



PROGRAMAÇÃO IV

AULA 3

CONVERSA INICIAL

Olá! Seja bem-vindo(a) a nossa aula!

Anteriormente, estudamos a linguagem Kotlin e aprendemos a sua sintaxe. Criamos algumas classes e vimos recursos modernos, como lambdas e extensões.

Nesta aula vamos aprender o básico sobre como criar o layout dos aplicativos.

O design de um aplicativo é muito importante para dar a ele aquele acabamento profissional, e muitas vezes é o que enche os olhos do usuário. Durante sua carreira como desenvolvedor Android, sempre procure se guiar pelo Material Design, o padrão de design de interfaces do Google.

Nesta aula, vamos aprender a criar layouts e estudar os componentes mais básicos e importantes. Esse será apenas o primeiro passo. Saiba que mesmo os mais experientes desenvolvedores aprendem todos os dias, portanto, é muito importante que você nunca pare de estudar, pois a tecnologia e os recursos disponíveis mudam a todo ano.

TEMA 1 – LINEARLAYOUT – BÁSICO

Como já estudamos, os arquivos de layout são escritos em XML. Pode parecer difícil no início, mas logo você se acostuma.

Por padrão, o wizard do Android Studio criou o arquivo de layout utilizando o **ConstraintLayout**. Esse gerenciador é muito utilizado e permite criar interfaces pelo editor visual apenas arrastando os componentes. Isso é algo que você vai acabar aprendendo com o dia a dia.

Para começarmos nossos estudos, é importante que você entenda os conceitos básicos sobre gerenciadores de layout e views. Podemos dizer que os gerenciadores de layouts são os caras que vão

organizar os demais componentes no layout. Veja alguns exemplos:

- **LinearLayout**: permite organizar os componentes na vertical ou horizontal.
- **FrameLayout**: permite organizar os componentes um sobre o outro, como se fossem uma pilha.
- **ConstraintLayout**: permite organizar os componentes utilizando o editor visual.

Os componentes visuais são chamados de **views**. Algumas das views mais importantes são:

- **TextView**: label com um texto.
- **EditText**: campo de texto que permite ao usuário digitar os dados.
- **Button**: botão.
- **ImageView**: imagem.
- **RecyclerView**: permite criar listas e grids.
- **CardView**: mostra a interface de um card (cartão).

Agora que você já entendeu o que é um gerenciador de layout e uma view, vamos criar o nosso primeiro exemplo. O objetivo é que você aprenda os conceitos passo a passo por meio de exemplos práticos, então vamos lá.

1.1 WRAP_CONTENT E MATCH_PARENT

Altere o código-fonte do arquivo **activity_main.xml** conforme demonstrado a seguir:

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

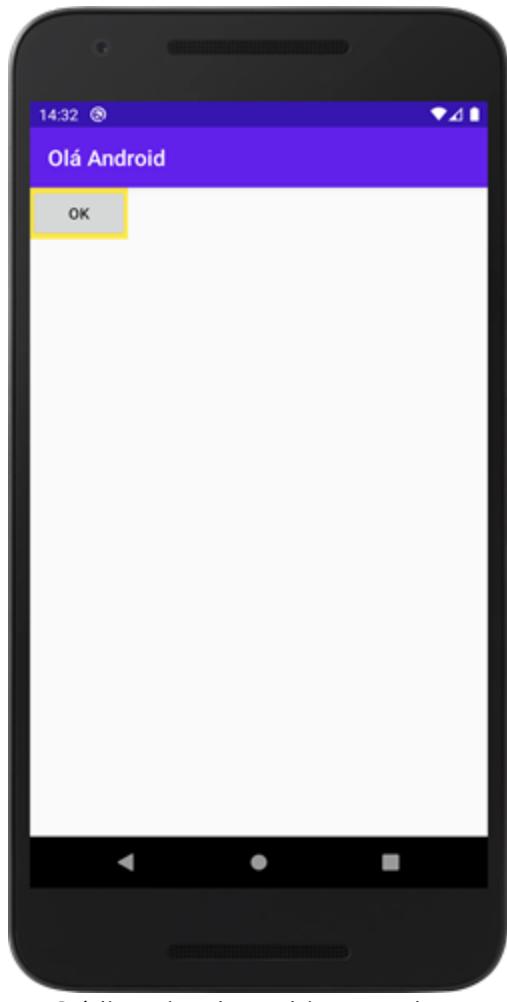
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#FFEB3B">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK" />
</LinearLayout>
```

Esse é o arquivo de layout mais simples que podemos ter. Note que alteramos a tag-raiz do XML para o **LinearLayout**, um gerenciador de layout que vai organizar as views na vertical ou horizontal.

Tente digitar o código-fonte no Android Studio e executar o exemplo no emulador – mas é muito importante assistir às práticas para pegar dicas valiosas sobre como digitar esse código rapidamente utilizando o assistente de código, além de entender explicações básicas desse layout.

Repare que, no arquivo de layout, foi definida uma cor de fundo com o atributo *android:background="#FFEB3B"*, que é a cor amarela em hexadecimal. Portanto, se executarmos o projeto no emulador, teremos o seguinte resultado.

Figura 1 – LinearLayout com wrap_content



Crédito: Ricardo Rodrigues Lecheta.

Repare na figura: a cor amarela do fundo ocupou apenas o espaço suficiente para adicionar o botão no layout. Agora vamos fazer o seguinte: vamos colocar sempre em **amarelo** a parte do código que você precisa alterar, e você vai testando conosco. Combinado?

Os atributos **layout_width** e **layout_height** são responsáveis por definir a largura e a altura de uma view. Nesse caso, altere ambos para *match_parent*, conforme demonstrado a seguir:

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:background="#FFEB3B" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK" />
</LinearLayout>
```

Para conferir o resultado, podemos executar o projeto no emulador ou clicar na guia **Design** no editor do arquivo XML para pré-visualizar a tela.

Figura 2 – LinearLayout com match_parent



Crédito: Ricardo Rodrigues Lecheta.

Com base nessas duas figuras, será que você consegue entender a diferença entre **wrap_content** e **match_parent**?

- **wrap_content**: deixa o tamanho da view (largura ou altura) somente com o espaço necessário. Geralmente utilizamos esse tipo de configuração em botões, campos de texto e imagens, pois não queremos esticá-los.
- **match_parent**: faz com que o tamanho da view ocupe todo o espaço disponível na tela ou no seu layout-pai, o que leva a view a dar aquela esticada. Geralmente adicionamos essa configuração nos gerenciadores de layout.
- **valor numérico em dp**: também podemos especificar um valor em dp (density independent pixel), mas isso é mais avançado, portanto, abordaremos esse assunto em outros exemplos. Sempre que possível, defina o tamanho das views combinando wrap_content com match_parent.

Saiba mais

Observação: nos próximos exemplos, lembre-se de alterar apenas a parte em amarelo e ir conferindo o resultado no emulador ou na pré-visualização do Android Studio.

1.2 ORIENTAÇÃO (HORIZONTAL OU VERTICAL)

Por padrão, o **LinearLayout** deixa as views organizadas na horizontal, ou seja, elas ficarão uma ao lado da outra. Para exemplificar, remova a cor de fundo que colocamos e adicione um segundo botão, conforme este código:

```
<?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" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 2" />
</LinearLayout>
```

O resultado é este:

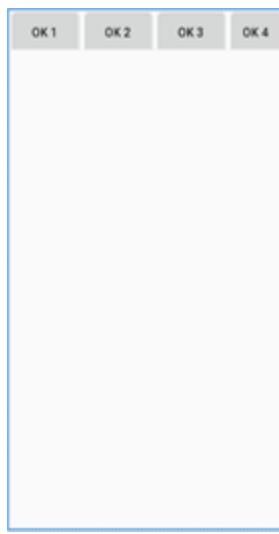
Figura 3 – Botões na horizontal



Crédito: Ricardo Rodrigues Lecheta.

Atenção: se houver muitas views uma ao lado da outra na horizontal, pode não existir espaço suficiente na tela para todas elas. Por exemplo, se você aumentar para cinco ou seis botões nesse mesmo exemplo, o resultado será o demonstrado a seguir. É claro que tudo depende do tamanho da tela do celular e da resolução.

Figura 4 – Botões na horizontal (cortando a tela)



Crédito: Ricardo Rodrigues Lecheta.

Na figura, onde está o quinto botão? Ele está voando fora da tela. Sempre tome cuidado quando a orientação do layout for horizontal. Agora vamos configurar o **LinearLayout** para a orientação vertical, o que é feito com o atributo **android:orientation="vertical"**:

```
<?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" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 2" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 3" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 4" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 5" />
</LinearLayout>
```

Resultado:

Figura 5 – Botões na vertical



Crédito: Ricardo Rodrigues Lecheta.

1.3 GRAVITY

O atributo **gravity** permite posicionar o conteúdo de uma view no centro, na esquerda, na direita etc. No próximo exemplo, adicione o atributo **android:gravity="center"** no LinearLayout. Obs: deixamos apenas dois botões na tela para diminuir a quantidade de código dos exemplos.

```
<?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:gravity="center" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK 2" />
</LinearLayout>
```

Resultado: podemos ver que o atributo **android:gravity="center"** deixou o conteúdo do layout no centro. Experimente brincar um pouco e altere as propriedades do gravity para **center|left** ou **center|end**. Tente também deixar o layout lá em baixo, alterando para **bottom**, **bottom|center** ou **bottom|end**. Uma dica é sempre utilizar o assistente do Android Studio para que ele auxilie na digitação: basta digitar **Ctrl+Espaço** no ponto desejado do código no editor.

Figura 6 – Propriedade gravity



Crédito: Ricardo Rodrigues Lecheta.

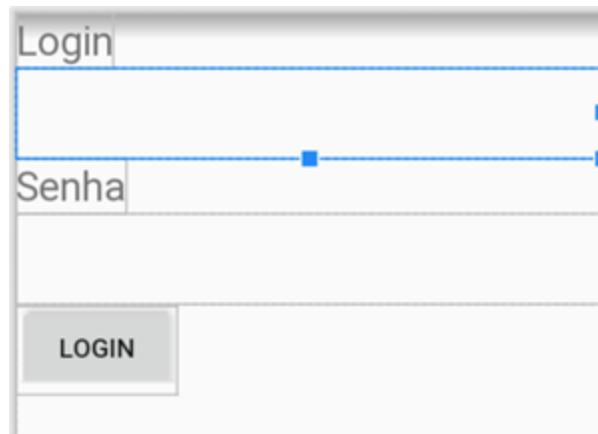
TEMA 2 – LINEARLAYOUT – FORMULÁRIO DE LOGIN

Vamos tentar juntar um pouco dos conceitos que aprendemos e criar um formulário de login. O objetivo é criar um layout simples com os campos de Login e Senha um embaixo do outro e, por último, o botão de Login. No layout, observe que o **TextView** é um label e o **EditText** é o campo de texto. Observe que nos dois campos de texto foi utilizado o atributo *layout_width="match_parent"* para que ele estique na horizontal (largura), ocupando todo o espaço disponível na tela.

```
<?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" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Senha" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login" />
</LinearLayout>
```

Resultado na pré-visualização do editor:

Figura 7 – Layout de um formulário de login



Crédito: Ricardo Rodrigues Lecheta.

Observe que o segundo campo **EditText** é a senha, portanto utilizamos o atributo **android:inputType="textPassword"** para ligar os asteriscos (***) . Para conferir o resultado do formulário, recomendo executar o exemplo no emulador.

Figura 8 – Formulário de login no emulador



Crédito: Ricardo Rodrigues Lecheta.

Saiba mais

O atributo **android:gravity="center"** foi removido do **LinearLayout**, mas, se você colocar o formulário, ele ficará centralizado.

2.1 PADDING

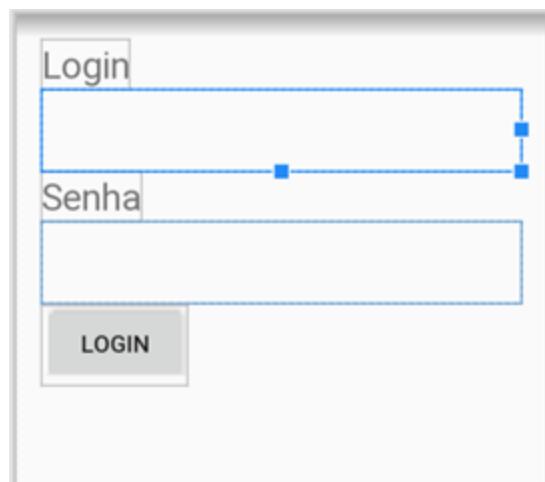
No exemplo anterior, o layout do formulário está colado nas margens, e isso não é muito bom em termos de design. Para adicionar um espaçamento dentro do formulário, vamos adicionar a propriedade **android:padding="16dp"** no **LinearLayout**.

```
<?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" >
    <!-- Mesmo código aqui -->
</LinearLayout>
```

Sempre que adicionarmos números ao layout para especificar tamanho de margens e espaçamentos, devemos utilizar a notação **dp** (density independent pixels), por isso colocamos o valor "16dp" no padding. Vale lembrar também que 16dp é o padrão de espaçamento do **Material Design** e que o Google recomenda esse espaçamento em todas as telas.

Podemos ver o resultado com padding na figura a seguir.

Figura 9 – Propriedade padding



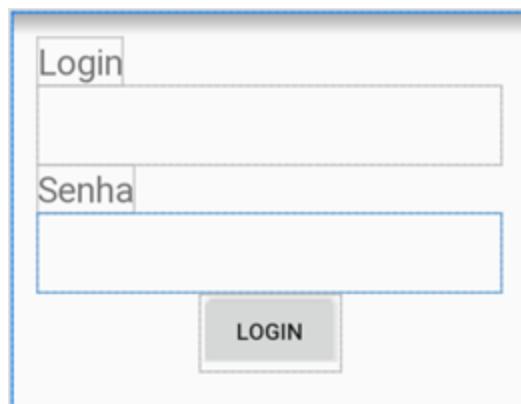
2.2 LAYOUT_GRAVITY

Seguindo nosso exemplo, como fazemos para centralizar o botão no formulário? Vimos que a propriedade **gravity** pode ser aplicada ao layout-pai (LinearLayout), mas isso faria com que todo o layout ficasse no centro, e o que queremos é que apenas o botão fique centralizado. Quando não podemos configurar o alinhamento no layout-pai, podemos utilizar a propriedade **layout_gravity** diretamente na view-filha. Para testar, adicione a propriedade **layout_gravity="center"** no botão.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . .>
    <!-- . . . -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login"
        android:layout_gravity="center"/>
</LinearLayout>
```

O resultado é que apenas o botão ficará centralizado:

Figura 10 – Propriedade layout_gravity



Crédito: Ricardo Rodrigues Lecheta.

Saiba mais

A propriedade **gravity** é sempre aplicada no layout-pai. Ela faz com que todas as views-filhas fiquem alinhadas daquela maneira. A propriedade **layout_gravity** deve ser utilizada nas views-filhas para configurar seu posicionamento relativo ao layout-pai.

2.3 LAYOUTS ANINHADOS

Você já deve ter visto em diversos aplicativos que, logo abaixo do botão de login, muitas vezes temos alguns botões ou links para as opções de "Esqueci a Senha" e "Cadastre-se", frequentemente se apresentando lado a lado. Pois muito bem, vamos tentar fazer esse layout agora. Por enquanto, deixaremos esses dois links como simples textos, então vamos utilizar o **TextView**. Como queremos deixar as duas views lado a lado, podemos colocá-las dentro de outro LinearLayout com a orientação horizontal. Vamos tentar? Para ajudar você a conferir o código, estamos colocando todo o layout XML do exemplo:

```
<?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"
    >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Senha" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login"
        android:layout_gravity="center"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cancelar" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="OK" />
    </LinearLayout>

```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Esqueci a senha"
        android:textColor="#0000ff" />

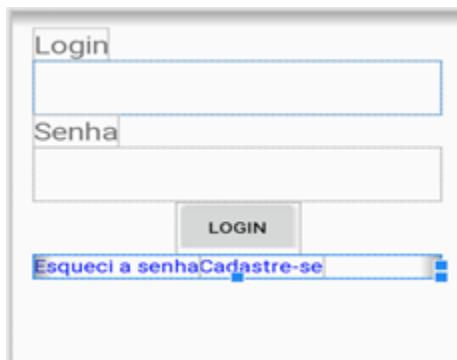
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cadastre-se"
        android:textColor="#0000ff" />
</LinearLayout>
</LinearLayout>

```

Veja que podemos colocar gerenciadores de layout dentro dos outros, ou seja, temos o LinearLayout vertical raiz (que é chamado de *layout root*) e agora temos esse LinearLayout filho.

O resultado podemos visualizar na próxima figura. Observe que o novo LinearLayout foi feito na horizontal, assim, os dois textos ficaram lado a lado.

Figura 11 – Layouts aninhados – horizontal



Crédito: Ricardo Rodrigues Lecheta.

Esperamos que tenha conseguido entender o exemplo. Se precisar, revise com calma todas as linhas de código até entender. Não prossiga sem que tenha entendido tudo.

2.4 MARGING

O atributo **marging** é muito parecido com o **padding**, a diferença está no fato de que o padding é adicionado no layout-pai e o margin é adicionado nas views ou layouts-filhos.

No layout que criamos, você percebe que os labels de "Esqueci a Senha" e "Cadastre-se" estão colados no botão de Login? Para dar um espaçamento entre eles, podemos definir uma margem entre essas duas partes do layout. Encontre o LinearLayout horizontal que acabamos de criar e adicione o atributo **layout_marginTop**, conforme demonstrado a seguir:

```
<LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:orientation="horizontal"  
    android:layout_marginTop="16dp" >  
  
    <!-- TextView esqueci a senha e cadastre-se aqui →  
  
    </LinearLayout>
```

O resultado pode ser visto na próxima figura:

Figura 12 – Definindo uma margem no topo



Crédito: Ricardo Rodrigues Lecheta.

TEMA 3 – LINEARLAYOUT – AVANÇADO

Nesta seção, vamos continuar estudando algumas das propriedades mais avançadas do LinearLayout.

3.1 CONTROLANDO O PESO COM LAYOUT_WEIGHT

Conforme vimos no exercício anterior, conseguimos deixar os dois textos ("Esqueci a Senha" e "Cadastre-se") lado a lado, porém, o ideal é que ficassem centralizados na tela e ocupando o mesmo espaço. Eles ficaram colados um ao outro e isso está ruim em termos de layout. Uma maneira de descolar as duas views seria adicionar uma margem à esquerda do TextView "Cadastre-se", assim: `android:layout_marginLeft="24dp"`.

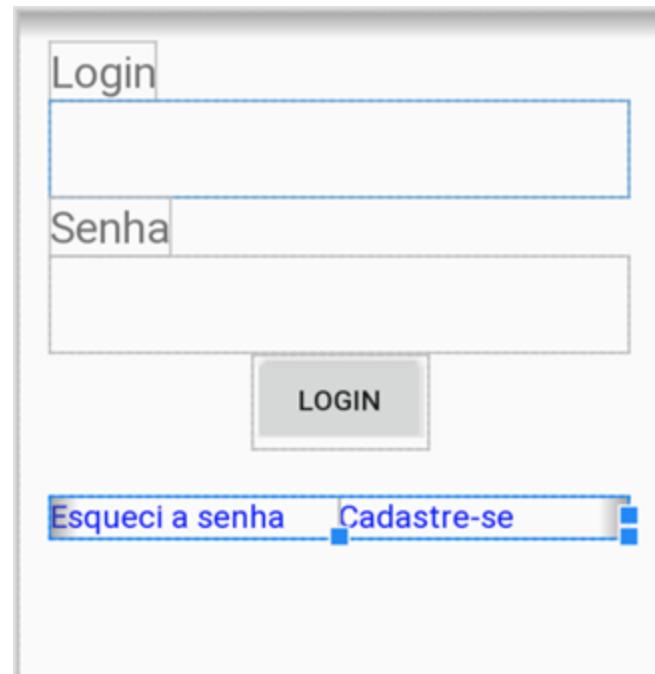
Embora essa opção talvez funcione, ela não é perfeita, pois é difícil centralizar exatamente as views na tela para que ambas ocupem o mesmo espaço. O ideal seria que tivéssemos uma maneira de falar que cada TextView deve ocupar 50% do espaço disponível. Felizmente podemos fazer isso com o controle de pesos.

Tecnicamente, muitas linguagens utilizam o termo *peso* para informar que vamos ter algum tipo de porcentagem ao calcular o tamanho. No Android, o controle de pesos é feito com o atributo **layout_weight**. O peso recebe um número e pode ter qualquer valor. Ainda há algo importante que você precisa entender: sempre que o peso de uma view for igual ao de outra, elas terão o mesmo tamanho. Nesse caso, como queremos que ambos os textos ("Esqueci a Senha" e "Cadastre-se") tenham a mesma largura, basta aplicar o **peso=1** em ambos. A seguir, temos o código destacado que orienta a como fazer isso. Note que também é necessário alterar a largura para ficar com tamanho **layout_width="0dp"** (zero), assim ela pode respeitar o peso adicionado **layout_weight="1"**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
<! -- Mesmo código aqui -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="16dp">
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Esqueci a senha"
        android:textColor="#0000ff"
        android:layout_weight="1"/>
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Cadastre-se"
        android:textColor="#0000ff"
        android:layout_weight="1"/>
</LinearLayout>
</LinearLayout>
```

Veja que interessante o resultado! Dessa vez, ambos os textos ("Esqueci a Senha" e "Cadastre-se") têm a mesma largura, pois respeitaram o peso=1.

Figura 13 – Controlando pesos com a propriedade layout_weight



Crédito: Ricardo Rodrigues Lecheta.

3.2 CENTRALIZANDO O TEXTO COM GRAVITY

O layout está quase bom, mas os textos "Esqueci a Senha" e "Cadastre-se" estão alinhados à esquerda e não centralizados. Contudo, já aprendemos que podemos configurar o posicionamento de uma view ou um layout com o **gravity**, lembra-se? Quando estudamos o gravity pela primeira vez, demos o exemplo de configurar **gravity="center"** no LinearLayout, assim, todo o seu conteúdo (as views do formulário) ficou centralizado. O gravity refere-se ao conteúdo de uma view! Seguindo o mesmo raciocínio, se adicionarmos o **gravity="center"** no TextView, o conteúdo ficará centralizado.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <! -- Mesmo código aqui -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Esqueci a senha"
            android:textColor="#0000ff"
            android:layout_weight="1"
            android:gravity="center"/>
        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Cadastre-se"
            android:textColor="#0000ff"
            android:layout_weight="1"
            android:gravity="center"/>
    </LinearLayout>
</LinearLayout>
```

Podemos visualizar o resultado final do formulário a seguir. Note que os dois textos abaixo do botão têm a mesma largura, pois têm pesos iguais (`layout_weight=1`) e também estão com o texto centralizado (`gravity="center"`).

Figura 14 – Propriedade gravity centralizado



Crédito: Ricardo Rodrigues Lecheta.

TEMA 4 – FRAMELAYOUT

Outro gerenciador de layout muito famoso é o FrameLayout. Ele permite organizar as views uma sobre a outra, como se fossem uma pilha. Literalmente falando, você pode imaginar uma pilha de tijolos, na qual cada tijolo é uma view ou gerenciador de layout. Sempre a view visível no aplicativo é aquela que vai estar no topo da pilha.

Exemplo prático do FrameLayout: adicionar uma imagem de fundo em algum layout. Vamos tentar adicionar uma imagem de fundo no formulário que fizemos?

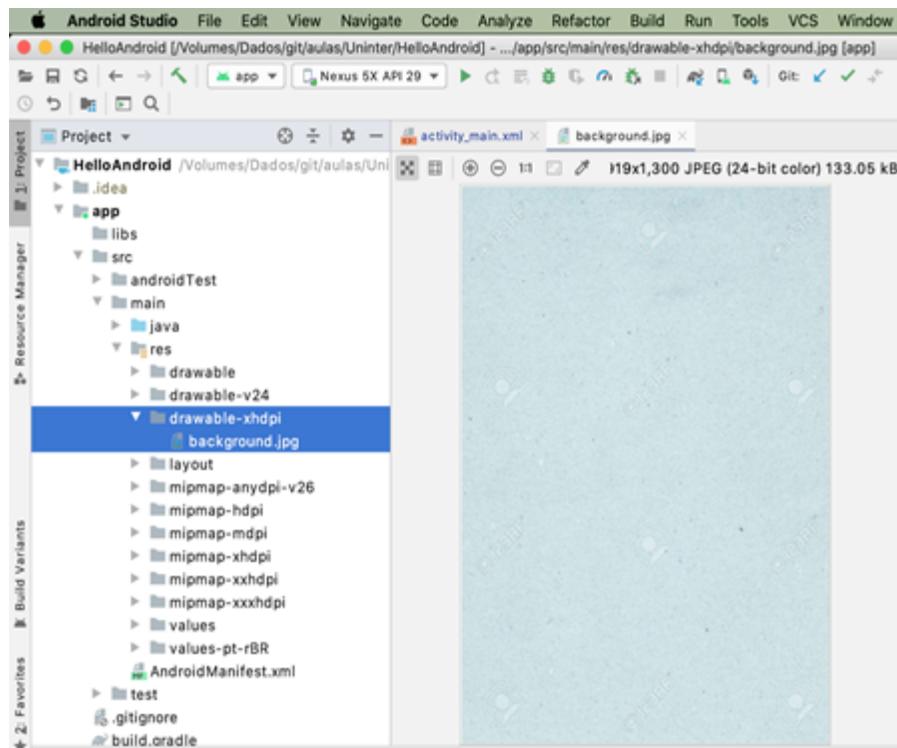
Como deixamos o texto do formulário com a cor preta, vamos procurar no Google alguma imagem com tons mais claros. Lembrando que cor e tamanho dos textos podem ser definidos com as propriedades **textSize** e **textColor** – o exemplo de código mostra como escrever um texto com fonte 22 e cor branca.

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Login"  
    android:textSize="22sp"  
    android:textColor="#ffffff" />
```

Vamos lá. Abra o site do Google Images e digite algo como "imagem vertical azul-claro" ou, em inglês, "*light blue vertical image*" (geralmente podemos encontrar mais resultados em inglês). Crie uma

pasta **/res/drawable-xhdpi**, para imagens de alta resolução, e faça o download para ela, conforme Figura 15.

Figura 15 – Imagem de fundo para o formulário



Fonte: Android Studio.

Não se preocupe agora com as pastas **/res/drawable**, **/res/drawable-xhdpi** e todas as suas variações. Geralmente as empresas têm designers, os quais vão lhe entregar as imagens e informar as pastas corretas. Em linhas gerais, podemos dizer que a pasta *xhdpi* tem imagens de alta definição. Sendo assim, a resolução ficará boa na maioria dos dispositivos. Em dispositivos com menos resolução, o Android vai redimensionar a figura para ocupar o tamanho necessário na tela. Isso tudo é transparente para o desenvolvedor.

Depois de fazer o download da figura que vamos usar como imagem de fundo para a tela de login, vamos continuar nosso exemplo. Atualize o código do arquivo de layout conforme demonstrado a seguir.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"

xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- Layout com imagem de Fundo -->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/background"
        android:scaleType="fitXY" />

    <!-- Layout do formulário -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Login"
            android:textSize="22sp"
            android:textColor="#ffffffff" />

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Senha" />
        <EditText
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
    />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login"
        android:layout_gravity="center"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Esqueci a senha"
            android:textSize="16sp"
            android:textColor="#0000ff"
            android:layout_weight="1"
            android:gravity="center"/>

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Cadastre-se"
            android:textSize="16sp"
            android:textColor="#0000ff"
            android:layout_weight="1"
            android:gravity="center"/>
    
```

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

Destacamos no layout XML a parte nova do código. Veja que o **FrameLayout** agora é a tag-raiz do XML. A tag-raiz do layout sempre precisa ter a declaração do atributo **xmlns:android**, pois esse é o

XML Schema desse arquivo e define o template de toda as tags.

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

Dentro do **FrameLayout** deixamos dois filhos. Primeiro foi adicionado o **ImageView** com a imagem de fundo. O atributo **scaleType="fitXY"** está fazendo com que essa imagem estique para ocupar o tamanho todo da tela.

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/background"  
    android:scaleType="fitXY" />
```

Logo depois do **ImageView**, foi adicionado o mesmo **LinearLayout** que já tínhamos criado para o formulário. Lembra-se de quando compararamos o **FrameLayout** com a pilha de tijolos? Nesse caso, o **ImageView** está na base da pilha e o layout do formulário está no topo. Essa pilha pode ter quantos tijolos forem necessários. O resultado do layout pode ser visualizado na figura a seguir.

Figura 16 – FrameLayout



Crédito: Ricardo Rodrigues Lecheta.

4.1 FRAMELAYOUT E PROGRESSBAR

Outra forma clássica de usar o FrameLayout é com um **ProgressBar** (aquele bolinha com uma animação que fica girando enquanto o aplicativo está executando alguma tarefa, como buscando dados na internet). O ProgressBar geralmente é adicionado no topo da pilha e fica fazendo a animação sobre um componente. Ele pode ser adicionado por cima de um formulário, ou de um botão, ou por cima de uma lista (que ainda vamos estudar). Vamos fazer o exemplo do botão? Encontre o botão no código, que deve ser assim:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Login"  
    android:layout_gravity="center" />
```

E altere para utilizar este código:

```
<FrameLayout  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center" >  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Login" />  
    <ProgressBar  
        android:layout_width="36dp"  
        android:layout_height="36dp"  
        android:layout_gravity="center"  
        android:translationZ="2dp"  
        android:visibility="visible"/>  
    </FrameLayout>
```

Veja que o botão foi envolvido com um FrameLayout. Como ele é a primeira view, está na base daquela pilha de tijolos. O ProgressBar está no topo da pilha, portanto, estará visível. Mas o ProgressBar é pequeno e foi definido com um tamanho de 36dp, então, vai ficar centralizado no centro do botão.

Resultado:

Figura 17 – FrameLayout e ProgressBar



Crédito: Ricardo Rodrigues Lecheta.

Repare que, no código, o android:layout_gravity="center" foi utilizado no FrameLayout e no ProgressBar para deixar o conteúdo centralizado.

Para visualizar a animação do ProgressBar, execute o código no emulador.

TEMA 5 – CRIANDO AS ACTIVITIES PARA OS NOVOS LAYOUTS

Já criamos um formulário para a tela de login. Agora, precisamos fazer outras telas do aplicativo, como a tela “Esqueci a senha”, “Cadastro” e até a Home do aplicativo depois de logar com sucesso. Lembra-se das primeiras aulas, quando foi explicado que cada tela do aplicativo é definida por uma activity e que cada activity tem a dupla de Classe + layout XML?

Muito bem, chegou o momento de criarmos mais três activities para praticar os conceitos. Clique com o botão direito do mouse no pacote em que está a MainActivity e utilize o seguinte menu para criar uma nova activity: > New > Activity > EmptyActivity. Repita esse processo três vezes; no nome da classe da activity, digite:

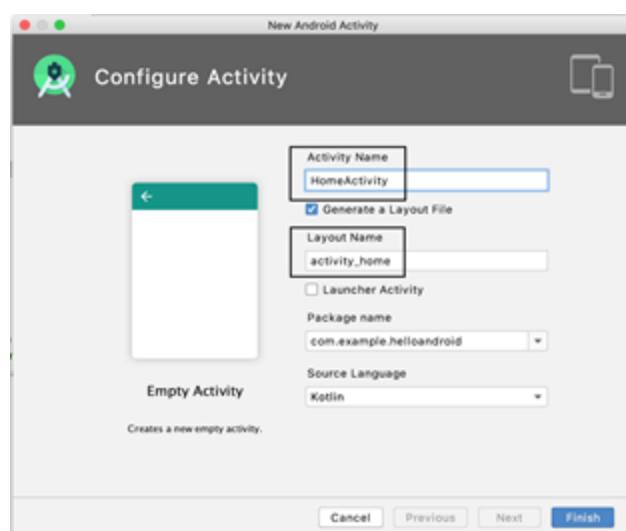
1. EsqueciSenhaActivity

2. CadastroActivity

3. HomeActivity

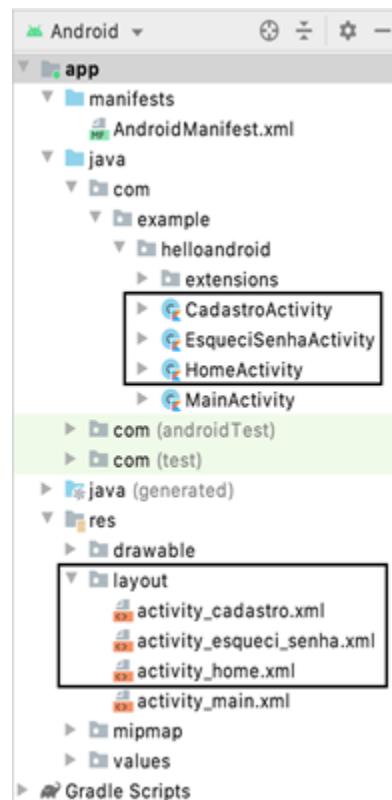
Aceite todas as outras sugestões do wizard e clique em Finish, conforme demonstrado na próxima figura. Observe que, ao digitar HomeActivity no nome da activity, o próprio wizard vai preencher automaticamente "activity_home" como sugestão do nome do arquivo de layout XML. Nunca altere isso. Deixe o wizard digitar por você.

Figura 18 – Wizard para criar uma nova activity



Depois de criar essas três activities, podemos ver que foram criadas as duplas com a classe da Activity e seu layout XML.

Figura 19 – Novas activities criadas no projeto



Conforme também explicamos em momento anterior, todas as activities precisam estar declaradas no arquivo de manifesto. Como criamos as activities com o wizard, essa configuração já foi feita automaticamente. Ao abrir o arquivo `AndroidManifest.xml`, podemos verificar que, dentro da tag `<application>`, foram adicionadas essas três activities. Essa configuração é obrigatória no Android.

```

<activity android:name=".CadastroActivity" />
<activity android:name=".EsqueciSenhaActivity" />
<activity android:name=".HomeActivity" />

```

Saiba mais

Também é possível criar a classe, o layout XML e editar o arquivo de manifesto manualmente. A vantagem de utilizar o wizard para criar a activity é que ele já faz tudo isso.

5.1 EXERCÍCIOS DE LAYOUT

Nas próximas aulas, vamos estudar como fazer a navegação de telas dentro do aplicativo, por exemplo, abrir a **HomeActivity** depois de clicar no botão de Login ou abrir a tela de "Esqueci a senha".

Para concluir esta aula, vamos fazer alguns exercícios e tentar construir alguns layouts. Seria interessante que você tentasse fazer os layouts digitando o código sozinho, pois isso vai maximizar o seu aprendizado.

5.2 LAYOUT DA TELA DE "ESQUECI A SENHA"

Vamos para o primeiro exercício.

Abra o arquivo /res/layout/activity_esqueci_senha.xml e tente fazer o seguinte layout:

Figura 20 – Layout da tela de esqueci a senha



Crédito: Ricardo Rodrigues Lecheta.

Conseguiu fazer o exercício? Utilize o código a seguir para conferir:

```
<?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">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Digite seu login" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Enviar" />
</LinearLayout>
```

5.3 LAYOUT DA TELA DE CADASTRO

Para continuar o exercício, abra o arquivo /res/layout/activity_cadastro.xml e tente fazer o layout do formulário de cadastro. Veja que adicionamos algumas views novas, como o Radio Button, para criar o campo *sexo*, e o Checkbox, muito comum para aceitar os termos de uso.

Para fins didáticos e para facilitar o exercício, colocaremos poucos campos no formulário. Naturalmente, dependendo do aplicativo, o cadastro pode ser mais complexo.

Figura 21 – Layout da tela de cadastro

Crédito: Ricardo Rodrigues Lecheta.

A seguir, podemos ver o arquivo XML de layout:

```
<?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" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nome" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Email" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sexo" />
    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <RadioButton
            android:id="@+id/radio_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Masculino" />
        <RadioButton
```

```
        android:id="@+id/radio_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Feminino" />

    </RadioGroup>
    <View
        android:layout_width="match_parent"
        android:layout_height="2dp"
        android:background="#eeeeee"
        android:layout_marginVertical="16dp"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Aceito os termos de uso" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Cadastrar" />
</LinearLayout>
```

5.4 NAVEGAÇÃO DE TELAS

Uma vez que fizemos os layouts das novas telas, precisamos adicionar o tratamento de eventos nos botões para fazer a navegação de telas. Assim, para cada elemento da tela ao qual queremos adicionar um evento, precisamos adicionar um identificador, que é chamado apenas de *id*.

Para adicionar o *id* em uma view, utilizamos o atributo **android:id** com o valor **@+id/codigoldAqui**, conforme demonstrado a seguir. Para botões, costumamos deixar o *id* como **btNomeBotao**.

Então, vamos lá. Localize o botão de login no layout e adicione o *id* conforme feito no código. É muito importante que você continue estudando para obter importantes dicas sobre como utilizar o assistente de código do Android Studio, a fim de digitar o código mais rapidamente.

```
<Button  
    android:id="@+id/btLogin"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Login" />
```

O próximo passo é adicionar ids nos dois textos (“Esqueci a senha” e “Cadastre-se”). Note que, no Android, podemos adicionar um id em qualquer view, o que permite encontrar essa view no código utilizando esse id.

```
<TextView  
    android:id="@+id/btEsqueciSenha"  
    android:text="Esqueci a senha"  
    . . . />  
<TextV
```

Pronto. Agora todas as views às quais queremos adicionar eventos têm ids (identificadores), pelos quais podemos encontrá-las. Para encontrar uma view, podemos usar o método **findViewById(id)** e, depois, o método **setOnClickListener()**, a fim de adicionar um evento nesse botão.

Por exemplo, para adicionar um evento no botão de login, basta usar este código:

```
findViewById<Button>(R.id.btLogin).setOnClickListener {  
    // Código aqui  
}
```

Veja que o método **findViewById** é genérico e você passa o tipo da view ao utilizá-lo – nesse caso, foi o tipo **Button**. Seguindo o mesmo raciocínio, para adicionar um evento no texto de “Esqueci a Senha”, podemos usar este código:

```
findViewById<TextView>(R.id.btEsqueciSenha).setOnClickListener {  
    // Código aqui  
}
```

Saiba mais

Para acessar uma view pelo id, usamos a classe R, gerada automaticamente ao compilar o projeto. Ela contém constantes para identificadores, imagens e textos do projeto.

Como agora temos um pouco de código e alguns detalhes são mais fáceis de explicar, podemos combinar o seguinte: tente digitar todos os códigos dos próximos exemplos, mas não deixe de estudar para pegar mais detalhes e dicas importantes.

Digite o código na classe **MainActivity** e execute o projeto no emulador. Confira os imports da classe para não digitar nada errado.

MainActivity

```
package com.example.helloandroid

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        findViewById<Button>(R.id.btLogin).setOnClickListener {
            startActivity(Intent(this, HomeActivity::class.java))
        }

        findViewById<TextView>(R.id.btEsqueciSenha).setOnClickListener {
            startActivity(Intent(this, EsqueciSenhaActivity::class.java))
        }

        findViewById<TextView>(R.id.btCadastrar).setOnClickListener {
            startActivity(Intent(this, CadastroActivity::class.java))
        }
    }
}
```

Ao executar o projeto no emulador, será possível clicar nos botões a fim de navegar para as outras telas do aplicativo. Para voltar à tela anterior, podemos usar o botão de “voltar” do Android. No entanto, geralmente os aplicativos adicionam o botão de voltar na AppBar (barra de navegação), no canto superior esquerdo. Isso é simples de ser feito: basta editar o arquivo de manifesto e adicionar o atributo `android:parentActivityName=".MainActivity"` em todas as novas activities. Isso indica que a **MainActivity** é a activity-pai e que o botão de voltar deve voltar para essa tela.

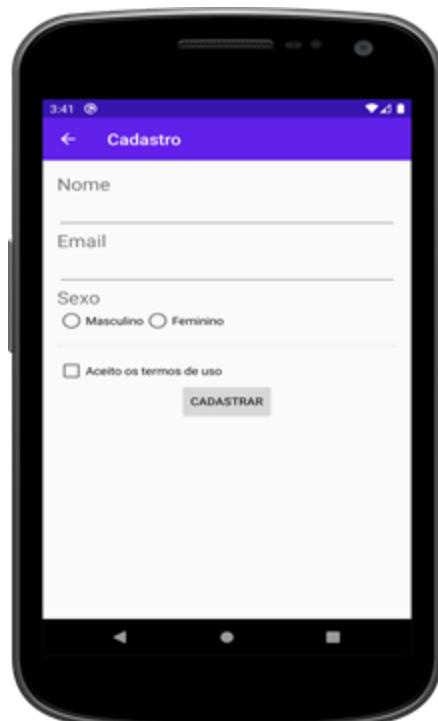
O arquivo `AndroidManifest.xml` ficará assim:

```
<activity
    android:name=".CadastroActivity"
    android:parentActivityName=".MainActivity"
    android:label="Cadastro"/>

<activity
    android:name=".EsqueciSenhaActivity"
    android:parentActivityName=".MainActivity"
    android:label="Esqueci Senha"/>
```

Observe que, além da tag **android:name**, que define o nome da classe, utilizamos a tag **android:label**, que define o título que será exibido na AppBar. Agora podemos executar o projeto no emulador e conferir o resultado. Ao clicar no botão “Cadastre-se” na tela de login, a tela de cadastro será chamada. Note que o botão de voltar na AppBar está funcionando e que o título da tela é “Cadastro”, conforme definido no arquivo de manifesto.

Figura 22 – Tela de cadastro no emulador



FINALIZANDO

Nesta aula, aprendemos a construir layouts simples no Android utilizando LinearLayout e combinando orientação vertical e horizontal. Também estudamos o FrameLayout e aprendemos a utilizar o conceito de *pilha*, para deixar uma view sobre a outra.

Com os conceitos aprendidos, será possível criar layouts de formulários e outros layouts simples.

Também começamos a estudar como adicionar o código na classe da activity e aprendemos a tratar os eventos dos botões para navegar para outras telas.

REFERÊNCIAS

DEVELOPER ANDROID. **Documentação Oficial.** Disponível em:

<<https://developer.android.com/>>. Acesso em: 2 jan. 2021.