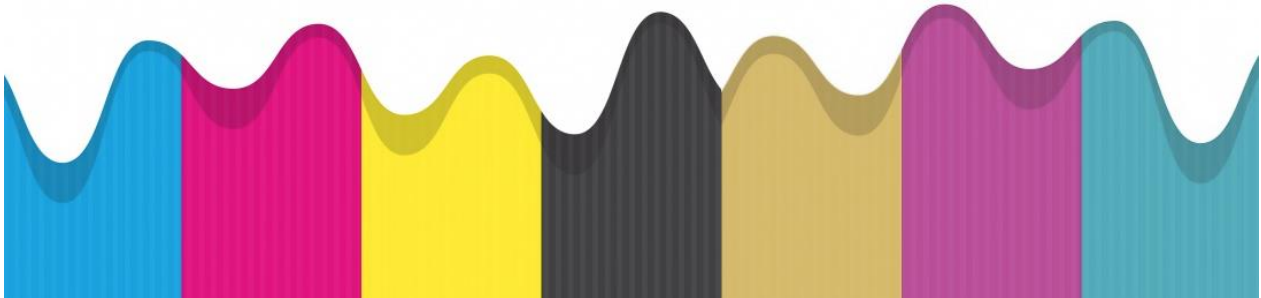


Imagem 4 - Fonte: www.freepik.com

Utilizar algumas técnicas para escolher as cores certas é fundamental para que o projeto fique harmonioso e atrativo, contudo, a validação por parte do cliente é também muito importante. Imagine o desastre que seria, caso o desenvolvedor de um aplicativo da Coca-Cola utilizasse a cor azul, da Pepsi, sua principal concorrente!

Por isso, é muito importante levar em consideração as aulas de Design Digital, estudadas no módulo I, para saber como utilizar as cores e combinações a fim de trazer harmonia para os seus aplicativos. Ao final dessa agenda você encontra um vídeo sobre cores. Vale a pena conferir!

Outro ponto importante diz respeito a utilização de botões. É fundamental destacar com cores ou movimentos os botões responsáveis por desempenhar ações importantes em seu aplicativo.

Imagine que no projeto “Conversor de Medidas” o botão para chamar o conversor de “Km para Metros” fique com um *layout* que o deixe discreto perante os demais recursos. O usuário pode pensar que aquela função está desabilitada, por isso é importante fazer com que esses botões e componentes fiquem perceptíveis. A padronização do estilo utilizado nesses botões também é fundamental. Utilize simetria nos tamanhos e alinhamento, garantindo uma organização ao seu aplicativo.

A preocupação com ícones, logos e imagens também é importante! Eles devem ser usado apenas se for necessário para que não gere poluição visual ou até mesmo uma desordem no layout do aplicativo. Cuidado com imagens e fotos, elas devem respeitar e gerar uma harmonia com as cores utilizadas.



Imagem 5 - Fonte: www.freepik.com

O acesso aos recursos do seu aplicativo deve ser fácil e intuitivo para o usuário. Imagine que para entrar na função do conversor de “Km para Metros” do app, o usuário tenha que clicar em quatro botões e deslizar duas telas. Isso se torna uma tarefa desgastante, gerando assim desestímulo para acessar as funções do aplicativo mobile.

Tela Splash

A tela de “*Splash*” é a tela de abertura de uma aplicação. Essa tela já foi muito mais explorada na programação de computadores, e atualmente é mais encontrada em aplicativos de jogos. Sua utilização no desenvolvimento mobile é garantir uma propaganda ao usuário para que ele fixe o nome do aplicativo ou até mesmo do desenvolvedor ou proprietário do sistema.

Uma outra função é que durante a exibição da tela de “*Splash*” o aplicativo pode carregar recursos para o seu funcionamento, transmitindo ao usuário sensação de que o sistema não “travou”.

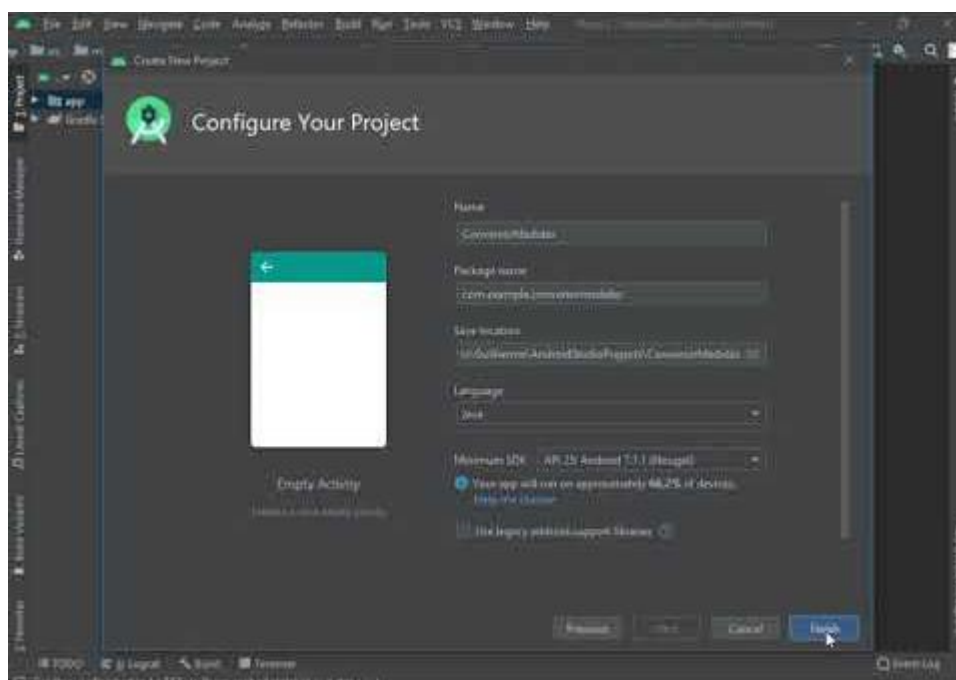
É necessário atenção na utilização deste recurso: uma tela de “*Splash*” que demore muito pode levar a impaciência do usuário. Se você já passou por alguma situação como essa deve saber como é exaustivo ter que esperar um tempo muito longo para a execução dessa tela. Por isso, é importante saber dosar o tempo de execução da tela de Splash.



Desenvolvimento da tela Splash

Chegou a hora de desenvolver o aplicativo “Conversor de Medidas”, para isso, crie no Android Studio um novo projeto com o nome de “ConversorMedidas”. Vamos utilizar a “API 25” e linguagem de programação “Java”.

Como visto anteriormente, vamos trabalhar com a inserção de mais uma Activity no projeto, com o nome de “Splash”. Acompanhe o desenvolvimento do projeto e da Activity no Vídeo 1 a seguir:



No projeto, a tela “MainActivity” é que inicia a aplicação. Portanto, vamos ter que acessar a configuração do arquivo “**AndroidManifest.xml**” para informar que a tela de “Splash” é a inicial.

Para isso, em **App > manifest > AndroidManifest.xml**, altere o código para que a tela inicial seja a tela “Splash”.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.conversormedidas">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        //----- Início da alteração do código -----
        //Activity normal
        <activity android:name=".MainActivity"></activity>
        //Activity inicial
        <activity android:name=".Splash">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            //----- Fim da alteração do código -----
        </activity>
    </application>
</manifest>
```

Inserindo imagem

Na tela inicial do projeto vamos inserir uma imagem, representando o logo do projeto. Para esse processo foi escolhido uma imagem aleatória da Internet. Essa imagem deve ser copiada para a pasta “Drawable” do projeto disponível em App > res > drawable. No Vídeo 2 você encontra o desenvolvimento dessa etapa.

Após o carregamento da imagem no projeto, vamos utilizar um componente “ImageView” para exibir a imagem na Activity como mostra na imagem a seguir:



Imagem 6 – Tela Splash.

Conforme o site de desenvolvimento do Android Studio, o `ImageView` exibe recursos de imagem, como por exemplo, o `bitmap` ou os recursos do `Drawable`. O `ImageView` também é utilizado para aplicar tonalidades a uma imagem e lidar com o dimensionamento dela. Disponível em: <https://developer.android.com/reference/android/widget/ImageView>.

Removendo a ActionBar da Activity

Para que a tela de “Splash” tenha boa aparência é necessário remover a “ActionBar” da Activity. Essa barra fica na extensão superior da Activity como mostra a imagem 4.

Para ocultar a “ActionBar” em cada Activity é necessário utilizar o comando a seguir, na programação “Java”.

```
getSupportActionBar().hide();
```

No Vídeo 2 você encontra o desenvolvimento dessa etapa, que altera a programação da classe “`Splash.java`”. Disponível em **App > java > com.example.menu > Splash**.

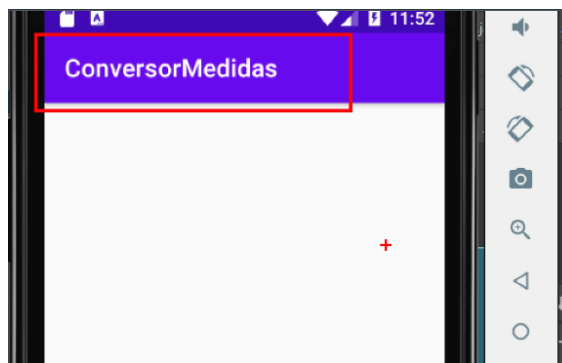


Imagem 7 – ActionBar.

Colocando a Activity em modo Tela Cheia

Para que a tela de “Splash” tenha uma boa aparência é necessário colocá-la em modo “Tela Cheia”. Desta forma, as informações do dispositivo, como: hora, bateria, e notificações são

ocultadas da tela, ficando apenas a Activity como mostra Figura 5. Para que Activity fique em modo Tela Cheia é necessário utilizar o comando a seguir, na programação “Java”.

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

No vídeo 2 você encontra o desenvolvimento dessa etapa, que altera a programação da classe “Splash.java”. Disponível em **App > java > com.example.menu > Splash**.

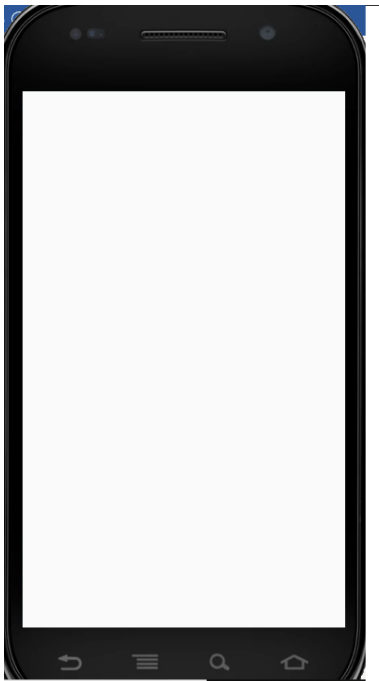
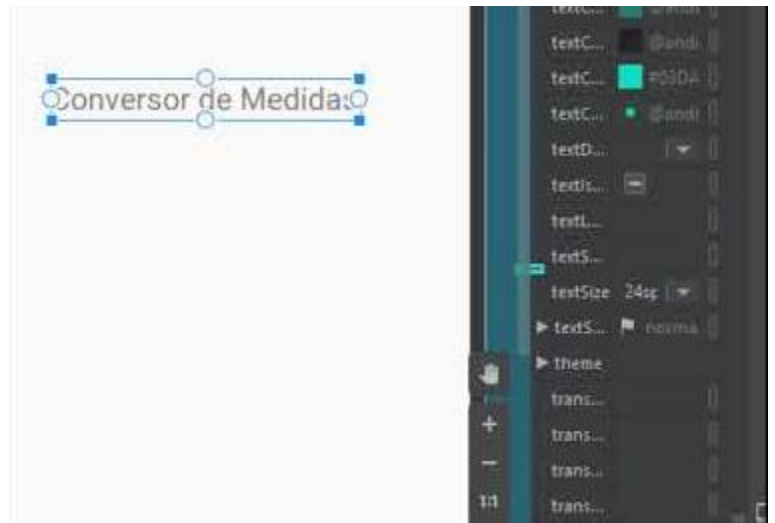


Imagem 8 – FullScreen.



Vídeo 2 do Youtube.com: <https://youtu.be/oSXA8cJvRU>

Programando a tela Splash

A programação da tela “Splash” conta com algumas novidades no processo de transição entre Activity’s. A lógica da programação é bem parecida com a utilizada no processo de “chamar” uma nova tela por meio de um botão. A diferença é que o usuário não precisa pressionar nada para que a tela principal seja “chamada”. O processo é todo automático e para esse desenvolvimento vamos utilizar a classe “Handler”.

A classe “Handler”, como a sua tradução já diz, é uma classe manipuladora que permite enviar e processar mensagens de objetos executáveis associados a uma fila. Cada instância do manipulador está associada a um único encadeamento e à fila de mensagens desse

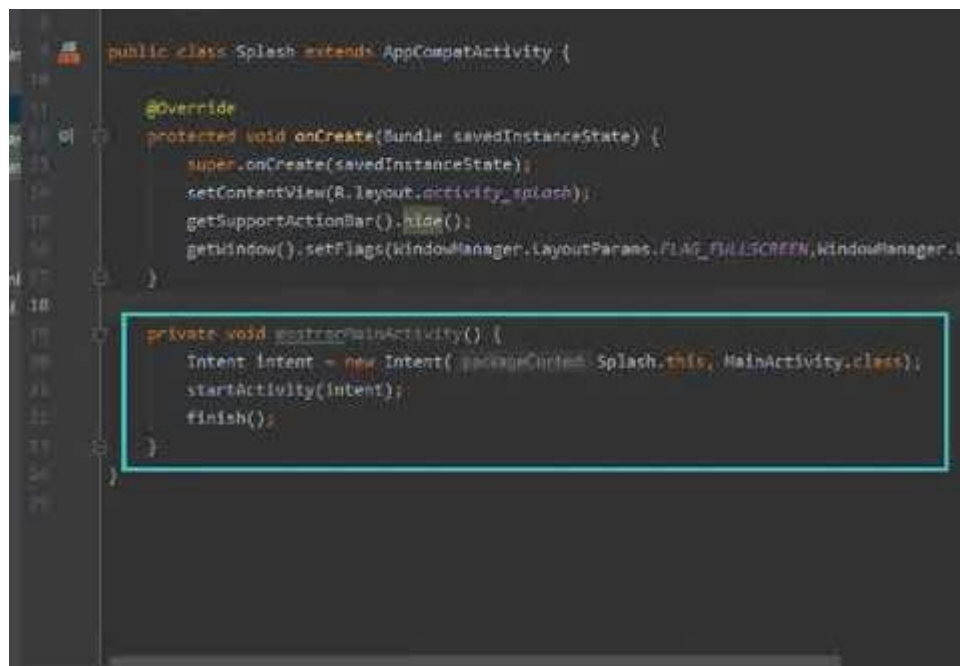
encadeamento. Quando você cria um manipulador ou “Handler”, ele é vinculado a um “Looper” que é uma fila de execuções. Esse manipulador entregará mensagens e executáveis na fila de mensagens do “Looper”, que por sua vez os executará.

Existem dois usos principais para um manipulador:

- 1 - Agendar mensagens e executáveis para serem executados em algum momento no futuro;
- 2 - Enfileirar uma ação a ser executada em um encadeamento diferente do seu.

No caso da tela “Splash”, vamos utilizar o “Handler().postDelayed()” para executar uma ação após um tempo determinado na programação.

Verifique no vídeo 3 como é desenvolvida essa programação da tela “Splash”:



```
public class Splash extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_splash);  
        getSupportActionBar().hide();  
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.L  
    }  
  
    private void extractMainActivity() {  
        Intent intent = new Intent(packageContext, MainActivity.class);  
        startActivity(intent);  
        finish();  
    }  
}
```

Vídeo 3 do Youtube.com: <https://youtu.be/RcgKLhyHGic>

Veja, a seguir, os comandos utilizados no desenvolvimento da classe “Splash.java” no Vídeo 3. Disponível em **App > java > com.example.menu > Splash**.

```
package com.example.conversormedidas;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.WindowManager;

public class Splash extends AppCompatActivity {

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

        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                mostrarMainActivity();
            }
        }, 2000);
    }

    private void mostrarMainActivity() {
        Intent intent = new Intent(Splash.this, MainActivity.class);
        startActivity(intent);
        finish();
    }
}
```

Alterando o nome do aplicativo

A imagem 9 mostra o nome completo para nosso aplicativo. Perceba que esse nome foi alterado, uma vez que o nome original vem do que foi utilizado no ato do desenvolvimento do projeto. Para alterar o nome abra o arquivo “strings.xml” disponível em **App > res > values > strings.xml**, e altere o código.

```
<resources>  
    <string name="app_name">Conversor de Medidas</string>  
</resources>
```

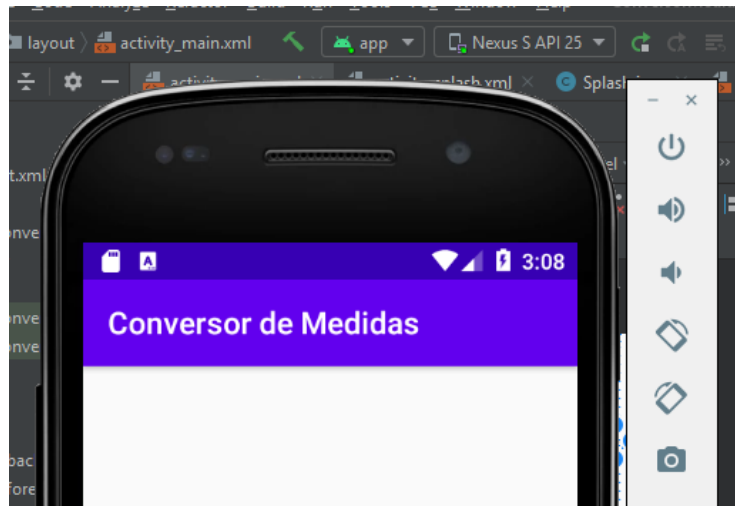


Imagem 9 – Nome do Projeto na ActionBar.

Programando a tela MainActivity

A tela “MainActivity” é o menu principal do projeto, responsável por chamar as telas posteriores de conversão. Vamos desenvolver uma nova Activity com o nome de “kmm” responsável pela conversão de medidas de quilômetros em metros.

Veja, a seguir, o código da classe “MainActivity.java”, disponível em App > java > com.example.menu > MainActivity, e na sequência, o vídeo 4, que mostra o desenvolvimento da programação.

```

package com.example.conversormedidas;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

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

        Button btnKmM_Prog = (Button)findViewById(R.id.btnKmM);

        btnKmM_Prog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, kmm.class);
                startActivity(intent);
            }
        });
    }
}

```



Vídeo 4 do Youtube.com: https://youtu.be/3rMkd9ruf_4

Programando a tela Kmm

A tela “kmm” é responsável pela conversão de medidas de quilômetros em metros. Vamos utilizar operações matemáticas já apresentadas na Agenda 1.

Veja, a seguir, o código da classe “kmm.java” disponível em App > java > com.example.menu > kmm, e na sequência, o vídeo 5, que mostra o desenvolvimento da programação.

```
package com.example.conversormedidas;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class kmm extends AppCompatActivity {

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

        final EditText edtKm_Prog = (EditText)findViewById(R.id.edtKm);
        final EditText edtM_Prog = (EditText)findViewById(R.id.edtM);
        Button btnConverter_Prog = (Button)findViewById(R.id.btnConverter);
        Button btnNovo_Prog = (Button)findViewById(R.id.btnNovo);

        btnConverter_Prog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                double km = Double.parseDouble(edtKm_Prog.getText().toString());
                double m = km*1000;
                edtM_Prog.setText(String.valueOf(m));
            }
        });

        btnNovo_Prog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                edtM_Prog.setText("");
                edtKm_Prog.setText("");
                edtKm_Prog.requestFocus();
            }
        });
    }
}
```

```

onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_kmm);

    km_Prog = (EditText)findViewById(R.id.edtKm);
    m_Prog = (EditText)findViewById(R.id.edtM);
    converter = (Button)findViewById(R.id.btnConverter);
    btnNovo = (Button)findViewById(R.id.btnNovo);

    converter.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            double km = Double.parseDouble(km_Prog.getText().toString());
            double m = km*1000;
            m_Prog.setText(String.valueOf(m));
        }
    });
}

```

Vídeo 5 do Youtube.com: https://youtu.be/rqRT3_QFj2k



Desenvolva mais três conversões para o projeto. Para isso, vamos alterar o Layout da aplicação, inserindo novos *Button's*. Atenção, é importante lembrar que cada componente possui uma ID, utilizadas no processo de programação.

Desenvolva a codificação na classe para que cada botão realize a sua respectiva ação/operação. A Figura 7 mostra um exemplo de como deve ficar o novo Layout do projeto.

Após a conclusão da tarefa, poste a sua atividade em um PDF com o código das classes utilizadas, e um *print* da AVD executando o seu projeto.

