

APRENDENDO ANDROID

- DO ZERO À GOOGLE PLAY -



Aprendendo Android - Do Zero à Google Play

Tiago Aguiar | tiagoaguiar.co ©2019

2º Edição

Sumário

1 Desenvolvedor Mobile: Uma Jornada Esperada!	3
2 O Que As Pessoas Dizem Sobre o Autor	5
3 Pré-requisitos: O Que Você Precisa Saber Antes De Começar	6
3.1 Os 4 Elementos Fundamentais Para Criar Aplicativos Android	6
3.2 Windows / Linux / MacOSX	7
4 Capítulo 0: Conceitos Sobre Programação	8
4.1 JVM - Java Virtual Machine	9
4.2 JDK e JRE	9
4.3 Como Instalar o Java no Windows	10
4.4 Como Instalar o Java/JDK no Mac OSX	14
4.5 Como Instalar o Java/JDK no Ubuntu/Linux	15
4.6 Meu Primeiro Programa	16
4.6.1 Olá Mundo!	17
4.7 Usando o IntelliJ IDEA	17
4.8 Primeiros Passos na Sintaxe Java	23
4.8.1 Comentários	23
4.8.2 Package e Import	24
4.8.3 Tipos de Dados	25
4.8.4 Condições e Lógica de Programação	25
5 Capítulo 1: Por Onde Começar?	28
5.1 Instalando o Android Studio	28
5.1.1 Passo a Passo para Instalar o Android Studio no sistema Windows .	29
5.1.2 Passo a Passo para Instalar o Android Studio no sistema Linux/Ubuntu	32
5.1.3 Passo a Passo para Instalar o Android Studio no sistema MacOSX .	35
5.2 Android SDK	36

6 Capítulo 2: Criando o Primeiro Aplicativo Android	42
6.1 Templates de Projetos	42
6.2 Configurando um Emulador	45
6.3 Estrutura de Pastas	50
6.3.1 Manifests	51
6.3.2 Java	52
6.3.3 generatedJava	52
6.3.4 Res	52
6.3.5 Gradle Scripts	53
6.4 Hello World	53
6.4.1 Que Tipo de Aplicativo Vamos Estar Criando?	53
7 Capítulo 3: Codificando o Primeiro Aplicativo Android	55
7.1 Dando Vida a Activity	66
7.2 Dando Vida ao Botão	69
8 Capítulo 4: Programação Orientada a Objetos em Kotlin	71
8.1 Primeiro Conceito: Objetos	71
8.2 Segundo Conceito: Classes	72
8.3 Tipos de dados	76
8.4 De Volta a MainActivity	76
9 Capítulo 5: Bug e Debugging	80
9.1 Breakpoints	80
9.2 LogCat	80
10 Capítulo 6: UX Design	83
10.1 Animations	83
10.2 Menus	84
11 Capítulo 7: Criando o Segundo Aplicativo Android - Chat Messenger	86
11.1 Configurando o Projeto Inicial	86
11.2 Interface do Usuário: Primeiros Passos Para Criar Telas e Activities	89
11.3 Criando Bordas Arredondadas	110
11.4 Tela de Cadastro de Usuário	112
11.5 Dando Vida a Nossa Atividade de Login	114

11.6 Interagindo Entre Activities	117
11.7 Firebase: Inicialização e Configuração	120
11.7.1 Registrando o Aplicativo no Firebase	125
11.7.2 #1 - Firebase Authentication	132
11.7.3 #2 - Firebase Storage	139
11.7.4 #3 - Firebase Database: Cloud Firestore	151
11.8 Lista de Mensagens e Contatos	153
11.8.1 Criando a Lista de Contatos	158
11.9 O Chat!	164
11.10 Refatorando Documentos e Coleções do Firestore	173
11.11 Enviando e Recebendo Mensagens do Chat	174
11.12 Listando as Últimas Mensagens de Cada Contato	183
12 Capítulo 8: Publicando o Aplicativo no Google Play	191
12.1 Criando um Aplicativo no Google Play	192
13 Conclusão	200
13.1 Links Úteis	200

1 Desenvolvedor Mobile: Uma Jornada Esperada!

Em 2011 eu decidi me tornar um programador para criar sistemas web. Em alguns anos eu estava participando de vários projetos de sistemas web que você possa imaginar: sistemas financeiros, sistemas de companhias aéreas e até sistema que se conectavam com hardwares específicos para medição de água e energia elétrica, tudo utilizando linguagem de programação de alto nível como Java, C/C++ e um pouco de baixo nível como Assembly.

Meu nome é Tiago Aguiar, especialista em desenvolvimento mobile (Android e iOS) a mais de 6 anos e foi no final de 2012 que tive meu primeiro contato com o mundo mobile - mais especificamente com Android - e mais uma vez, aquilo parecia mágico, muito mais do que a web. Eu mergulhei de cabeça para aprender a criar meus próprios aplicativos. No começo eu nem tinha intenção de trabalhar como desenvolvedor Android em empresas, nem mesmo ser freelancer me passa pela cabeça, mas depois do que eu descobri, esse pensamento mudou.

Depois de ter criado e publicado diversos aplicativos na loja Google Play, apps pessoais e para clientes atingindo a marca de **top rank 25** de uma categoria na loja com mais de **100k downloads**, eu sabia que tinha escolhido a profissão certa.

Essa mudança me proporcionou duas coisas incríveis. **Paixão pelo que faço e dinheiro!**

É mais do que comprovado que o mercado mobile possui basicamente duas vantagens e tendências muito grandes:

1. Uma demanda gigantesca (basta você ver o número de usuários que possuem smartphones andando na rua).
2. Baixa “mão de obra” no Brasil.

Logo, quanto maior a demanda e mais escasso o profissional, maior o valor daquele profissional. E outra, cada vez mais as empresas precisam de presença online, ou seja, é muito improvável uma empresa lançar um produto na web e não lançar sua versão de aplicativo. Em alguns casos, acontece o contrário - começamos com aplicativo e depois a versão web.

E por que eu estou te contando isso!?

Porque eu quero te mostrar que é sim possível ser um desenvolvedor mobile - mesmo que você ainda não saiba programar ainda.

Basta você aplicar o passo a passo correto e ir na direção certa. É exatamente isso que quero te mostrar nesse primeiro passo desse e-book - os exatos primeiros passos que me transformaram em um desenvolvedor Android Sênior profissional, coordenando um time com 4 desenvolvedores, e me dedicar a compartilhar o que tem funcionado na minha carreira com você. ;)

Eu sei que o mercado mobile é um mercado em potencial porque eu vivo nele dia a dia, ajudo outras pessoas a entrarem na área, evitando passar pelas mesmas dificuldades que passei quando comecei, que tenham mais clareza por onde começar e para onde devem ir. Que possam trabalhar com aquilo que gostam e não se preocupar com um salário baixo, sem reconhecimento e um futuro promissor.

Eu comecei fazendo isso com meu [canal do youtube](#), o [Blog](#) e o [facebook](#) compartilhando um pouco de como é viver sendo especialista em desenvolvimento mobile.

Se você quiser ver um pouco dos aplicativos que já criei, basta assistir esse video [aqui](#).



2 O Que As Pessoas Dizem Sobre o Autor

Dá uma olhada em algumas mensagens de pessoas que já ajudei com meus conteúdos disponibilizados na internet:

Parabéns por um dos melhores conteúdos Android que tem aqui no Brasil! Acompanho faz tempo e vou continuar acompanhando ~ Leo Carvalho

Quando comecei eu realmente estava perdido, sem rumo, e quando encontrei os videos do Tiago eu consegui abstrair tanto conhecimento, que cursos que eu comprei e paguei caro, se mostraram inúteis, com informações vagas, o Tiago consegue passar o conhecimento dele de tal forma que o aluno se cativa e realmente aprende, com informações reais, de qualidade e com muito profissionalismo. ~ Igor Silva - Ribeirão Preto - SP

A uma semana atras um programador Android me apresentou o seu canal, estou assistindo sua playlist de como criar o feed de filmes do Netflix, no código. Então estou passando aqui pra agradecer você pelo ótimo trabalho, trabalho como o seu está me ajudando muito nessa jornada. ~ Marcos

Suas técnicas me ajudaram a melhorar meus conhecimentos e minha forma de programar ~ Danillo Santos - São Paulo - SP - Brasil

Tiago estou na área à 4 anos e vendo os seus vídeos vejo que a cada dia podemos aprender muito mais, com os seus vídeos estou aprendendo muito mais. Muito obrigado pela iniciativa desta troca de experiências e informações. Um forte abs Thiago Baeza ~ Thiago J Baeza - São Paulo - SP - Brasil

Esse é apenas alguns dos diversos depoimentos que recebi de pessoas que já evoluiram de alguma forma com meus conteúdos e cursos e espero que este e-book possa ajudar você também ;)

3 Pré-requisitos: O Que Você Precisa Saber Antes De Começar

Se você quer aprender como criar aplicativos Android e não sabe por onde começar esse e-book irá te mostrar passo a passo o que você precisa saber para entrar na área mobile, onde cobriremos a linguagem de programação, as ferramentas de desenvolvimento como Android Studio, os componentes necessários para criar aplicativos Android e alguns aspectos e padrões de desenvolvimento de software.

Ao final deste material você será capaz de criar um aplicativo de Chat que funciona em tempo real - um Messenger - e publicá-lo no Google Play.

Continue aqui comigo que logo logo você estará criando seus aplicativos, combinado!? ;)

A minha intenção ao final desse e-book é que você seja capaz de entender como um software (leia-se apps) é construído. Que linguagem devemos aprender e dominar para criar qualquer tipo de aplicativo. Entretanto, o processo e a sua melhoria contínua em cada etapa é tão importante quanto o objetivo final.

3.1 Os 4 Elementos Fundamentais Para Criar Aplicativos Android

Muitas pessoas ficam com dúvidas e não sabem por onde começar quando o assunto é aplicativos Android. O fato é que você precisa entender 4 conceitos para criar os seus aplicativos:

1. Dominar a linguagem de programação: Existe 2 linguagens predominantes no mercado para se criar aplicativos, elas são Java e Kotlin (há outras também mas vamos manter o foco no que o Google recomenda). Oficialmente, Kotlin é a linguagem de programação moderna para criar aplicativos Android. Entretanto, não foi a primeira. Antes disso, os aplicativos eram construídos usando somente a linguagem de programação Java. Essa foi a linguagem inicial escolhida pelo próprio Google. Apesar de não mais ser o número 1, ela tem muita força e há milhões de projetos que foram, e ainda estão sendo, construídos em Java nos dias de hoje. O próprio Kit de Desenvolvimento Android que falaremos mais à frente, está escrito em Java. Entretanto, o mercado já exige que os profissionais saibam a criar seus aplicativos usando Kotlin. Dito isto, esse material irá seguir o passo a passo usando a linguagem **Kotlin** e falaremos também de Java por aqui - O app que vamos criar em Kotlin aqui, também possui a sua versão em Java no Canal do YouTube, confere lá ;)
2. Dominar o Android Studio: Android Studio é a única ferramenta que você precisa para criar a interface do seu aplicativo, escrever linhas de código, construir/compilar e publicar o seu aplicativo.
3. Dominar SDK: SDK (Software Development Kit) - Kit de desenvolvimento de software fornecido pela Google deixa nossa vida bem mais fácil. Este é um kit com diversos códigos-fonte e ferramentas que nos auxiliam a criar componentes Android como Listas, Cards para listar dados ou componentes para realizar popUps (alertas) no nosso aplicativo.

4. Execução: O processo de aprender uma nova habilidade depende de esforço e execução. Logo, além de ler e entender cada material, você precisa executar, por em prática cada passo e escrever mais e mais código!

Você tem que enriquecer seu conhecimento nessas **4 categorias** para criar aplicativos e melhorar cada vez mais.

E talvez você possa estar se perguntando...

“Nossa, é tanta coisa para aprender, estou ficando paralisado. Como eu vou aprender tudo isso?”

E você tem 2 boas notícias aqui:

A primeira é que isso é completamente normal. Eu já passei por isso e vou te ajudar a entender cada detalhe desses 3 primeiros conceitos. A segunda notícia boa é que você tem esse material (e outros) que publiquei na web que irão te ajudar nessa jornada. Então fique tranquilo ;)

Eu vou me dedicar 100% para que você domine as 3 primeiras categorias (linguagem de programação, Android Studio, Android SDK). Logo, eu apenas te peço que você se dedique 100% na quarta categoria (**Execução**).

Resultado vem de esforço!

É assim que melhoramos cada dia e que aprendemos de fato uma nova habilidade seja ela qual for.

EXECUÇÃO.

Escreva essa palavra em um post-it e cole no seu computador ou algo assim. Pratique ;)

3.2 Windows / Linux / Mac OSX

Primeiramente você precisará de um computador com o sistema operacional Windows, Linux ou Mac atualizado que seja, no mínimo um i5 com 4GB de RAM - claro que se tiver mais melhor!

Você também vai precisar de um editor de texto para escrever os programas (neste caso o Android Studio) que pode ser baixado neste [link aqui](#).

4 Capítulo 0: Conceitos Sobre Programação

Este capítulo é destinado àqueles que nunca escreveram nenhum programa e que, precisa entender os conceitos teóricos sobre linguagem de programação e compilação. Se você já entende estes conceitos, pode pular para o próximo capítulo onde iniciaremos a construção do nosso primeiro aplicativo Android.

Na computação existem diversas linguagens de programação que nos ajudam a escrever todo tipo de software para resolver problemas reais do mundo com a ajuda da tecnologia. Nos primórdios existiam diversas linguagens de máquina que chamamos de **linguagem de baixo nível**. Baixo nível porque é a linguagem que uma máquina consegue interpretar.

Porém, programar em uma linguagem de baixo nível como Assembly por exemplo, não é uma tarefa fácil. Sua leitura e escrita não é, digamos, humanizada. Veja:

```
lea si, string
call printf

hlt
string db "Olá mundo!", 0

printf PROC
    mov AL, [SI]
    cmp AL, 0
    je pfend

    mov AH, 0Eh
    int 10h
    inc SI
    jmp printf

pfend:
    ret
printf ENDP
```

Eis que surge as linguagens de alto nível como C++, Java, Swift, Python, Kotlin, Javascript, Objective-C entre outras.

Todas essas linguagens possuem suas peculiaridades e que podem ser melhores para resolver um determinado problema do mundo real de uma forma mais tranquila do que em uma linguagem de baixo nível. Essas linguagens são mais humanizadas, chegando perto da escrita que fazemos no dia a dia com um detalhe de “dizer” a máquina o que ela deve fazer.

No caso de aplicativos, é onde escrevemos todas as lógicas como, por exemplo: se o usuário digitar o login e a senha, exiba os dados X. Ou se o usuário apertar um botão, efetue o pagamento, etc.

Além disso, o Kotlin e o Java são linguagens que se popularizaram por ser muito poderosa com recursos de tipagem (onde podemos identificar valores de dados com tipos específicos) e com novos paradigmas como a orientação a objetos. Falaremos mais sobre isso.

Para que o códigos escritos nessas 2 linguagens possam ser executados em qualquer sistema operacional (Linux, Mac, Windows, Android), os engenheiros de software desenvolveram o que chamamos de **Máquina Virtual Java - JVM**.

4.1 JVM - Java Virtual Machine

Este “programa” nada mais é que uma máquina virtual que roda em cima do sistema operacional seja ele Linux, Windows ou Mac. Em resumo, é como se fosse uma outra máquina com configurações específicas que roda usando o seu processador e suas memórias. A JVM é responsável por executar um código gerado (que chamamos de compilado) pelo desenvolvedor que o escreveu usando a linguagem Java ou a Kotlin. A compilação gera um arquivo intermediário conhecido como **ByteCode**.

Ou seja, se você escrever um código em um arquivo **.kt** ou **.java** usando um computador Windows, você poderá compilar esse arquivo que será executado pela JVM em qualquer outro computador. Logo, se você quiser executar esse ByteCode em um Mac por exemplo, é possível. Tudo isso graças a JVM que está instalado tanto no Linux quanto no Mac.

É por isso que quando escrevemos nossos códigos no Windows, Linux ou Mac podemos executá-lo no Smartphone Android. Porque todos os smartphones que rodam um Android, são na verdade, um sistema Linux que está dentro dele!

Resumindo, escrevemos código em uma linguagem de alto nível como Java/Kotlin que pode ser compreendida pelo ser humano. Esse código é transformado - leia-se compilado - em um arquivo intermediário chamado de **byteCode**. E por fim, esse **byteCode** pode ser executado em qualquer sistema que tenha um interpretador desse **byteCode**, para algo de mais baixo nível que a máquina compreenda.

Quando entrarmos na parte prática esse conceito irá clarear a sua mente.

Se você é daqueles que gostam de entender como tudo começou, você pode seguir os próximos passos (que são opcionais) e instalar o Java para rodar o seu primeiro programa usando essa linguagem muito utilizada no início do Android.

Agora, se você já domina o Java e/ou já quer seguir em frente criando aplicativos modernos, pule para o próximo capítulo.

4.2 JDK e JRE

Assim como o Google desenvolveu o SDK do Android (o Kit de Desenvolvimento de Software), a Sun desenvolveu o **JDK - Java Development Kit**. Dentro desse Kit existem diversos códigos que compõe o Java como compiladores para **ByteCode**, códigos prontos para trabalhar com estrutura de dados e muito mais. O programa pronto que compila um código **java** em **bytecode** é chamado de **javac**. Esse ‘javac’ já está dentro do **JDK** depois que você fizer o download dele no site oficial da Oracle - ela comprou da Sun os direitos do Java hehe.

Já o JRE é o ambiente para a execução desses programas compilados em byteCode, a extensão desses arquivos é **.class**.

Se você instalar apenas o JRE, seu computador estará apto apenas a executar programas que foram escritos em Java. Para desenvolver um programa em Java, você precisará do kit completo JDK.

Não sei se você já usou algum programa da receita como declaração de imposto de renda, etc. Esses programas geralmente pedem para instalar o Java. O que na verdade é a instalação do interpretador JRE para rodar os programas desenvolvidos por eles.

Agora vamos aprender a instalar o JDK para conseguir desenvolver o primeiro programa em Java, compilá-lo e executá-lo com a ajuda do JRE e JVM.

Escolha o sistema operacional que você utiliza e siga o passo a passo para instalar. Caso queira testar o conceito de multiplataforma, você poderá criar um programa Java no Windows e executá-lo no Linux ou Mac por exemplo.

4.3 Como Instalar o Java no Windows

Para baixar o Java/JDK você precisará acessar o site oficial da Oracle. No momento que escrevo agora a última versão disponível é o Java 12 - caso você encontre uma versão superior a 12, não tem problema, pode baixar.

Você pode acessar por aqui no [site da oracle](#) ou pesquisar no google pelo termo **download jdk**.

Na página, você deverá aceitar os termos e condições e escolher o arquivo de acordo com o seu sistema operacional.

Java SE Development Kit 12.0.2		
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.		
Thank you for accepting the Oracle Technology Network License Agreement for Oracle Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux	155.14 MB	jdk-12.0.2_linux-x64_bin.deb
Linux	162.79 MB	jdk-12.0.2_linux-x64_bin.rpm
Linux	181.68 MB	jdk-12.0.2_linux-x64_bin.tar.gz
macOS	173.63 MB	jdk-12.0.2_osx-x64_bin.dmg
macOS	173.98 MB	jdk-12.0.2_osx-x64_bin.tar.gz
Windows	158.63 MB	jdk-12.0.2_windows-x64_bin.exe
Windows	179.57 MB	jdk-12.0.2_windows-x64_bin.zip

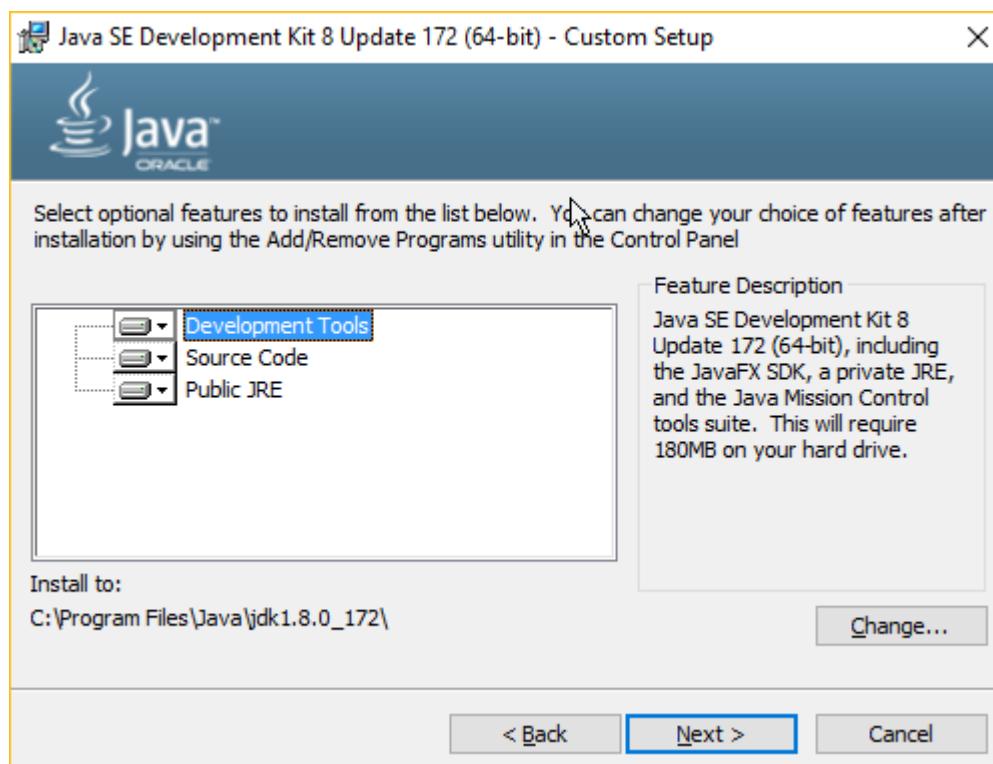
Caso não saiba o tipo de sistema operacional (64bits ou 32bits) você pode encontrar em:

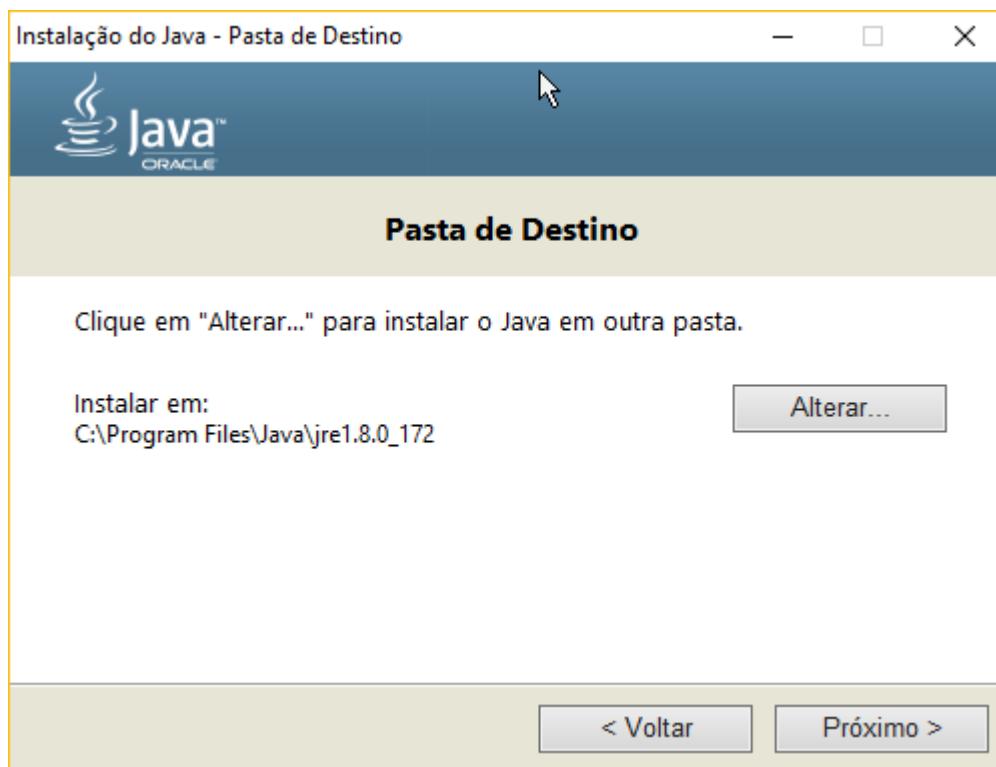
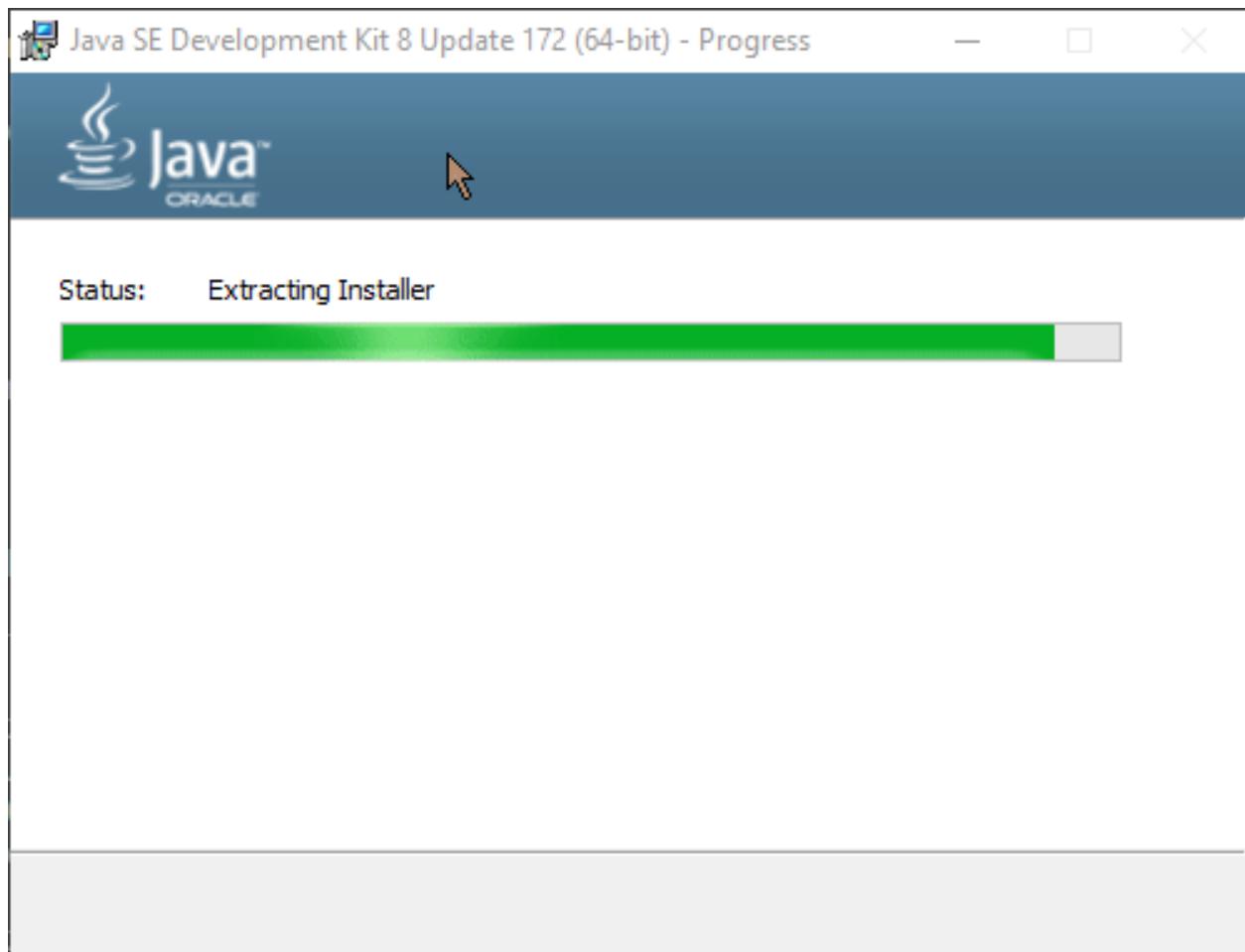
- Clique com botão direito em **Meu Computador** > **Propriedades** E veja a seção sistema.

Sistema

Processador: Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz
 Memória instalada (RAM): 16.0 GB (utilizável: 15.9 GB)
 Tipo de sistema: Sistema Operacional de 64 bits, processador com base em x64
 Caneta e Toque: Suporte a Toque com 10 Pontos de Toque

Após efetuar o download do arquivo, execute-o e siga o passo a passo do instalador - verifique exatamente o local onde ele irá instalar o Java porque futuramente você pode precisar.







Agora seu computador já possui o Java. Para realizar a última verificação, procure pelo prompt de comando no menu iniciar e digite: `java -version`.

A saída do terminal deverá exibir a versão do Java:

```
C:\Users\tiago>java -version
java version "1.8.0_172"
Java(TM) SE Runtime Environment (build 1.8.0_172-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.172-b11, mixed mode)

C:\Users\tiago>
```

Alguns programas utilizam os recursos de variáveis de ambiente, que nada mais é do que o caminho onde ficam os programas para o computador e os desenvolvedores acessarem rapidamente o Java. Logo, é interessante também adicionar essa variável de ambiente ao seu sistema.

Abra o prompt de comando novamente, porém desta vez, como Administrador - clique com o direito **executar como administrador**.

Digite o comando a seguir: `setx -m JAVA_HOME "c:\Program Files\Java\jdkx.x.x_xxx\bin"`. Sendo que o *x.x._xxx* é a versão do seu JDK e o caminho que você instalou o Java anteriormente.

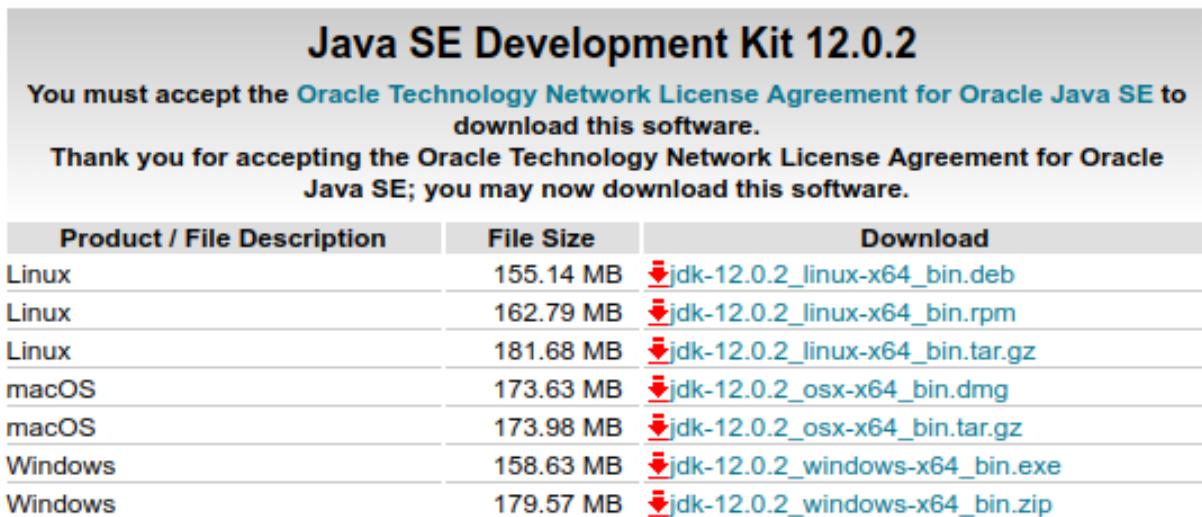
Feche e abra novamente o prompt de comando e execute o comando `set`. Isto listará uma série de variáveis de ambiente. Procure a que você definiu como **JAVA_HOME**.

4.4 Como Instalar o Java/JDK no Mac OSX

Para baixar o Java/JDK você precisará acessar o site oficial da Oracle. No momento que escrevo agora a última versão disponível é o Java 12.

Você pode acessar por aqui no [site da oracle](#) ou pesquisar no google pelo termo **download jdk**.

Na página, você deverá aceitar os termos e condições e escolher o arquivo que corresponde ao **Mac OSX** com a extensão **dmg**.



Java SE Development Kit 12.0.2

You must accept the [Oracle Technology Network License Agreement for Oracle Java SE](#) to download this software.

Thank you for accepting the Oracle Technology Network License Agreement for Oracle Java SE; you may now download this software.

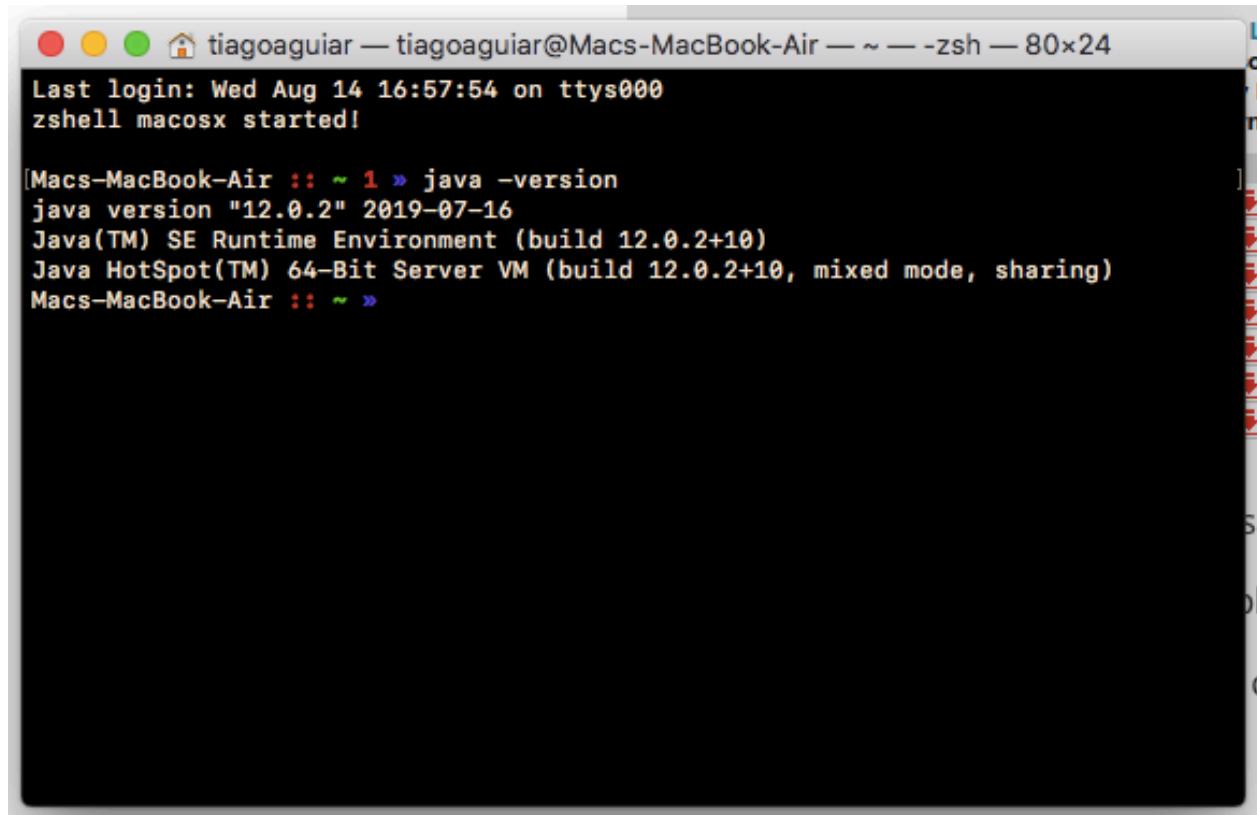
Product / File Description	File Size	Download
Linux	155.14 MB	jdk-12.0.2_linux-x64_bin.deb
Linux	162.79 MB	jdk-12.0.2_linux-x64_bin.rpm
Linux	181.68 MB	jdk-12.0.2_linux-x64_bin.tar.gz
macOS	173.63 MB	jdk-12.0.2_osx-x64_bin.dmg
macOS	173.98 MB	jdk-12.0.2_osx-x64_bin.tar.gz
Windows	158.63 MB	jdk-12.0.2_windows-x64_bin.exe
Windows	179.57 MB	jdk-12.0.2_windows-x64_bin.zip

Abra o arquivo baixado e você verá um arquivo na extensão **.pkg** e inicie o instalador do Java.

O *Wizard* para instalação do Java no Mac OSX é bem simples e intuitivo. Siga o passo a passo.

Ao final, feche o terminal, abra-o novamente e execute o comando: `java -version`

Você verá a saída com a versão do Java.



```
tiagoaguiar — tiagoaguiar@MacBook-Air — ~ — -zsh — 80x24
Last login: Wed Aug 14 16:57:54 on ttys000
zsh shell macosx started!

[Macs-MacBook-Air :: ~ 1 » java -version
java version "12.0.2" 2019-07-16
Java(TM) SE Runtime Environment (build 12.0.2+10)
Java HotSpot(TM) 64-Bit Server VM (build 12.0.2+10, mixed mode, sharing)
Macs-MacBook-Air :: ~ »
```

4.5 Como Instalar o Java/JDK no Ubuntu/Linux

Para baixar o Java/JDK você precisará acessar o site oficial da Oracle. No momento que escrevo agora a última versão disponível é o Java 12.

Você pode acessar por aqui no [site da oracle](#) ou pesquisar no google pelo termo **download jdk**.

Na página, você deverá aceitar os termos e condições e escolher o arquivo de acordo com o seu sistema operacional.

Java SE Development Kit 12.0.2

You must accept the [Oracle Technology Network License Agreement for Oracle Java SE](#) to download this software.

Thank you for accepting the [Oracle Technology Network License Agreement for Oracle Java SE](#); you may now download this software.

Product / File Description	File Size	Download
Linux	155.14 MB	jdk-12.0.2_linux-x64_bin.deb
Linux	162.79 MB	jdk-12.0.2_linux-x64_bin.rpm
Linux	181.68 MB	jdk-12.0.2_linux-x64_bin.tar.gz
macOS	173.63 MB	jdk-12.0.2_osx-x64_bin.dmg
macOS	173.98 MB	jdk-12.0.2_osx-x64_bin.tar.gz
Windows	158.63 MB	jdk-12.0.2_windows-x64_bin.exe
Windows	179.57 MB	jdk-12.0.2_windows-x64_bin.zip

Baixe a versão do arquivo com a extensão `.tar.gz` para seguir com os exemplos abaixo:

Acesse o diretório onde se encontra o arquivo `jdk` e extraia-o com o comando: `tar -zxvf jdk-xxx.tar.gz`. Depois, execute a sequência de comandos abaixo no terminal do Ubuntu / Linux:

```
cd /usr/lib/
sudo mkdir java
sudo mv ~/Downloads/jdk-xx/ /usr/lib/java/
sudo update-alternatives --install "/usr/bin/java" \
"java" "/usr/lib/java/jdk-xxx/bin/java" 1
sudo update-alternatives --install "/usr/bin/javac" \
"javac" "/usr/lib/java/jdk-xxx/bin/javac" 1
sudo gedit ~/.bashrc
```

Com o editor aberto (`gedit`) adicione as 2 linhas seguintes (trocando a versão do seu `java`).

```
export JAVA_HOME=/usr/lib/java/jdk-xxx
export PATH="$PATH:$JAVA_HOME/bin"
```

Feche o terminal, abra-o novamente e execute o comando: `java -version`

Você verá a saída com a versão do Java.

Por fim, verifique o caminho onde se encontra a variável de ambiente `JAVA_HOME`: `echo $JAVA_HOME`.

4.6 Meu Primeiro Programa

Com o java instalado no seu Mac, Linux ou Windows vamos escrever nosso primeiro programa. Se você já conhece um pouco sobre programação sabe que esse programa é o famoso **Hello World** - Olá Mundo!.

Geralmente, para escrever qualquer tipo de software usamos editores de textos ou IDE (Integrated Development Environment) que são ferramentas para otimizar nosso tempo e conseguir evitar o máximo de erros durante o desenvolvimento.

Mas neste exemplo, vamos usar um editor padrão do seu computador para realizar esta tarefa.

4.6.1 Olá Mundo!

Abra seu editor preferido de texto (Bloco de Notas, XCode, Gedit, etc). E escreva o seguinte código.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Não se preocupe com o que está escrito ainda, apenas salve o arquivo como **Main.java** e feche o editor.

Agora abra o seu Terminal e digite:

javac Main.java. Esse comando irá compilar o seu programa e gerar um arquivo **Bytecode** para o JVM/JRE interpretar - o arquivo **Main.class**.

Agora execute o comando **java Main**. E você terá a mensagem **Hello World!** no seu terminal.

Aparentemente não foi uma coisa tão excepcional de se fazer, mas é importante notar que você criou um programa e há infinitas possibilidades agora. Claro que não escrevemos nada que crie uma interface gráfica amigável para um usuário comum, mas podemos criar isso futuramente.

Dito isto, vamos começar a realmente adentrar em um projeto Java e entender o que significa cada palavra. Para fazer isso usaremos uma IDE para otimizar o nosso tempo.

Existe várias ferramentas desenvolvidas para desenvolvedores Java e Kotlin, mas como o foco aqui é aprender a criar aplicativos Android, vamos usar o **IntelliJ IDEA** que é uma ferramenta base do Android Studio.

4.7 Usando o IntelliJ IDEA

[Clique no link](#) e faça o download da **versão Community** que é a versão gratuita da ferramenta. Instale sua nova IDE seguindo os passos necessários indicados pela plataforma.

Após abrir a IDE, ela irá perguntar se você deseja importar algumas configurações de outro IntelliJ. Marque que não - **Do not import settings**.

Aceite os termos para começar a usar a ferramenta.

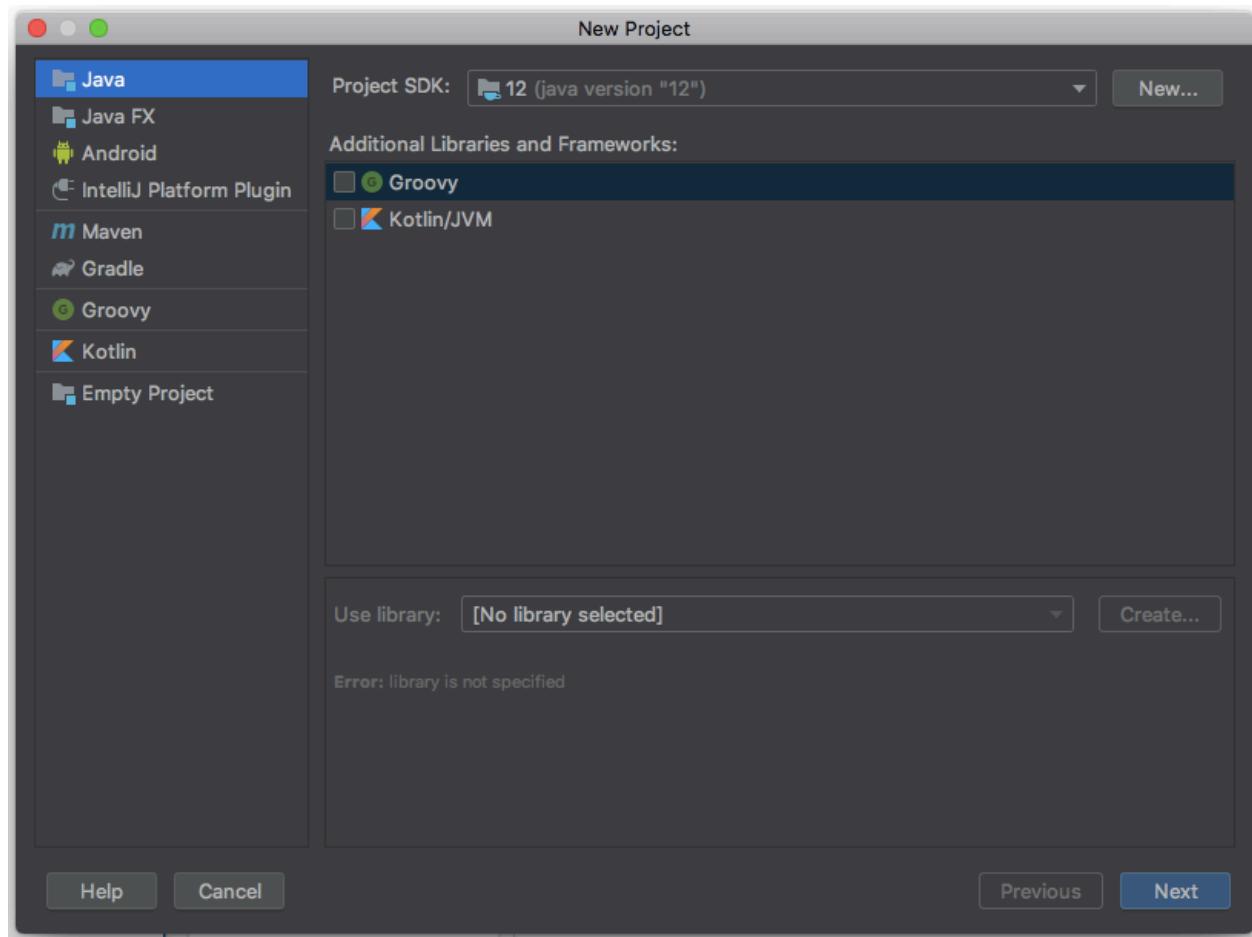
Depois, a ferramenta irá provavelmente perguntar se você aceita enviar dados de análise e uso de estatística para melhorar a plataforma. Como não vamos usar a fundo o IntelliJ nesse primeiro instante, marque como não - **Don't send**.

Escolha um tema - **Darcula ou Light**. Depois é só seguir com o Next até o **Start using IntelliJ IDEA**.

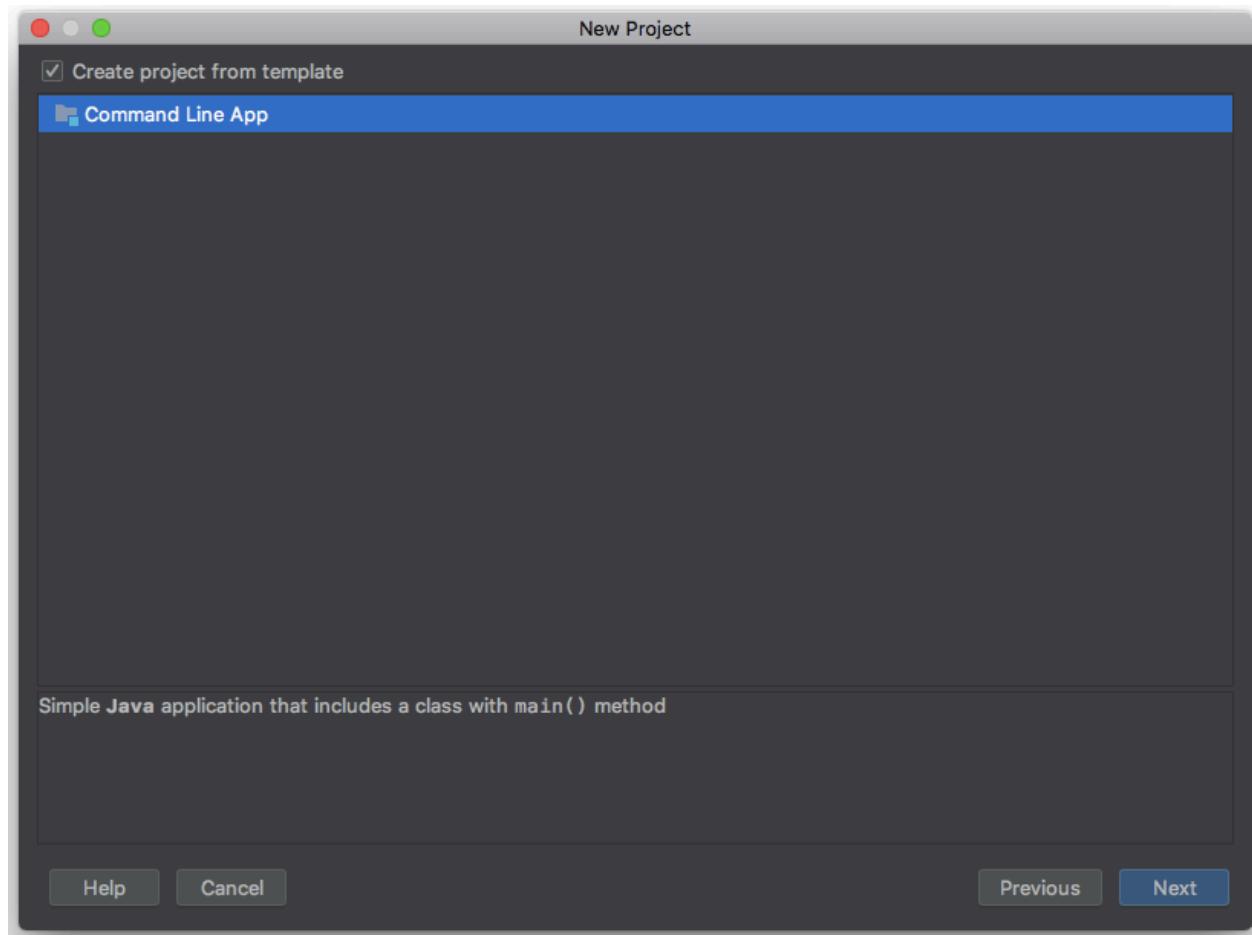
Clique em **Create New Project**:



Selecione Java e clique em **Next**:

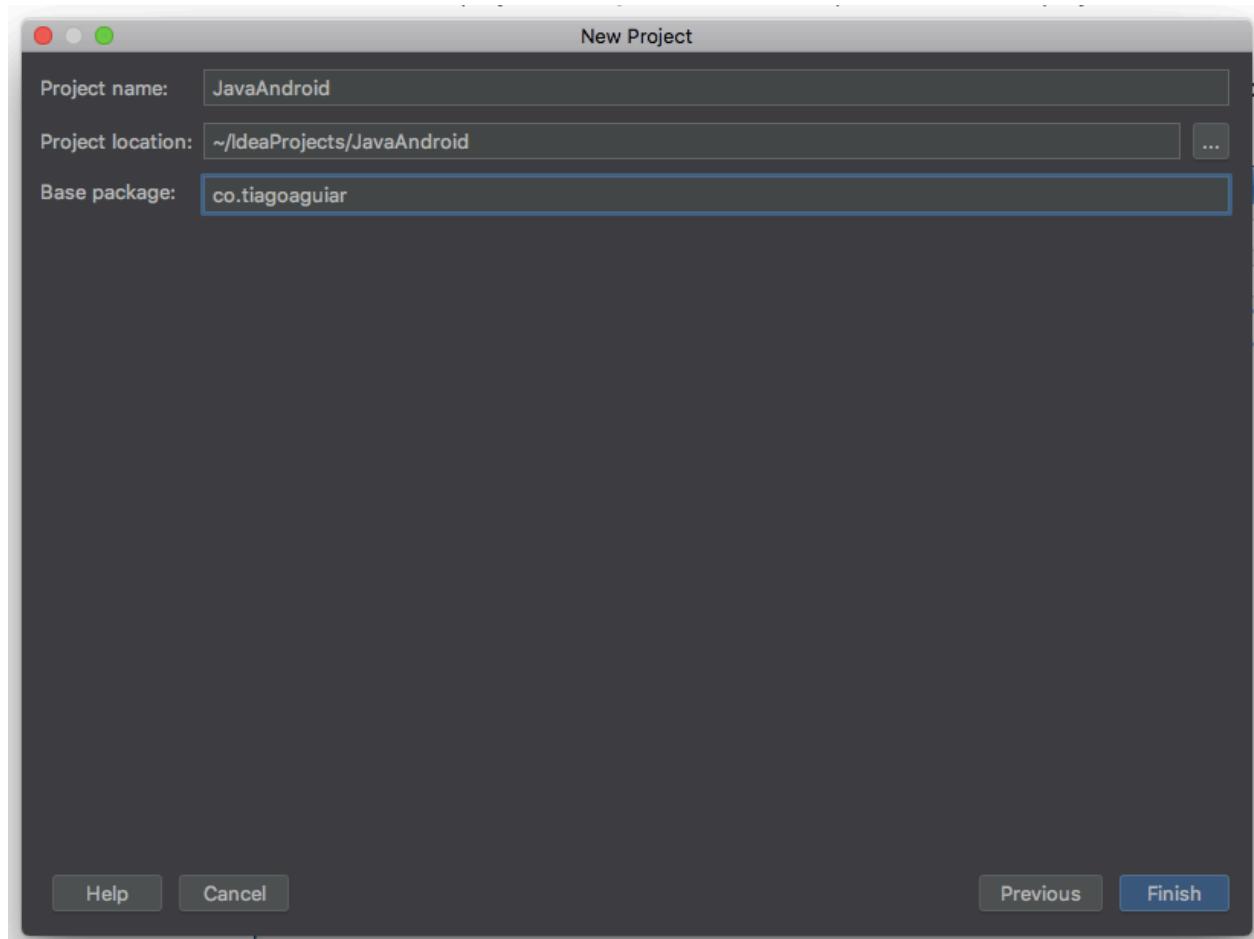


Selecione a opção **Create project from template** e crie um projeto **Command Line App**:



Defina o nome do projeto como **JavaAndroid**, marque o local onde o projeto ficará no seu computador (veja o Project Location) e adicione o pacote **co.tiagoaguiar**.

Nomes de pacotes em programação costumam ser o domínio da empresa invertido. Ex: com.google, br.com.uol, etc.



Escreva novamente o código do Hello World anterior e clique no botão de Play no menu superior do editor.

Novamente você verá a saída **Hello World!** só que agora na saída interna do editor.

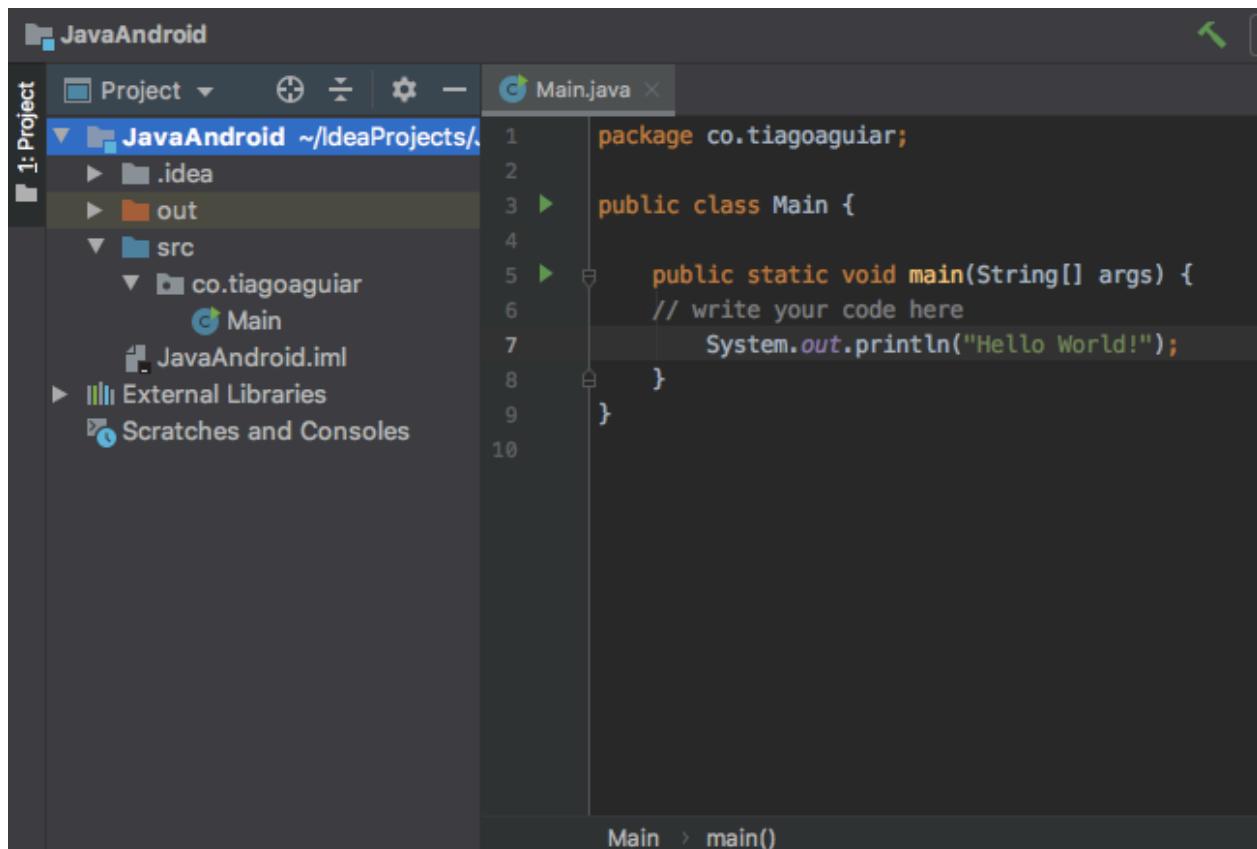
```

1 package co.tiagoaguiar;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // write your code here
7         System.out.println("Hello World!");
8     }
9
10

```

Também é possível ver como é estruturado um projeto Java simples. Veja que temos a pasta **src** que é uma abreviação de **source** = fonte. Dentro de src temos o pacote que definimos anteriormente **co.tiagoaguiar**. Esse pacote é apenas uma forma de organizarmos nosso projeto para que os arquivos não começem a ficar confuso. Imagine um projeto com 1000 arquivos rs, precisamos de organização para manter a sanidade do desenvolvedor XD.

Falaremos de pacotes mais à frente.



```

1 package co.tiagoaguiar;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // write your code here
7         System.out.println("Hello World!");
8     }
9 }
10

```

4.8 Primeiros Passos na Sintaxe Java

Nos próximos capítulos vamos entender como é a sintaxe Java voltada para projetos Android. O Java é uma linguagem de programação muito rica. Logo, isso nos permite, além de criar aplicativos, criar:

- sistema web como e-commerce
- portais ou qualquer outro sistema corporativo
- sistemas desktop
- e outros softwares embarcados em automóveis, etc.

Para começar, vamos entender primeiramente os conceitos de lógicas de programação e como um programa é escrito para o usuário interagir e tomar decisões.

4.8.1 Comentários

Como você já deve saber, tudo que escrevemos em um arquivo **.java** é compilado e interpretado pelo programa, **exceto comentários!**

Comentários em um código nos permite escrever qualquer coisa que um programa não interprete, como por exemplo, citações de como aquele código funciona, nome de autor do código, lógica de algoritmos e documentação do código-fonte.

Para escrever comentários em Java você pode fazer isso de 2 formas.

A primeira é usar comentários de uma linha só.

```
// este é um comentário
```

Para comentários em várias linhas usamos os caracteres `/* */`.

```
/* Este é um novo comentário
 * em java com multiplas linhas de código
 */
```

Simples não!

Sempre que precisar atribuir uma nota ao seu código-fonte, utilize o recurso de comentários (com moderação para não poluir o seu código).

4.8.2 Package e Import

No começo do nosso código-fonte, mais especificamente na linha 1, temos uma sintaxe chamada de **package co.tiagoaguiar;**

Isso é nada mais nada menos do que o pacote - estrutura de pastas - onde se encontra o arquivo. Quando desenvolvemos vários projetos, sempre definimos os pacotes para organizar nossos códigos e para separar as coisas.

Você irá notar futuramente que, conforme você cria novos códigos Java, você precisa definir pacotes para organizar responsabilidades de arquivos como por exemplo, pacote de autenticação, pacote de acesso a banco de dados, etc.

Imagine o seguinte: em um projeto você tem 2 códigos-fonte chamado **User.java**. Para que o programa entenda qual você quer usar, ele precisa olhar para os pacotes e entender que o User que você quer usar é do pacote **co.tiagoaguiar.model.User** e não do pacote **co.tiagoaguiar.model.app.User**.

Os dois são usuários, mas de partes diferentes do sistema.

Sabemos que o package define onde fica o arquivo, já o **import** indica que você quer usá-lo no arquivo corrente.

Por exemplo: Se nós tivéssemos criando uma lista de produtos de um determinado usuário, muito provavelmente, precisaríamos importar no arquivo **Product.java** o código de **User.java**.

```
import co.tiagoaguiar.model.User;

// agora consigo usar os códigos do arquivo User.java
```

Talvez ainda não faça sentido para você o conceito de import e package porque ainda não usamos nenhum conceito de classe e/ou programação orientada a objetos. No capítulo sobre orientação a objetos (que é um paradigma de desenvolvimento) iremos fazer alguns exercícios a respeito.

Por hora, só tenha em mente que *package* define onde o arquivo fica e *import* define que queremos usá-lo no arquivo corrente.

4.8.3 Tipos de Dados

Ao final de todo programa, o computador apenas lê de forma binário (1 ou 0). Porém, seria insano programar usando apenas 1 ou 0 e alocando na memória do computador esses valor.

Para facilitar nossa vida, as linguagens de programação de alto nível acabam definindo tipos para os seus dados. Tipos como número inteiro, número decimal, textos, tipos lógico como verdadeiro ou falso e assim por diante.

Como falei, no fim, tudo virá dado na memória do computador. E em Java esses dados são fortemente tipados.

Para criar um número definimos `int x = 10`. Para criar um número com casa decimal como o PI fazemos `double pi = 3.14`. Textos são chamados de **Strings** `String name = "Android"`; e usando aspas duplas.

Para elementos verdadeiro ou falso - booleanos - usamos `boolean autenticado = false`.

Existe além desses tipos que chamamos de primitivos os tipos Objetos. Que nada mais é do que um conjunto de atributos que o forma.

Por exemplo: Um objeto chamado **Usuário** possivelmente terá atributos como **email** e **senha**, sendo os dois Strings - textos.

Quando começarmos a criar nossos próprios objetos vamos entender o que eles representam e como eles são definidos.

Na verdade você já usou 1 objeto no primeiro código **Hello World**. O objeto **System**, do próprio Java.

Os tipos primitivos em Java são:

- boolean
- byte
- char
- short
- int
- long
- float
- double

4.8.4 Condições e Lógica de Programação

Já diria um mentor meu que programar é o ato de passar instruções a máquina de forma lógica baseada em algumas condições. Isto tudo é verdade.

Agora vamos entender como um computador “toma decisões” quando é apresentado uma ou mais condições a se seguir.

Quando queremos que um programa faça A ou B ou fique executando algo repetidamente, usamos a sintaxe de **if**, **else**, **for**, **while** e **switch**.

Vou explicar melhor.

Quando em uma variável temos um valor X, podemos instruir o programa a tomar duas decisões, semelhante quando você digita o nome de usuário em um formulário com a senha. Se a senha estiver correta, efetue o login, senão, exiba uma mensagem de erro para o usuário.

```
public class Main {
    public static void main(String[] args) {
        int idade = 16;
        if (idade > 16) {
            System.out.println("Você já pode tirar a habilitação CNH");
        } else if (idade > 14) {
            System.out.println("Você ainda NÃO pode tirar a habilitação CNH");
        } else {
            System.out.println("Você ainda é uma criança");
        }
    }
}
```

O mesmo se aplica quando queremos executar instruções repetidas vezes.

```
int idade = 14;
while (idade > 0) {
    System.out.println("idade é: " + idade);
    idade = idade - 1;
}
```

Enquanto a idade for maior que 14, o programa irá permanecer em *loop* sempre executando os códigos dentro do bloco **while**. Veja também que na última linha, é atribuído a variável **idade** o valor da própria idade menos 1. Logo, a cada interação, a idade irá reduzir em uma vez. Quando chegar a 0, a condição (**idade > 0**) será falsa e o programa irá sair desse loop.

Existe uma outra maneira de fazer um loop utilizando o **for**, que também verificará uma condição, mas sempre irá aumentar ou diminuir uma variável. Veja:

```
for(int idade = 14; idade > 0; idade--) {
    System.out.println("idade é: " + idade);
}

// ou ao contrário
for (int idade = 0; idade > 14; idade++) {
    System.out.println("idade é: " + idade);
}
```

Execute os dois exemplos acima e veja o que acontece.

Por fim, temos uma alternativa ao **if** que chamamos de **switch/case**.

```
int idade = 16;
switch (idade) {
    case 16:
        System.out.println("você tem 16 anos");
        break;
    case 18:
        System.out.println("você tem 18 anos");
        break;
    default:
        System.out.println("sua idade não correspondeu a nenhuma das anteriores");
        break;
}
```

Mais de 50% das decisões que tomamos nos aplicativos pode ser tratado com **if / else** como no exemplo de formulários de cadastro / login, acessos a internet, verificação de posição no GPS e assim por diante.

Entender os fundamentos de lógica de programação é o seu primeiro passo antes de entrar em paradigmas e padrões de projetos como a Orientação a Objetos ou programação reativas.

No começo deste ebook eu falei que a melhor forma de aprender é executando! Por isso, use e abuse desses conceitos dentro do seu método **main**. Entenda como funciona as lógicas de programação para então, partir para conceitos um pouco mais avançados como estruturas de dados e etc.

Nota: é importante ressaltar que para manter um código limpo, sempre devemos identar as linhas conforme os blocos. Alguns usam 4 espaços outros TAB. O importante é não manter tudo na mesma linha ok.

Agora que você já se familiarizou um pouco com um programa de computador, vamos criar algo mais interessante para o nosso objetivo - vamos criar o primeiro aplicativo Android moderno usando Kotlin!

5 Capítulo 1: Por Onde Começar?

A partir deste e dos próximos capítulos iremos concluir as seguintes etapas:

- Instalar a ferramenta Android Studio
- Entender a linguagem de programação (Kotlin)
- Criar nosso primeiro Aplicativo
- Explorar conceitos sobre desenvolvimento de software
- Utilizar recursos de Debugging
- Aprender sobre Design
- Criar nosso segundo Aplicativo
- Publicar o aplicativo no Google Play e distribuir para o Mundo!

Pois bem. Para criar aplicativos Android você precisará de um editor de texto para escrever a lógica e desenhar as telas do seu aplicativo.

Você até poderia fazer isto em um editor de texto como bloco de notas ou qualquer outro. Porém, é extremamente improdutivo e os erros que você vai cometer são enormes e difícil encontrá-los.

Para criar algum tipo de software (como aplicativos) usamos ferramentas que possuem a capacidade de escrever códigos + outros recursos como: compiladores integrados - compilar é o ato de transformar o que você escreve em linguagem de máquina que pode ser entendido pelo computador.

Além de compiladores, essas ferramentas que chamamos de **IDE - Integrated Development Environment** ainda possuem recursos para depurar, autocomplete de código, indicadores de erro de sintaxe na escrita da linguagem e muito mais.

Essa IDE é o Android Studio onde você pode:

- Editar o código
- Compilar para linguagem de máquina
- Executar o aplicativo em Emuladores Android
- Publicar no Google Play

O Android Studio é um programa escrito em Java e precisa do Kit (JDK) para funcionar. As versões mais recentes já possuem o Java incorporado - que dispensa essa instalação - mas se você quiser instalar o Java, no capítulo 0 cobrimos esse passo a passo.

5.1 Instalando o Android Studio

O **Android Studio** é a ferramenta oficial criada pelo Google com recursos para criar aplicativos Android em todos os tipos de dispositivos que possuem o sistema Android (além de smartphones). Desenvolvida a partir do **IntelliJ IDEA** tornando-a uma ferramenta muito poderosa.

- O poder dos emuladores para simular configurações de dispositivos e smartphones e todos os recursos avançados em um só lugar.
- Ambiente unificado para criar aplicativos em todos dispositivos que são baseados em Android.
- Criar os emuladores mais rápidos e semelhantes a um smartphone real.
- Lint (ferramenta para encontrar problemas no código) rico para encontrar problemas, compatibilidade e outros recursos
- Inspeccionar a fundo o seu aplicativo para otimizá-lo e reduzir ao máximo o seu tamanho. Além disso você pode comparar APKs mesmo que eles não tenham sido desenvolvidos pelo Android Studio
- Produtividade no desenvolvimento do aplicativo é o que difere programadores de “programadores”. Aprenda todos os recursos que só o Android Studio tem a oferecer para deixá-lo mais produtivo.
- O Gradle pode deixar o seu desenvolvimento mais flexível e conseguir criar projetos com diversas variações como dispositivos e bibliotecas.
- Recursos muito interessantes para você analisar estatisticamente a performance do seu aplicativo. Seja a memória, CPU, internet e outros recursos do aparelho. Isso pode ser o diferencial para maximizar a performance do seu aplicativo.
- Integração com sistemas de controle de versão Git.
- Integração com frameworks de testes
- Suporte a C++, NDK e Kotlin
- Editar o layout visualmente. Tenha uma prévia em qualquer tamanho de dispositivo e veja como ficará o seu aplicativo de forma rápida.

Para se manter atualizado acesse a página oficial com os últimos lançamentos sobre o [Android Studio Release](#).

O Android Studio está disponível para Mac, Windows e Linux. Logo abaixo, segue o passo a passo de como instalá-lo de acordo com o seu sistema operacional.

5.1.1 Passo a Passo para Instalar o Android Studio no sistema Windows

Assim como outros programas de desenvolvimento você precisará verificar os requisitos mínimos para executar o Android Studio.

Você precisará de:

- Um PC com Windows 7/8/10 de 32bits ou 64bits - Mínimo de 6G RAM. 8GB RAM é o recomendado pelo Google e 1GB para os emuladores caso for utilizar - 2GB de espaço livre em disco - recomenda-se 4GB porque a IDE usará 500mb e 1,5GB para os emuladores

Para baixar o Android Studio você precisará acessar o site oficial do Google conhecido como Android Developer [developer.android.com](#). Inclusive, além de **baixar o Android Studio** você também encontra toda a documentação da plataforma Android e das suas linguagens de programação.

Ao acessar a página oficial o site identificará que você está acessando-o pelo Windows e irá exibir o download para o sistema Windows.

Aceite os termos e condições para ter acesso ao arquivo de instalação final.

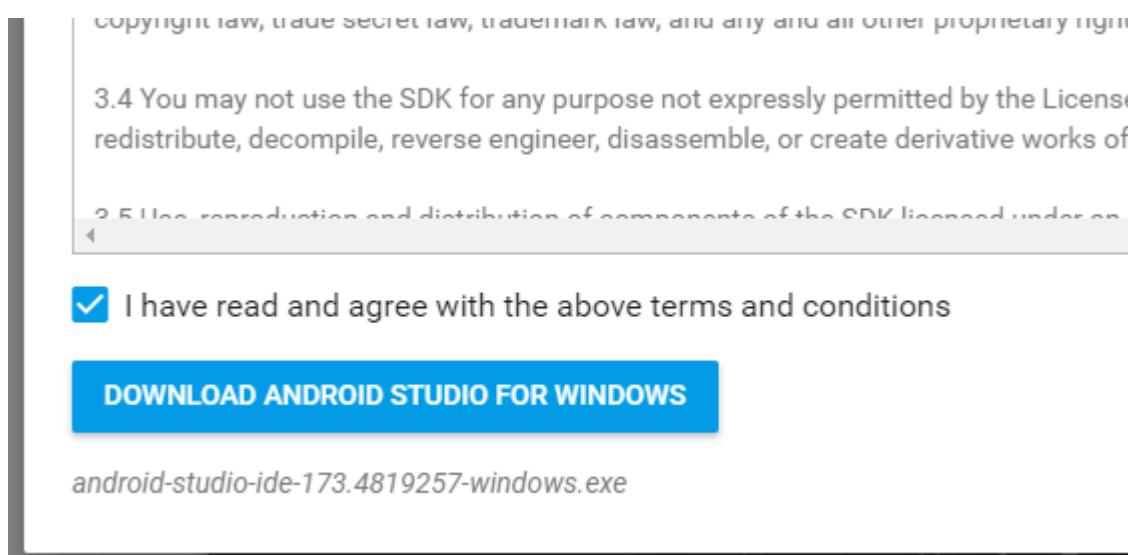
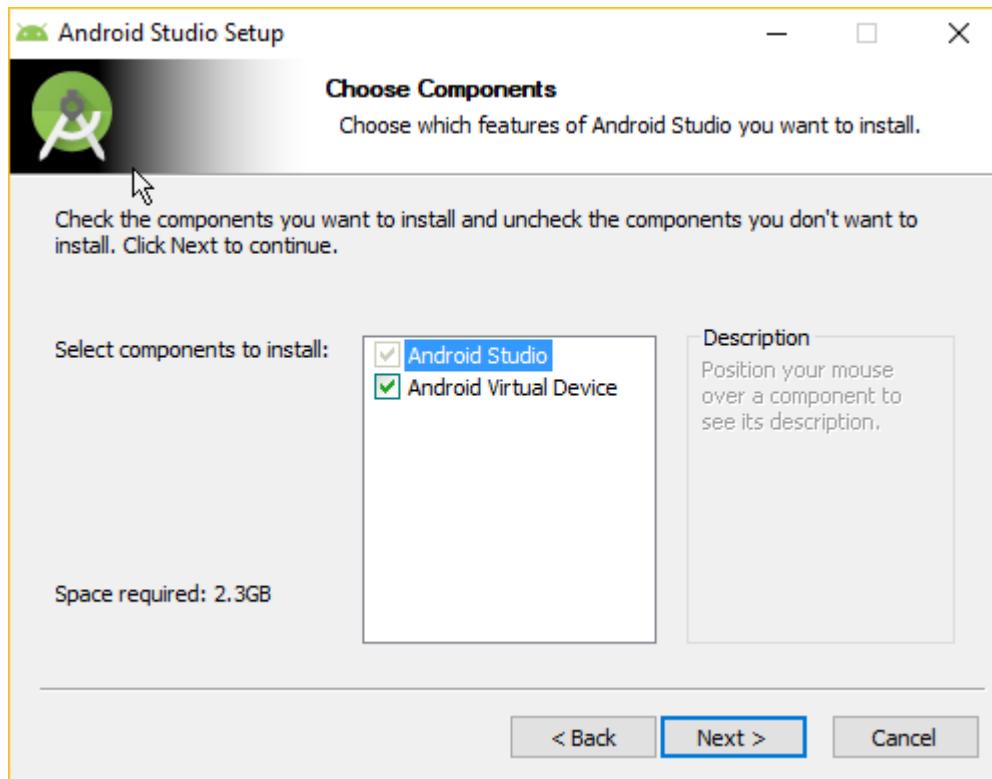


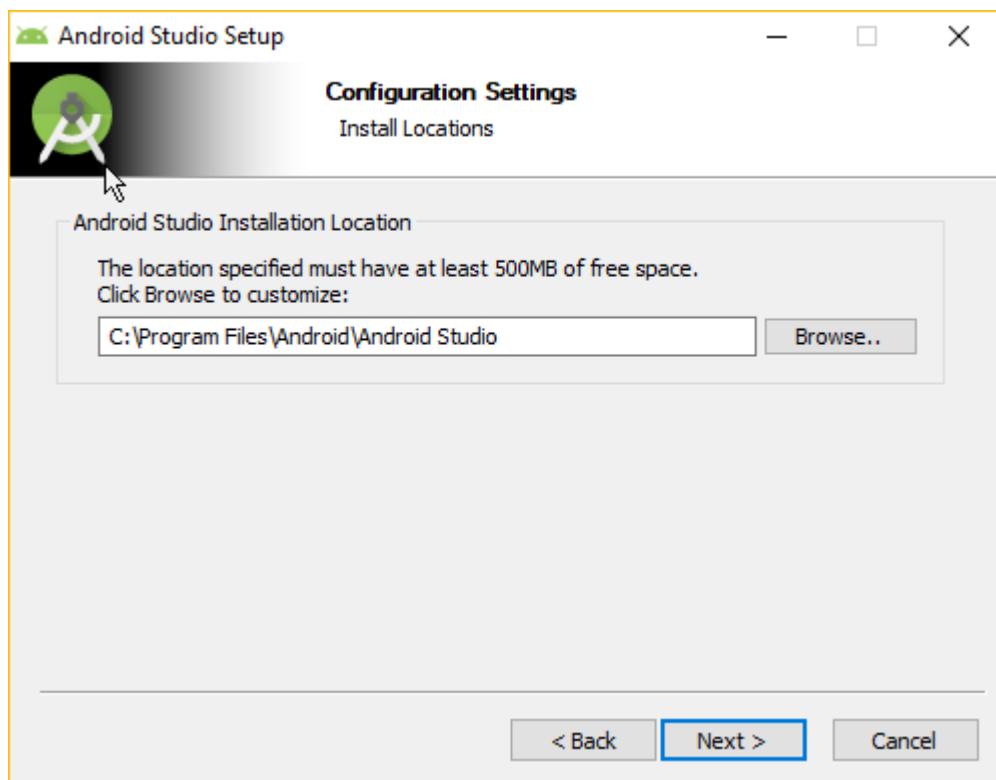
Figure 1: Como Instalar o Android Studio no Windows 01

Execute o arquivo baixado (instalador .exe) e siga o passo a passo do instalador



Mantenha o *check* para instalar também os dispositivos virtuais - emuladores.

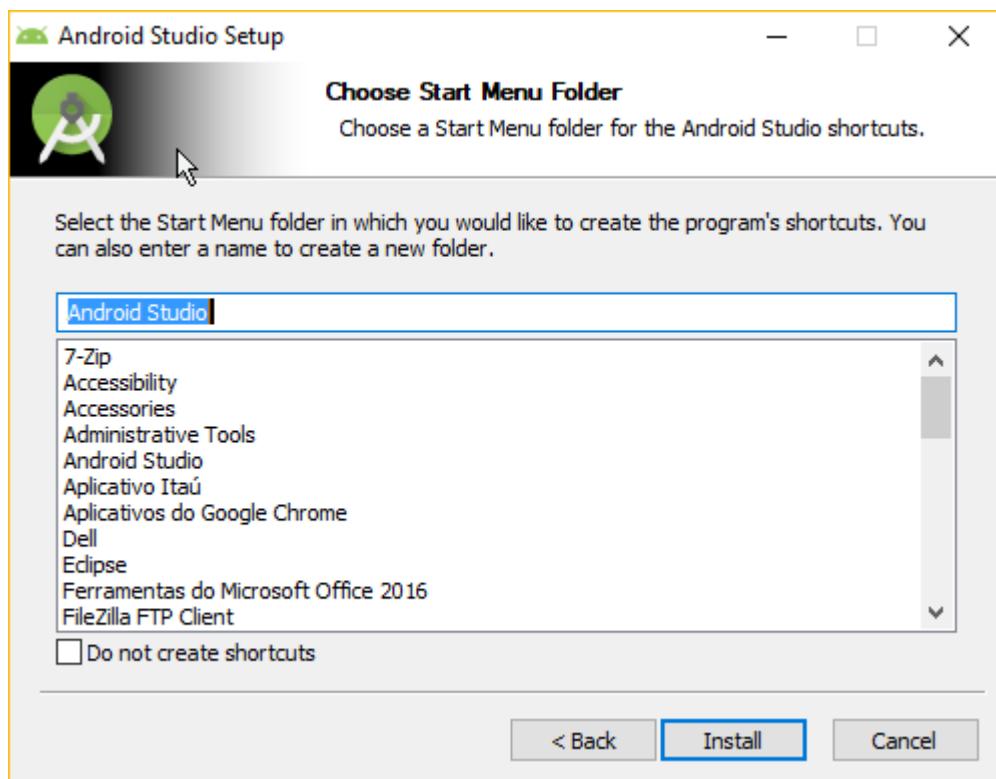
Especifique o local onde o computador deverá instalar o Android Studio

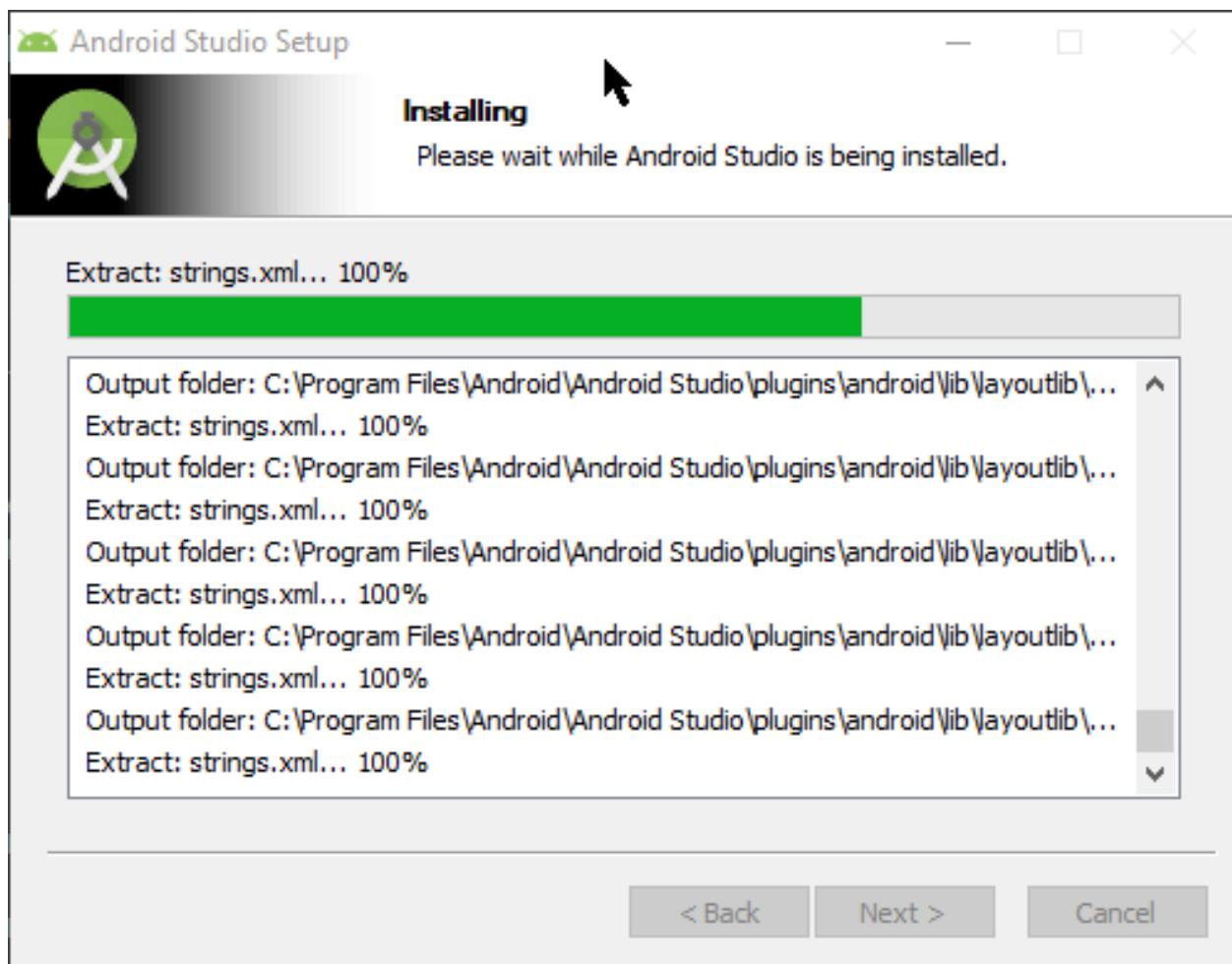


Aconselho manter sempre a recomendação do sistema operacional.

> Se você optou pelo arquivo .zip, descompacte-o em c:\Program Files e execute o arquivo studio64.exe para máquinas com 64bits ou studio.exe para máquinas 32bitis.

Clique em instalar.





Clique em concluir para iniciar o programa **Android Studio**.

Escolha a opção para não importar as preferências anteriores e siga com o passo a passo do próprio Android Studio que é muito simples.

Quando o instalador pedir para escolher um tipo de configuração, aconselho manter a padrão **standard**.

Agora o Android Studio está pronto para ser utilizado.

5.1.2 Passo a Passo para Instalar o Android Studio no sistema Linux/Ubuntu

Assim como outros programas de desenvolvimento você precisará verificar os requisitos mínimos para executar o Android Studio.

Você precisará de:

- Uma distribuição 64bits capaz de rodar aplicações de 32bits.
- Mínimo de 3G RAM - 8GB RAM é o recomendado pelo Google e 1GB para os emuladores caso for utilizar.
- 2GB de espaço livre em disco - recomenda-se 4GB porque a IDE usará 500mb e 1,5GB para os emuladores.
- Até o momento, o Google garante uma boa execução na distribuição Ubuntu 14.04 LTS - Acredito que as outras também devem funcionar bem.

Para baixar o Android Studio você precisará acessar o site oficial do Google conhecido como **Android Developer developer.android.com**. Inclusive, além de **baixar o Android Studio**

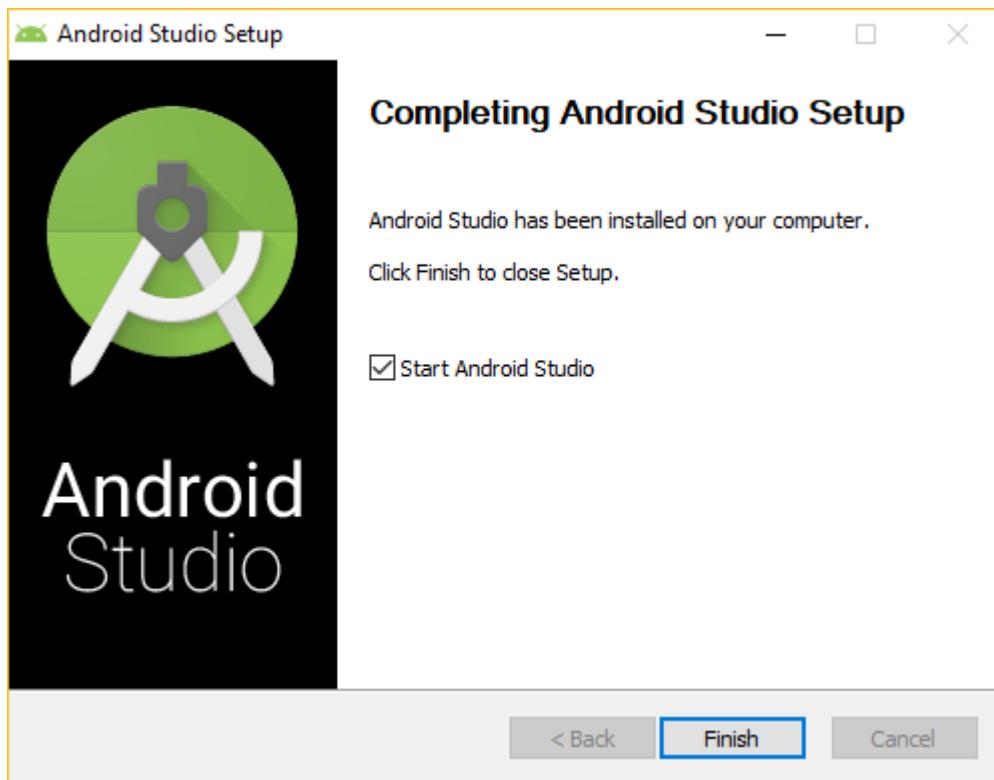


Figure 2: Como Instalar o Android Studio no Windows

Li e concordo com todos os termos e condições acima

DOWNLOAD ANDROID STUDIO FOR LINUX

[android-studio-ide-173.4819257-linux.zip](http://dl.google.com/dl/Android/ide-173.4819257-linux.zip)

Figure 3: Como Instalar o Android Studio no Ubuntu 01

você também encontra toda a documentação da plataforma Android e das suas linguagens de programação.

Ao acessar a página oficial o site identificará que você está acessando-o pelo sistema Linux e irá exibir o download para o sistema (Ubuntu nesse caso).

Aceite os termos e condições para ter acesso ao arquivo de instalação final.

Caso utilize um Ubuntu 64 bits verifique se as dependências seguintes foram instaladas.

Para o Ubuntu:

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

Para o Fedora:

```
sudo yum install zlib.i686 ncurses-libs.i686 bzip2-libs.i686
```

Extraia o arquivo **.zip** e mova-o para uma das seguintes pastas:

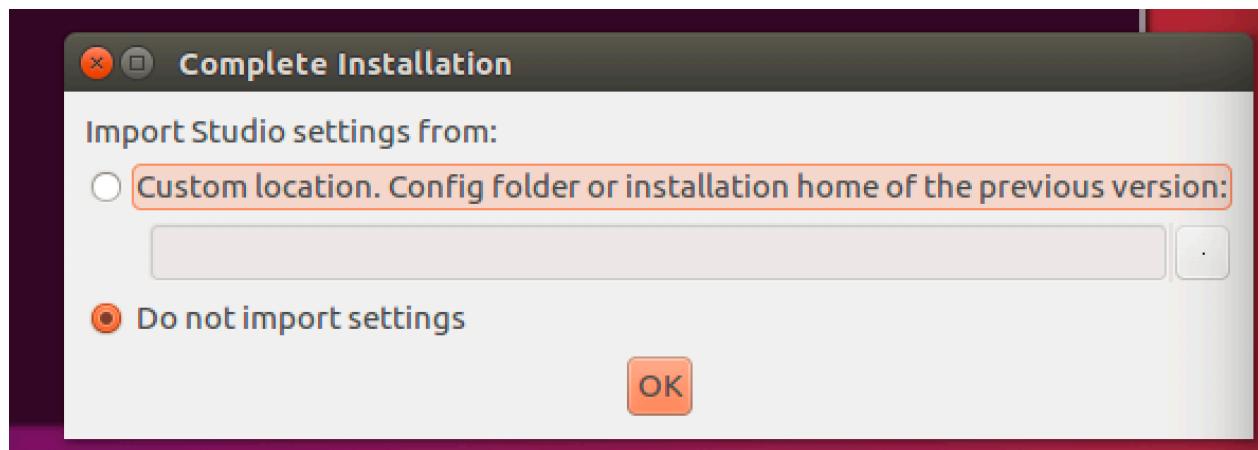
- `/usr/local`: pode ser acessado pelo seu perfil de usuário.
- `/opt/`: pode ser compartilhado entre usuários. Para mover use o comando `mv`: `sudo mv ~/Downloads/android-studio/ /usr/local/`.

Acesse a pasta e execute o inicializador do Android Studio:

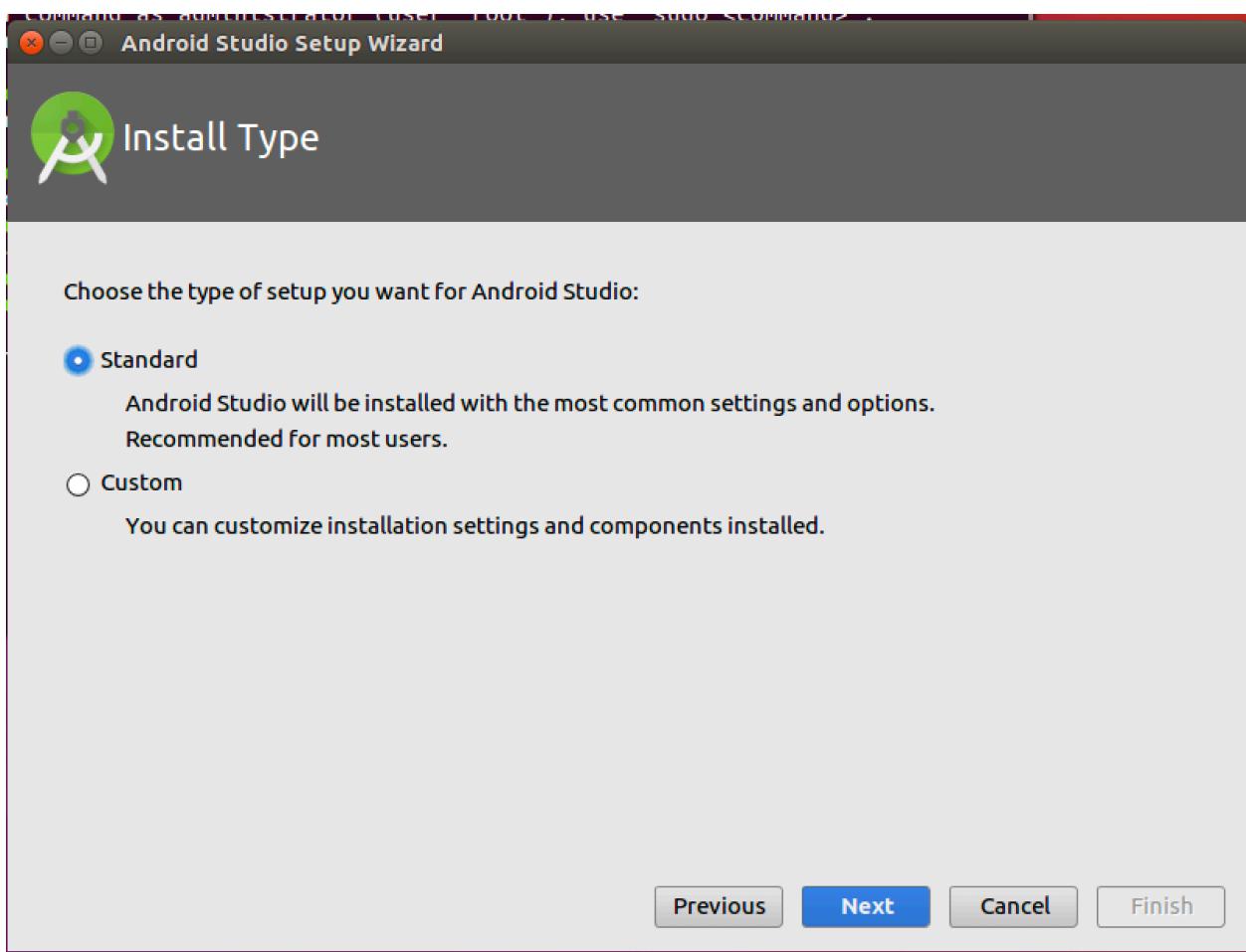
```
cd /usr/local/android-studio/bin seguido de ./studio.sh
tiago@ubuntu16:~$ sudo mv ~/Downloads/android-studio/ /usr/local/
[sudo] password for tiago:
tiago@ubuntu16:~$ ls /usr/local
android-studio bin etc games include lib man sbin share src
tiago@ubuntu16:~$ cd /usr/local/android-studio/bin/
tiago@ubuntu16:/usr/local/android-studio/bin$ ./studio.sh
```

Agora, sempre que precisar iniciar o Android Studio, basta executar este script `studio.sh`.

Escolha a opção para não importar as preferências anteriores e siga com o passo a passo do próprio Android Studio que é muito simples.



Quando o instalador pedir para escolher um tipo de configuração, aconselho manter a padrão **standard**.



Agora o Android Studio está pronto para ser utilizado.

5.1.3 Passo a Passo para Instalar o Android Studio no sistema MacOSX

Assim como outros programas de desenvolvimento você precisará verificar os requisitos mínimos para executar o Android Studio.

Você precisará de:

- Um Mac OS X 10.10 ou superior - Mínimo de 3G RAM - 8GB RAM é o recomendado pelo Google e 1GB para os emuladores caso for utilizar - 2GB de espaço livre em disco - recomenda-se 4GB porque a IDE usará 500mb e 1,5GB para os emuladores

Para baixar o Android Studio você precisará acessar o site oficial do Google conhecido como Android Developer developer.android.com. Inclusive, além de **baixar o Android Studio** você também encontra toda a documentação da plataforma Android e das suas linguagens de programação.

Ao acessar a página oficial o site identificará que você está acessando-o pelo Mac OSX e irá exibir o download para o sistema Apple.

Aceite os termos e condições para ter acesso ao arquivo de instalação final.

O arquivo **.dmg** do Mac OSX é um tipo de imagem de disco e sua instalação é a mais simples de todos os sistemas operacionais.

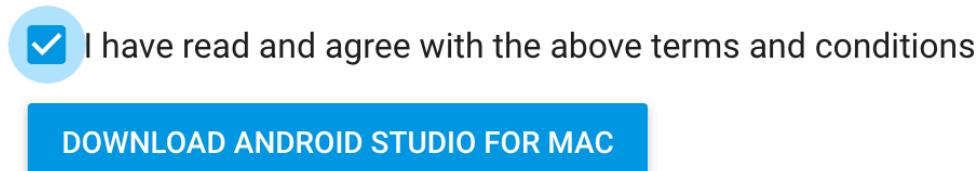


Figure 4: Como Instalar o Android Studio no Mac 01

Clique no arquivo .dmg e arraste o ícone do android studio para a pasta Applications do Mac.



Escolha a opção para não importar as preferências anteriores e siga com o passo a passo do próprio Android Studio que é muito simples.

Quando o instalador pedir para escolher um tipo de configuração, aconselho manter a padrão **standard**.

Agora o Android Studio está pronto para ser utilizado.

5.2 Android SDK

Quando desenvolvemos para Android ou qualquer outra linguagem precisamos usar um SDK (Software Development Kit - Kit de Desenvolvimento de Software).

O Android SDK possui códigos para criar aplicativos em versões específicas para o Android. Isso porque alguns dispositivos são mais novos outros mais velhos. Por isso, utilize sempre a última versão do SDK para compilar o seu programa, corrigir bugs e para desfrutar de novas funcionalidades modernas.

Além disso, o SDK também possui ferramentas de construção (build tools) para compilar e transformar seu código em um arquivo binário que pode ser entendido pelo smartphone.

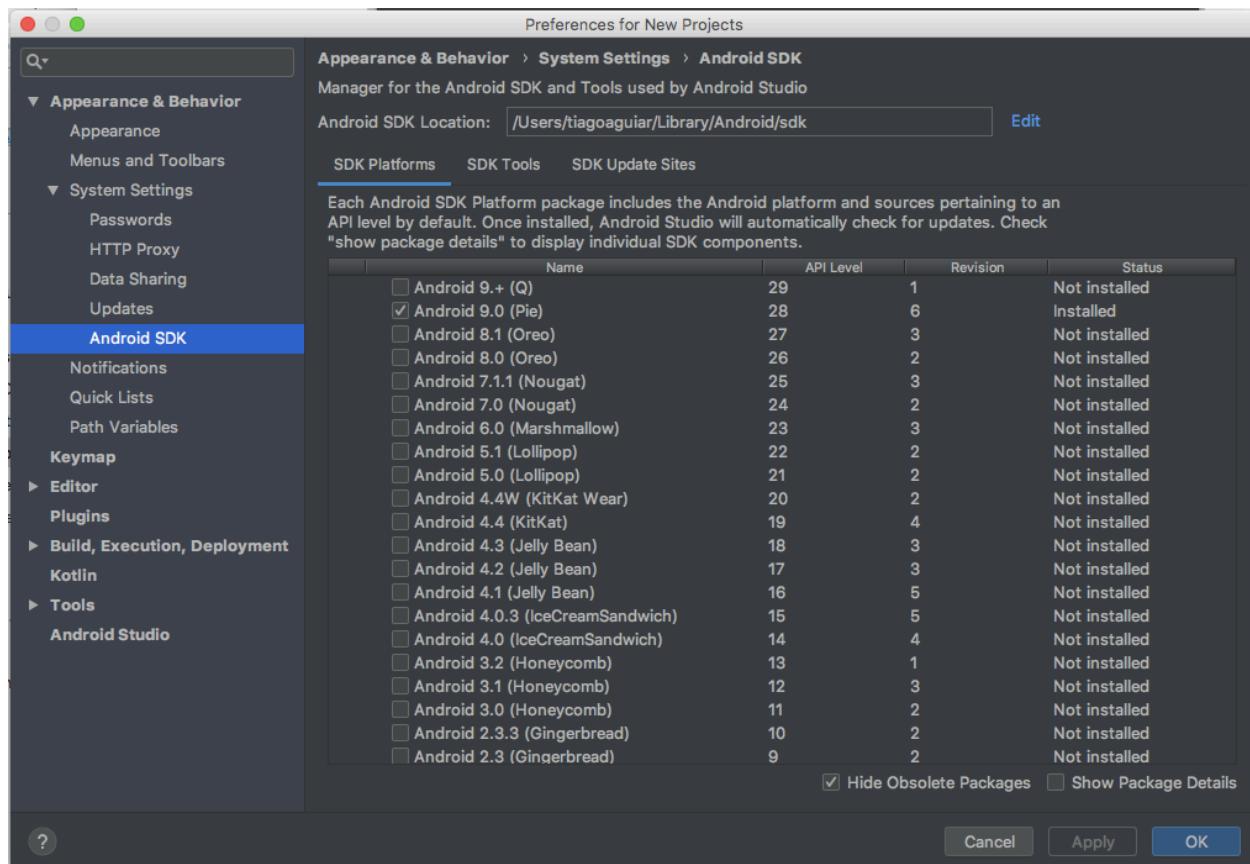
Para manter o seu SDK atualizado e as ferramentas de construção (build tools) você precisará acessar o **SDK Manager**.

Abra o Android Studio e selecione a opção **Configure > SDK Manager**:



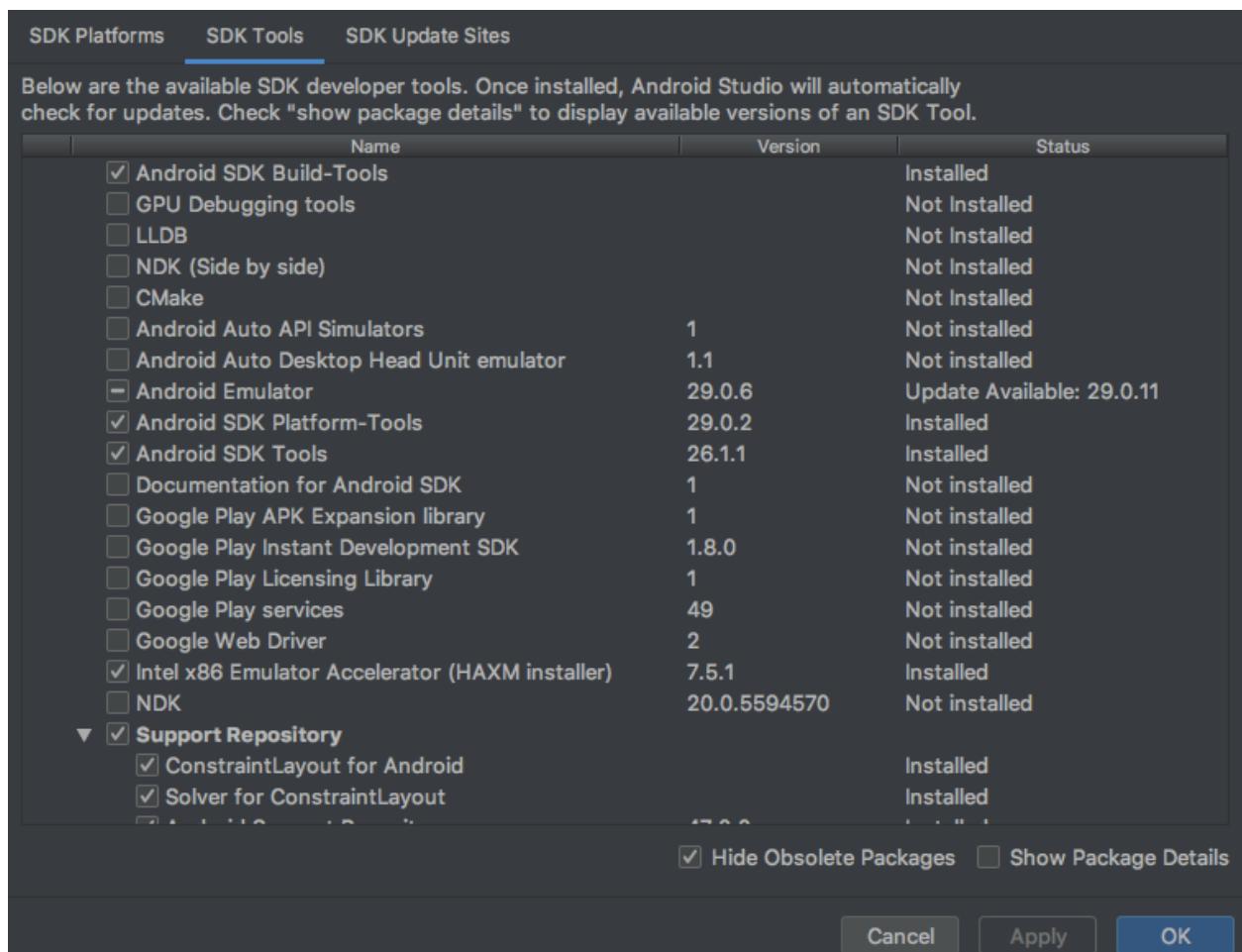
Verifique se a versão mais recente do SDK Android já está instalado.

Quando precisarmos testar uma versão antiga do aplicativo como o KitKat por exemplo, precisamos baixar essa versão legada para realizar os testes.



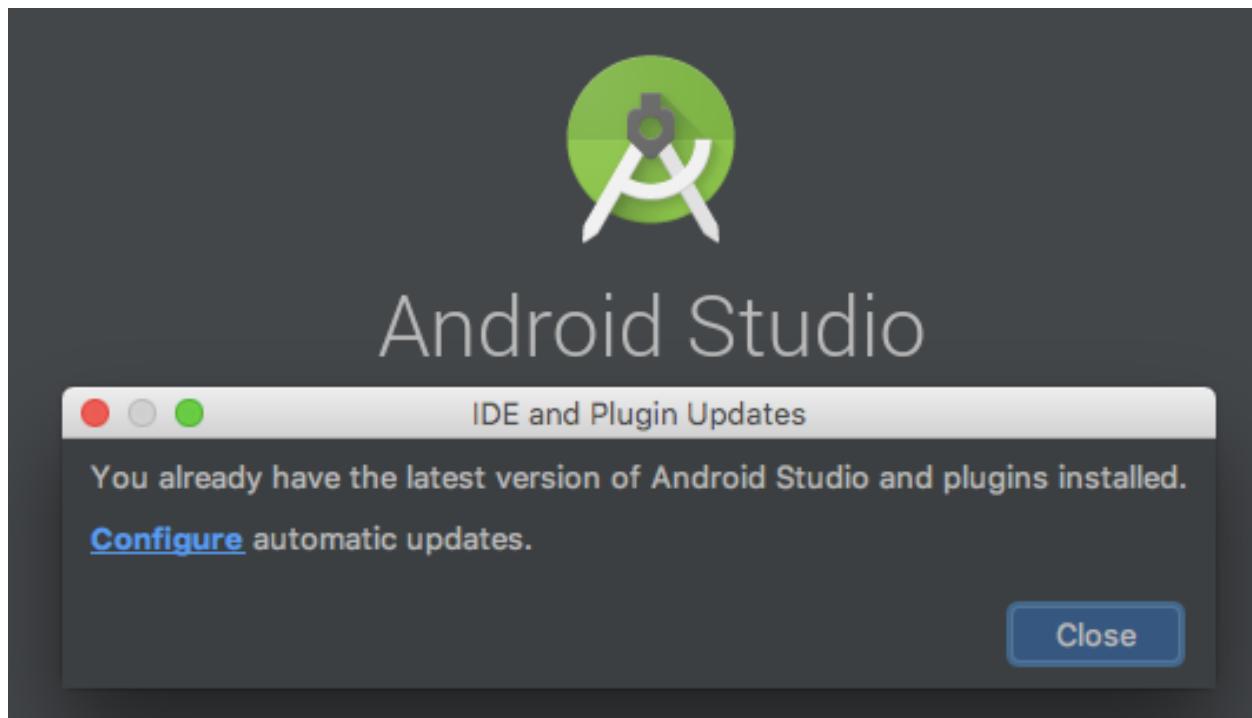
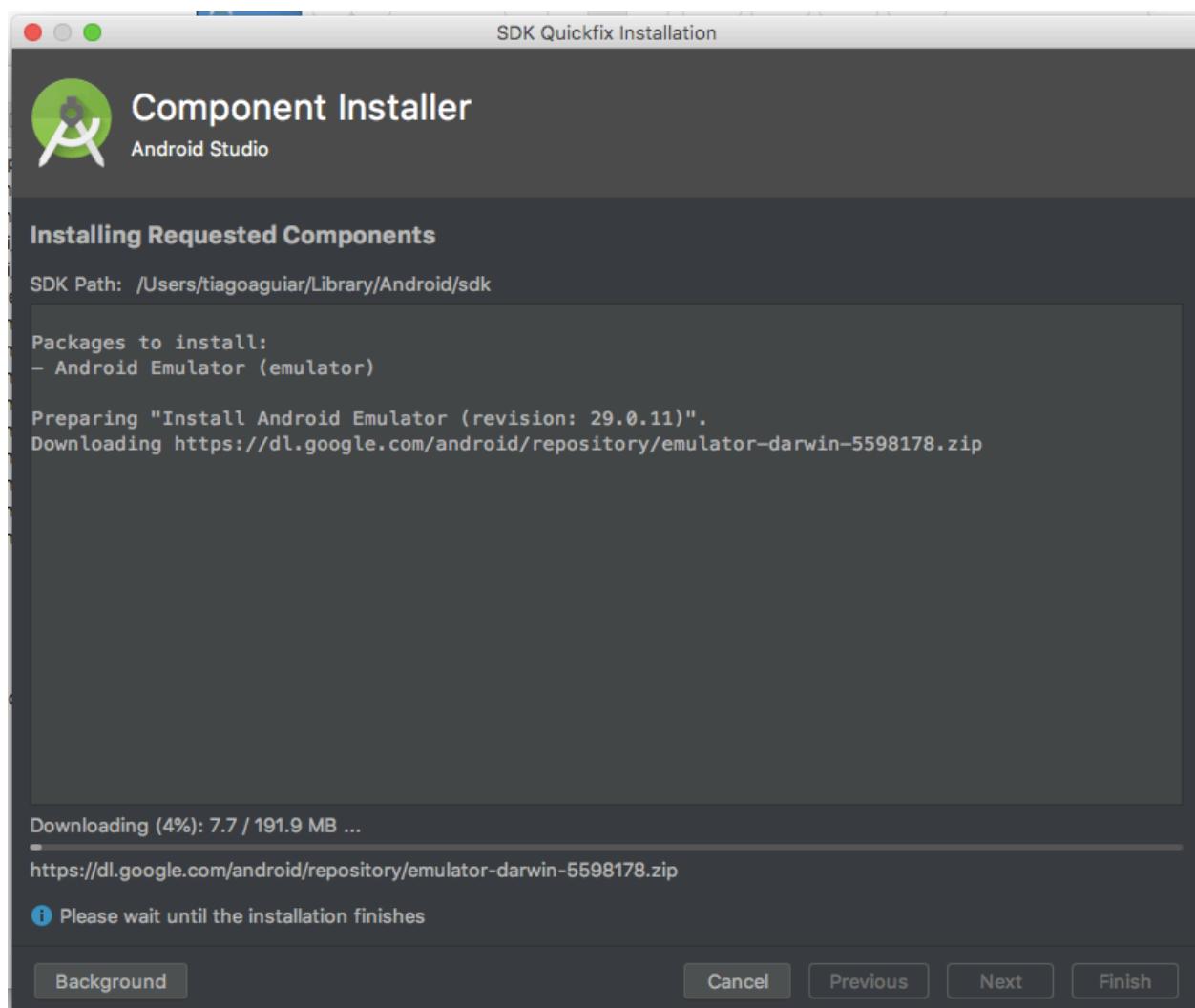
Selecione o **SDK Tools** e garanta que as seguintes ferramentas estão instaladas e atualizadas:

- Android SDK Build-Tools
- Android SDK Platform-Tools
- Android SDK Tools
- Android Emulator



Para garantir que o próprio Android Studio está atualizado, clique em **Configure > Check for Updates** e baixe as atualizações se necessário.





6 Capítulo 2: Criando o Primeiro Aplicativo Android

Partindo do princípio que o Android Studio está atualizado, vamos criar nosso primeiro projeto Android usando os Templates pré-definidos pela IDE.

Abra o Android Studio e clique em **Start a new Android Studio project**. A partir deste passo, vamos construir um projeto Android com diversos arquivos gerados pelo Android Studio.

Aos poucos irei detalhar a função de cada arquivo e cada pasta da estrutura do projeto.

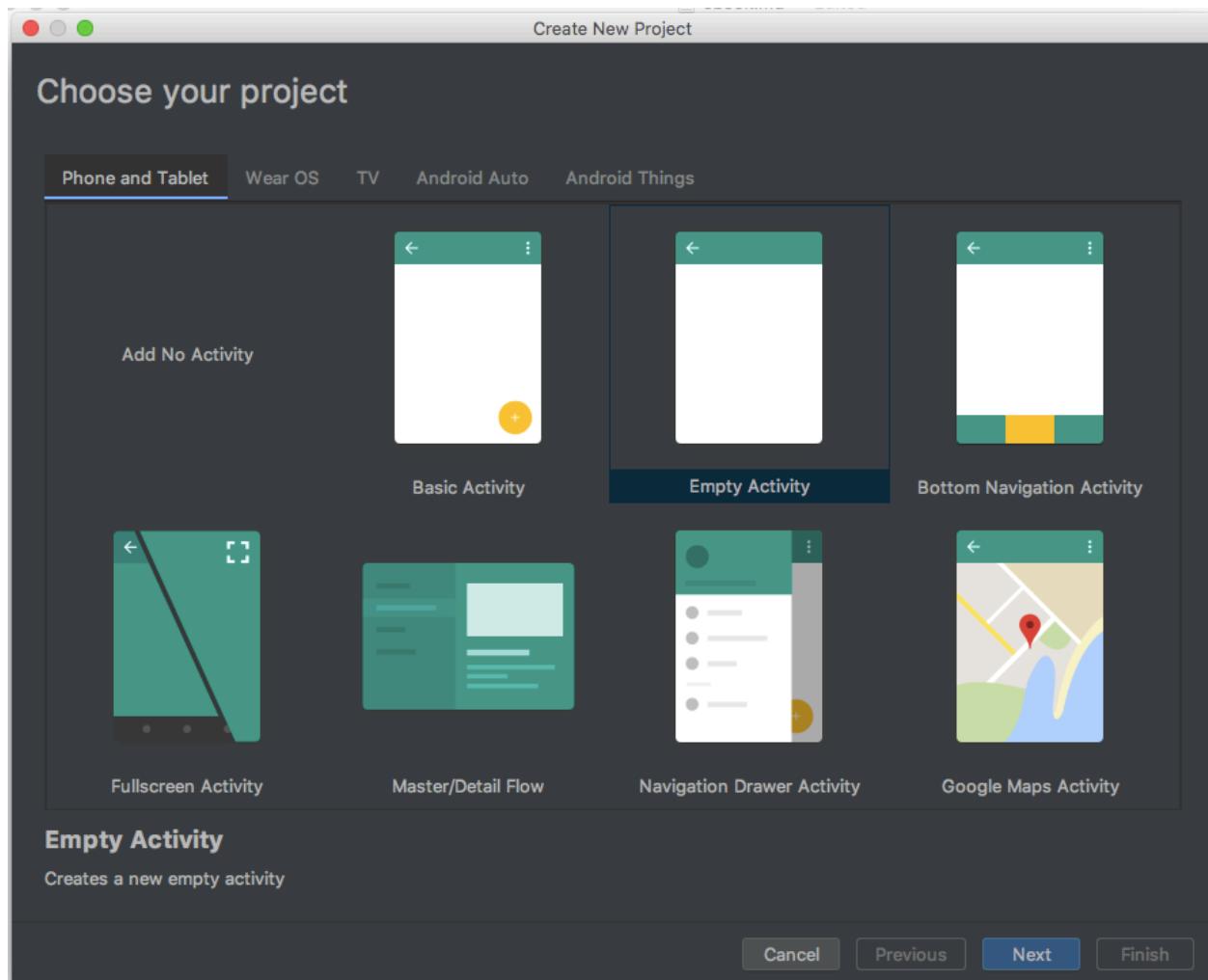
6.1 Templates de Projetos

Quando começamos um projeto do zero, temos a opção de escolher templates iniciais para já ter alguns códigos prontos como por exemplo, um aplicativo com o Google Maps incorporado.

Contudo, vamos começar pelo simples! Cada tela do Android é gerenciada por um recurso que chamamos de **Activity** ou Atividade em português. Como o aplicativo precisa iniciar em uma tela principal, precisamos criar uma Activity com um Layout para ela se conectar.

Vamos escolher a opção **Empty Activity** para que o Android Studio crie uma Atividade vazia onde possamos dar continuidade na construção desse layout e dessa activity.

Clique em Next:



Agora precisamos definir um nome para o projeto. No campo **Name** digite o nome do aplicativo: **PrimeiroApp**.

O campo **Package Name** é utilizado para definir um identificador único para o aplicativo e que será usado para identificar seu aplicativo na loja. Por convenção no ambiente de desenvolvimento de software os nomes de pacotes gerados a partir de domínios são escritos de trás para frente. Ou seja, um aplicativo do Google por exemplo pode ter um pacote identificado como `com.google.gmail`.

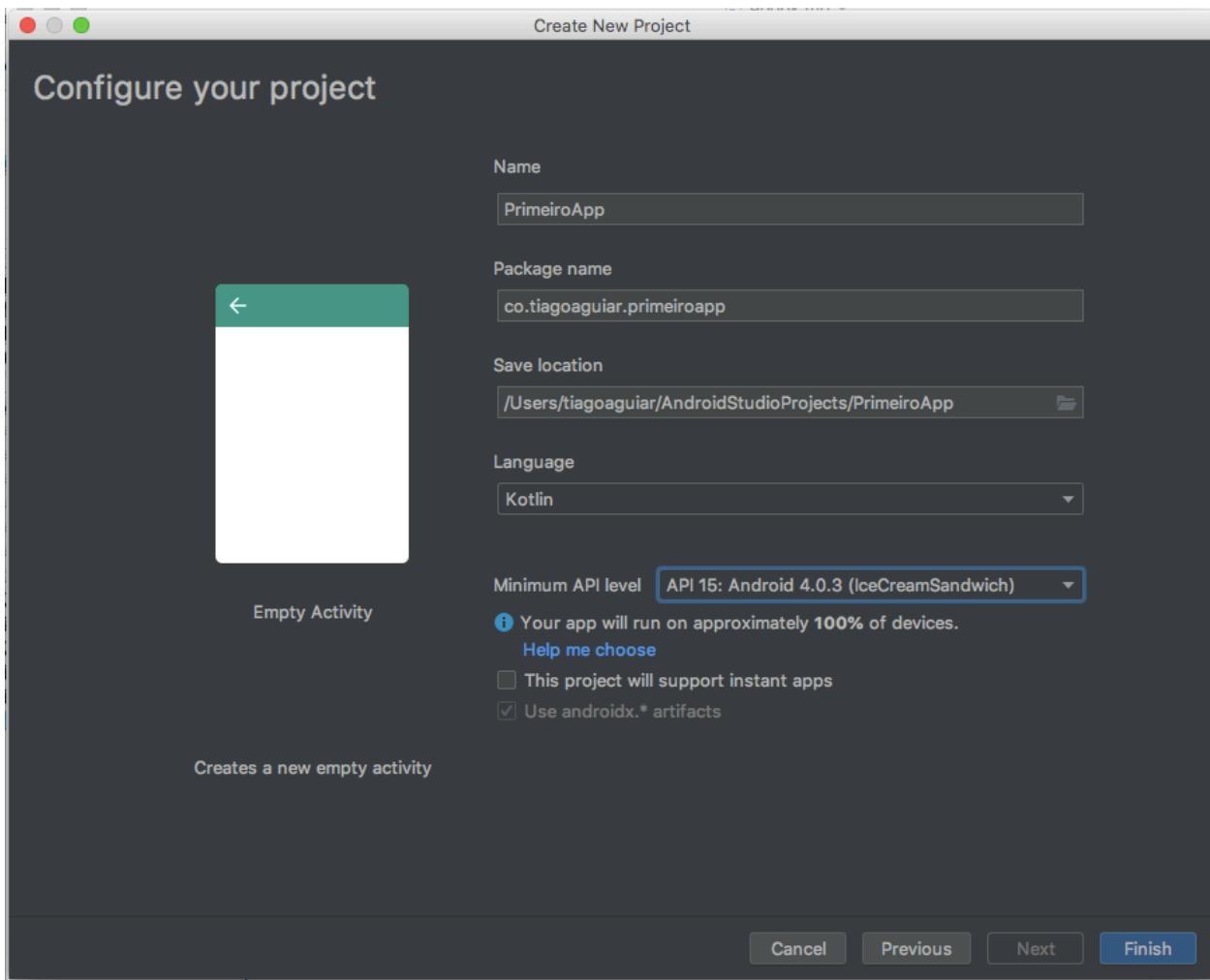
Neste caso, defina o domínio do meu website que será utilizado para gerar o pacote do aplicativo.

Você pode ver o pacote logo abaixo: `co.tiagoaguiar.primeiroapp`.

O campo **Save location** indica o diretório onde o projeto será criado no seu computador. Escolha uma pasta fácil para você organizar os futuros projetos.

Ao selecionar **Language** garantimos que o projeto será construído usando a linguagem Java ou Kotlin. Selecione a opção Kotlin para construir um aplicativo moderno com essa linguagem de programação oficial do Google.

Clique em **Next** para continuar a construção do projeto.



A opção **This project will support instant apps** não vamos usar por ser algo mais avançado neste momento. Já a opção **Use androidx.* artifacts** já estará selecionada se você estiver usando um SDK maior ou igual ao **API 29**. Caso este item esteja disponível para selecionar, selecione-o.

Agora vamos escolher a versão mínima do Android que o aplicativo vai funcionar. Quanto mais antiga, mais usuários você poderá atingir.

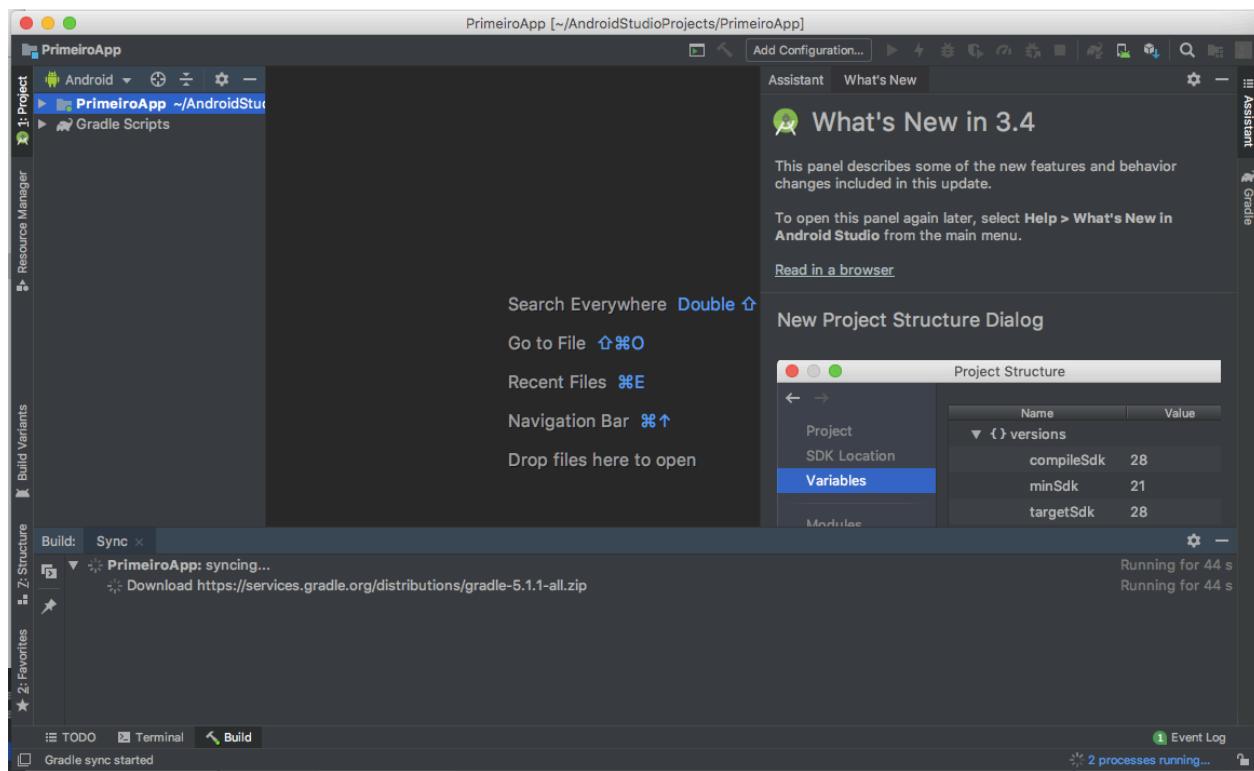
O próprio Android Studio mostra o percentual de alcance daquela versão.

Também é possível escolher se o aplicativo será feito para:

- Smartphones e Tablets
- Wear
- TV
- Android Auto
- Android Things

Como nosso foco é construir aplicativos para Smartphones, vamos deixar somente a primeira opção selecionada e manter a versão escolhida pelo Android Studio. No momento que escrevo esse material a versão de API 15 é a versão mínima recomendada.

Clique em **Finish** para o Android Studio gerar a estrutura de pastas e arquivos do primeiro aplicativo.



6.2 Configurando um Emulador

Existe duas maneiras de testar o seu aplicativo: através de um emulador ou de um smartphone real.

Para facilitar o dia a dia e evitar que você sempre precise estar com seu smartphone por perto, vamos criar um emulador usando o Android Studio.

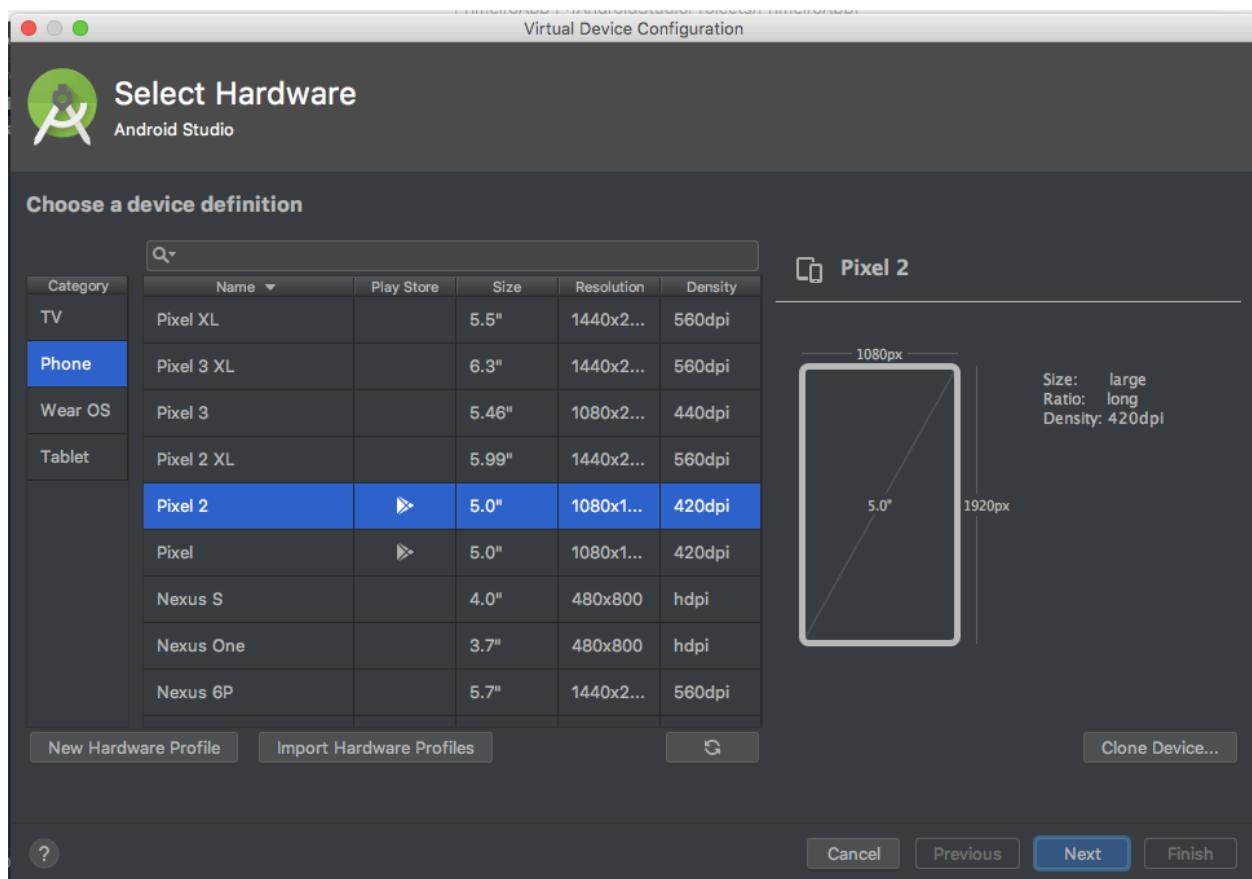
Clique no ícone superior a direita que indica o **AVD Manager** - Android Virtual Device Manager.



Agora clique em **Create Virtual Device....**

Existe diversos smartphones de tamanhos diferentes para emularmos. Algumas versões já acompanha o aplicativo Google Play e outros serviços do Google. Dê preferência a estas versões.

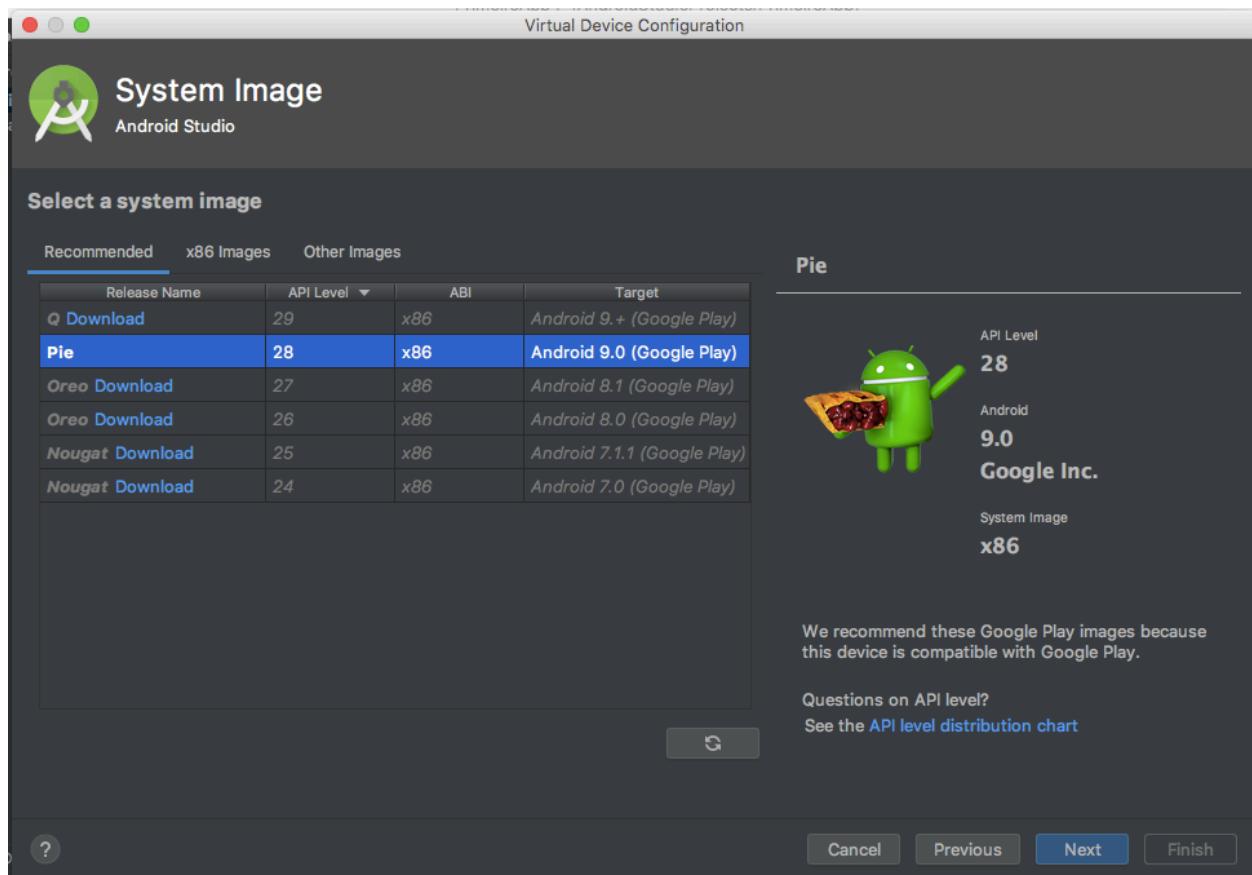
Dito isto, vamos escolher o **Pixel 2** que é a versão mais recente até o momento. Caso haja outra versão mais recente com o Google Play pode escolher - desde que você tenha espaço na HD para baixar.



No próximo passo você deve selecionar uma *system image* que representa a versão do Android. Caso você queira testar em uma versão onde ainda não tenha a *system image* e o SDK, clique em download para baixar a versão respectivamente.

No nosso exemplo, vamos baixar a versão **Pie**.

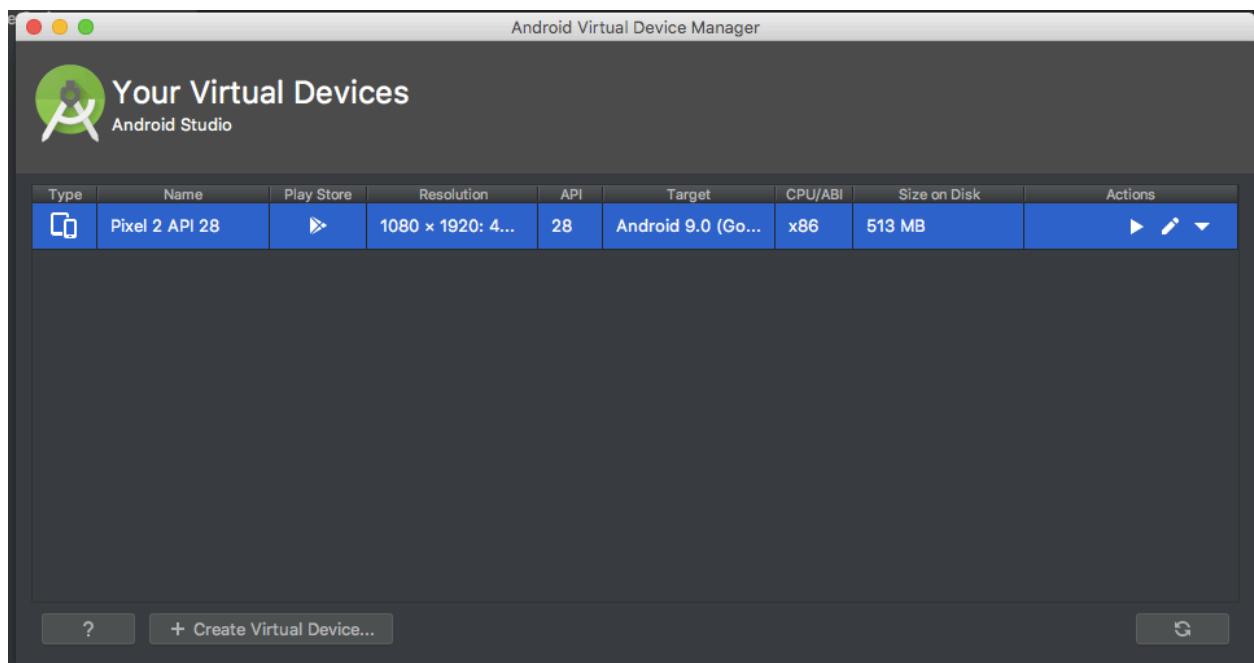
Os emuladores são bem grandes em termos de megabytes. Logo, pode demorar um tempinho dependendo da sua conexão com a internet.



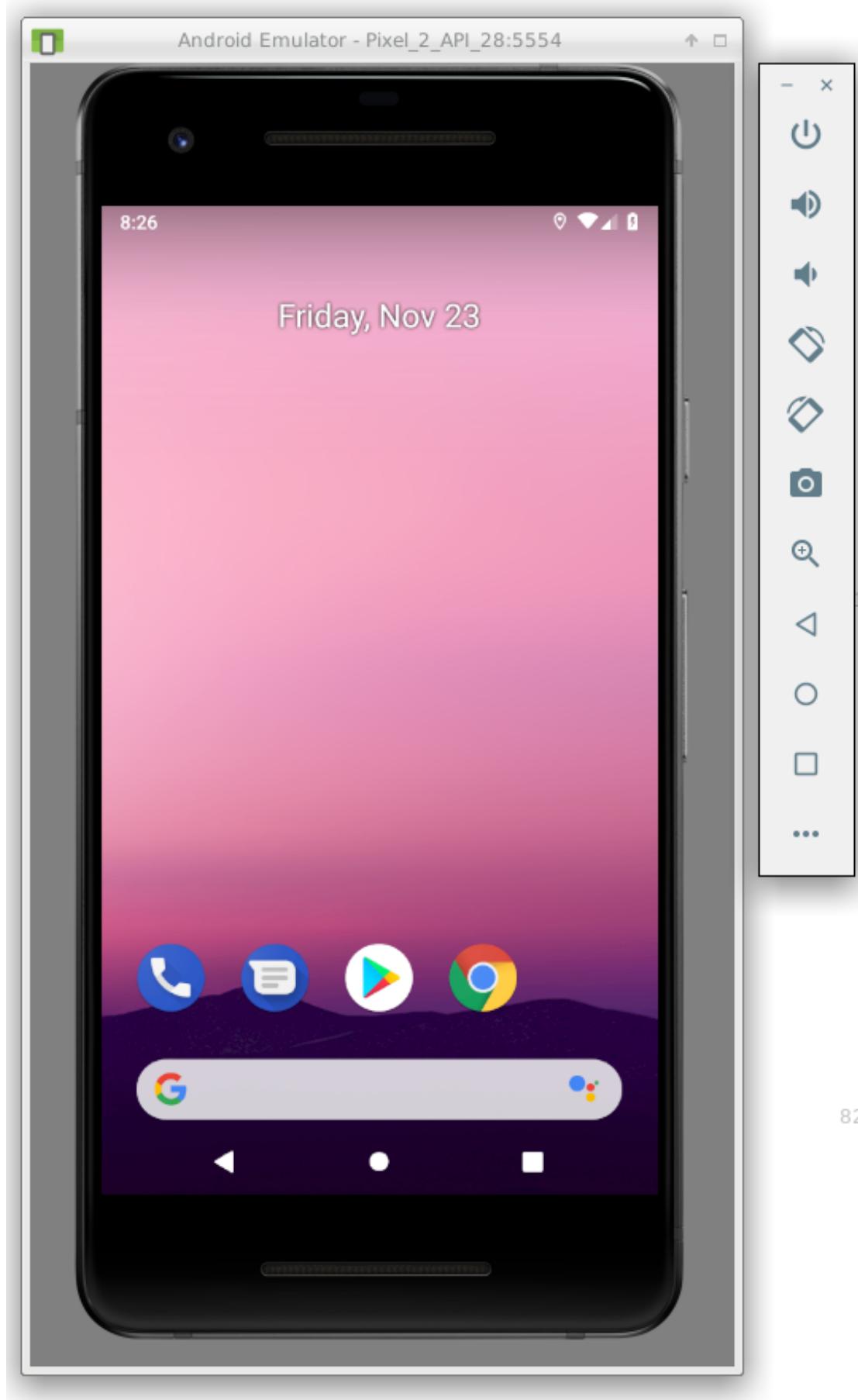
Nesta tela vamos definir um nome para nosso emulador. Geralmente mantenho a opção do próprio Android Studio ou altero de acordo com os testes que estou fazendo.



Clique em **Finish** e o emulador começará a ser construído.



Com o Emulador pronto, clique no botão de Play para iniciá-lo. Geralmente, na primeira vez que executarmos, o emulador poderá demorar um pouco. Isso também pode variar dependendo do processador e de outras configurações do seu computador.



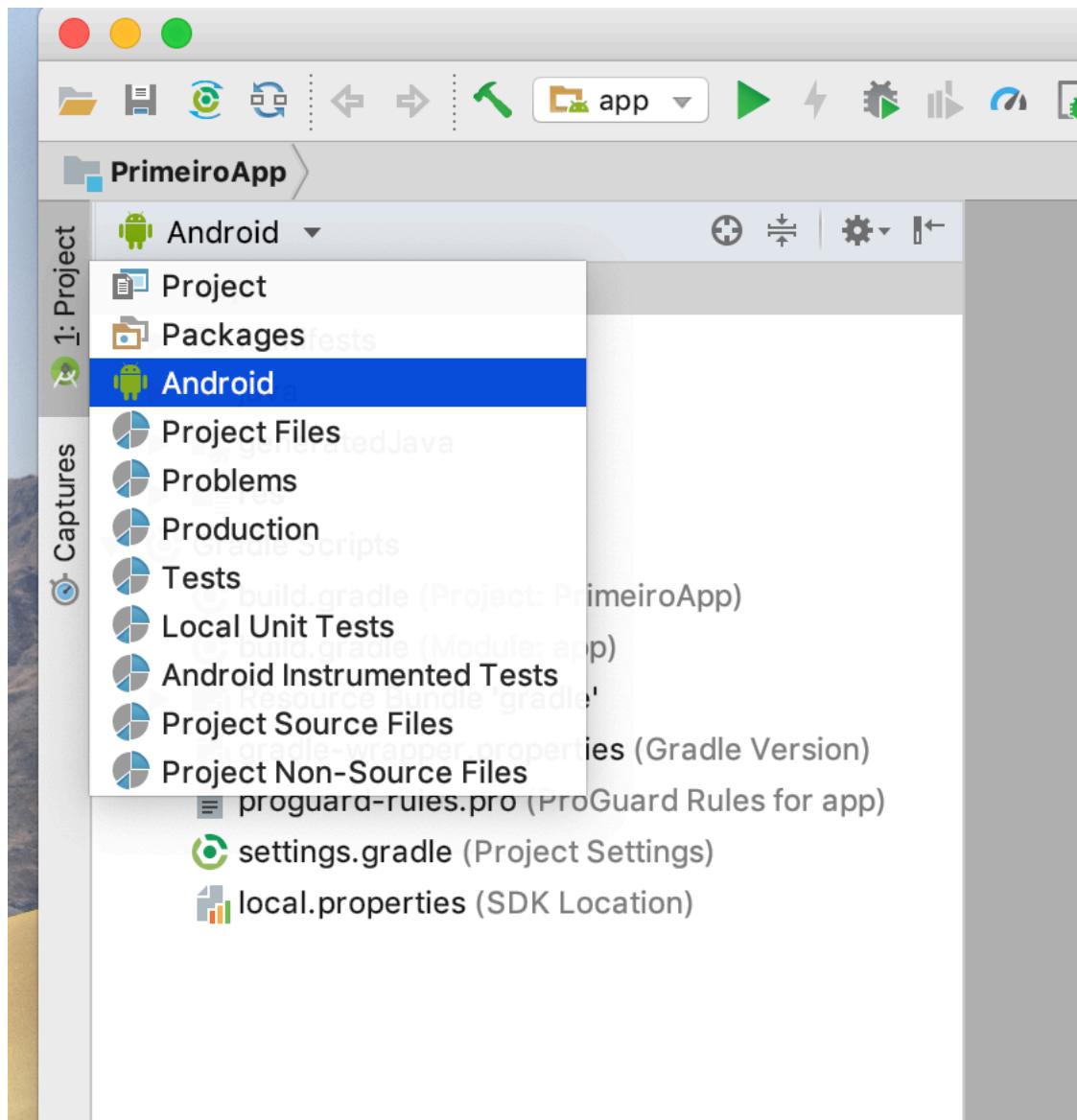
Seu emulador já está pronto para ser utilizado como dispositivo de testes do seu aplicativo!

6.3 Estrutura de Pastas

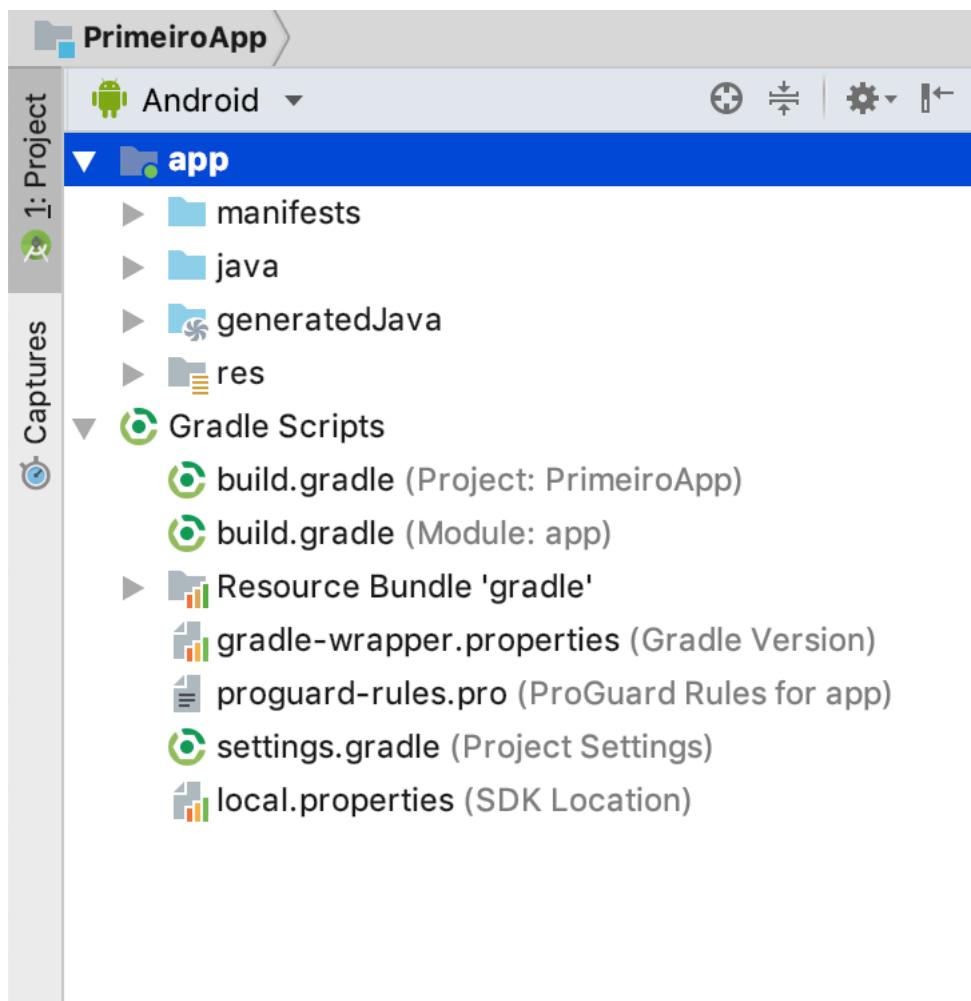
Quando criamos um projeto do zero no Android Studio a ferramenta constrói uma série de arquivos e pastas que define como o aplicativo vai ser organizado durante o desenvolvimento.

Basicamente na visão **Android** é exibido somente o módulo **app**. Esse módulo é onde escreveremos nossos arquivos de lógica, bem como os layouts das telas do aplicativo.

Mudando a perspectiva para **Project** você verá uma outra visão geral de todos os arquivos - arquivos que não são necessariamente parte do módulo app.



De volta na visão **Android** no módulo app, podemos ver algumas pastas e arquivos onde irei descrever em detalhes logo abaixo:



6.3.1 Manifests

O diretório **Manifests** possui apenas um arquivo chamado de **AndroidManifest.xml**. Esse arquivo é um arquivo de configuração que diz como nosso aplicativo vai funcionar. Ou seja, ele diz quais telas ele terá (Activity), as permissões de acesso como Internet, acesso a agenda de contatos, imagens do lançador e etc.

Além disso, ele define qual será a tela inicial do aplicativo - o **main** do nosso projeto.

Outros recursos avançados podem ser definidos ali como: o método de busca para o seu aplicativo aparecer no google buscador, entre outros.

Em resumo, esse é um arquivo no formato **XML** onde definimos como o aplicativo deve funcionar e quais recursos usar.

Se você ainda não conhece o formato xml, basicamente ele trabalha no formato de **<TAG />** semelhante a HTML sendo que, algumas tags podem ter nós “filhos” e atributos.

Abra o `AndroidManifest.xml` e dê uma olhada na estrutura que esse arquivo é escrito.

Por enquanto, não se preocupe o que cada linha escrita nesse arquivo significa. Mais à frente, iremos detalhar o que ele faz. Por hora, vamos focar na estrutura de pastas.

6.3.2 Java

Esse é o diretório onde armazenamos nossos arquivos com a lógica do aplicativo. Por mais que esteja escrito **java** é nele que se encontra os arquivos tanto Java quanto Kotlin, muito provavelmente ainda está assim porque Kotlin é a nova linguagem oficial para criação de aplicativos Android. E ainda há projeto bilhões de projetos sendo feitos em Java.

Dentro desse diretório temos vários outros diretórios que chamamos de **pacotes**.

Cada pacote pode ter 1 ou N arquivos que possuem responsabilidades únicas na lógica do aplicativo. Por exemplo, podemos ter um pacote **co.tiagoaguiar.primeiroapp.view** onde colocaremos arquivos referentes a regras de tela e visualização do usuário como Botões, Campos de Textos, etc.

Existem N maneiras de se organizar os arquivos que podem facilitar (e muito) a manutenção futura do seu aplicativo. Novamente, não se preocupe com pacotes agora, apenas entenda que o diretório **java** será o seu diretório principal para codificar.

Além do pacote raiz principal **co.tiagoaguiar.primeiroapp** ainda temos mais 2 pacotes que se encontram no diretório de tests. Esses diretórios são exclusivos para testes de lógica de programação e testes de interface gráfica, uma metodologia avançada para desenvolvimento de software no geral.

6.3.3 generatedJava

Esse diretório é construído automaticamente pela IDE. Ou seja, NÃO mexa nele para não “quebrar” seu aplicativo. Ali contém arquivos onde o compilador se encarrega de gerar.

Em resumo, tudo vai funcionar se não alterarmos esse arquivo. Mas você pode abri-los se estiver curioso sobre o que foi gerado ;)

6.3.4 Res

A pasta **res** é a nossa pasta de **recursos - resources**. Tudo que é layout, cores, mídias como fotos ou vídeos e temas são tratados como recursos.

É ali onde criamos nossas telas e construímos animações por exemplo. Tudo na estrutura XML. Note que boa parte do seu conhecimento deve ser direcionado a entender a lógica do aplicativo + recursos estáticos de layout em XML. Acredito que você passa 80% do seu tempo escrevendo nessas estruturas.

Dentro de **res** você encontra já 4 subpastas:

- **drawable**: Armazena recursos de imagens e ícones do aplicativo
- **layout**: Armazena as telas e layouts que podem ser incluídos em outros layouts
- **mipmap**: Armazena ícones do lançador (aqueles imagens que ficam na Home do Smartphone)

- values: Values podem ter arquivos como: **colors.xml**, **strings.xml**, **styles.xml**. Todos esses recursos de textos ou cores podem ser acessados em qualquer parte do código. Falaremos mais sobre isso.

6.3.5 Gradle Scripts

Aqui se encontra arquivos de construção (que chamamos de scripts) que rodam para compilar o projeto.

Você deve se preocupar apenas com os arquivos **build.gradle**. Existem 2 tipos: 1 para o projeto como um todo e outro para o módulo app. São nesses arquivos onde definimos a versão do SDK, o pacote que aparecerá na Google Play, as dependências de outras bibliotecas (que são códigos de terceiros que podemos usar no nosso projeto), etc.

Assim como o **AndroidManifest.xml** define como o aplicativo deve funcionar, o **build.gradle** define como o aplicativo deve ser construído e que recursos ele vai precisar para ser construído.

Concluindo, é assim que o Android Studio organiza o seu projeto. Eu sei que pode parecer assustador para muitos a quantidade de arquivos, mas não se preocupe pois quando você por em prática o que vier aprendendo nesse material tudo ficará claro.

Agora que já temos um template inicial pronto para usar, emuladores configurados e uma visão geral da estrutura de pastas. Vamos começar a codificar nosso primeiro aplicativo Android e executá-lo no emulador.

6.4 Hello World

No mundo da computação o primeiro programa que escrevemos se chama **Hello World**. Para testar nosso “hello world” no Android, basta clicar no botão verde de Play na barra de ícones do Android Studio e escolher um Emulador.

Se tudo ocorrer bem, você verá seu primeiro aplicativo com a mensagem de texto *Hello World* no centro da tela.

Agora para deixar as coisas um pouco mais interessantes, vamos definir um plano de ação de como nosso aplicativo vai funcionar. Depois vamos “desenhar” a tela com componentes do SDK e por fim, escrever a lógica nos arquivos Kotlin.

6.4.1 Que Tipo de Aplicativo Vamos Estar Criando?

Nosso primeiro aplicativo terá como principal função “responder” a eventuais perguntas que fizermos a ele. Ou seja, ele irá se parecer com a bola mágica 8 onde chacoalhamos ela para receber uma “resposta”.

Basicamente ela irá responder **Sim, Não Conte com Isso e Talvez**.

Você já deve ter visto essa bola mágica em filmes como o Toy Story.



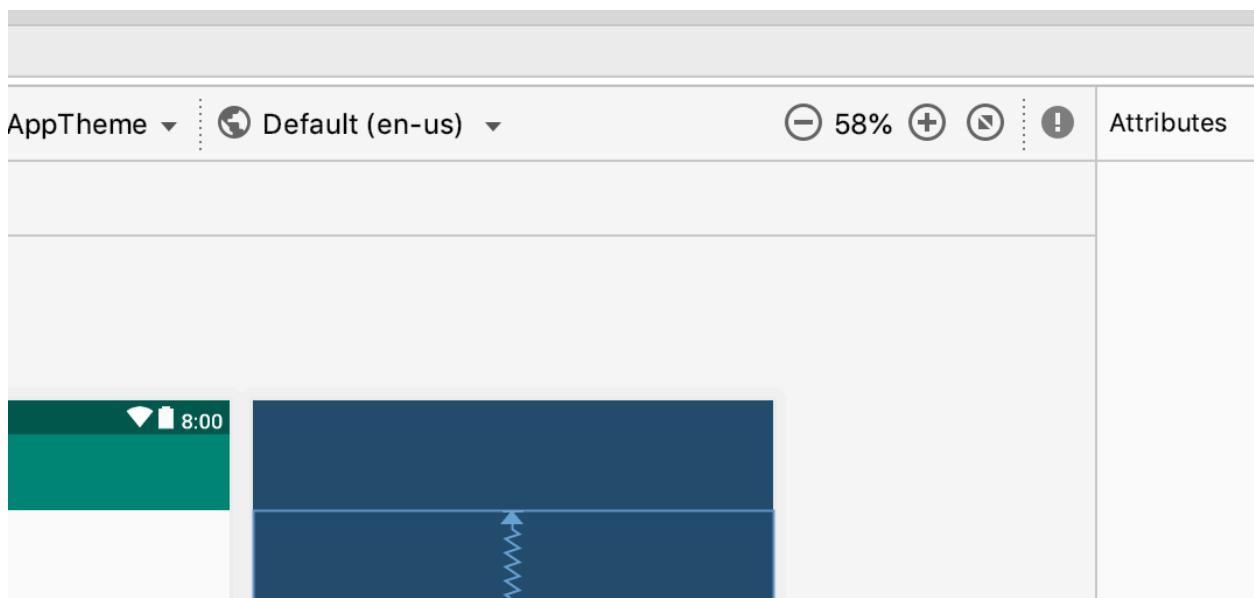


Figure 5: k007

7 Capítulo 3: Codificando o Primeiro Aplicativo Android

A primeira coisa a se fazer é abrir o arquivo `act_main.xml` que se encontra no diretório `res/layout/`. Um editor visual de layout irá abrir.

Utilize o `zoomIn` que se encontra no canto superior direito para melhorar a visão do editor. Selecione o campo de texto padrão *Hello World* e apague-o com **Delete**.

Agora selecione no painel ao lado **TextView** e arraste-o para dentro da tela do aplicativo. Esse componente nos ajuda a criar labels - rótulos para exibir textos.

O próximo passo é muito importante. É aqui onde vamos definir as posições na tela de onde cada elemento irá aparecer.

No Android, os elementos são tratados por “regras” definidas em cada elemento e seu *pai*. Ou seja, um `TextView` pode depender de um `Button` ou de outro elemento para se posicionar como o container principal que é a própria tela do celular.

Como estamos colocando nosso primeiro elemento nós iremos fazer com que as posições laterais, superior, direita e inferior se dimensione de acordo com o tamanho total da tela. Este recurso se chama **constraint**.

Para dizermos ao aplicativo que o `TextView` será alinhado no centro da tela, devemos criar as constraints. Segure e arraste o click nas “bolinhas” que aparecem nas extremidades do componente.

No exemplo abaixo, usei um `EditText` (um editor de texto). Mas faça isso com um `TextView`.

Ao segurar e arrastar a parte superior, esquerda e direita teremos algo parecido com isso.

Como falei, fiz um teste com um `EditText`, seu `TextView` deverá ficar da seguinte forma:

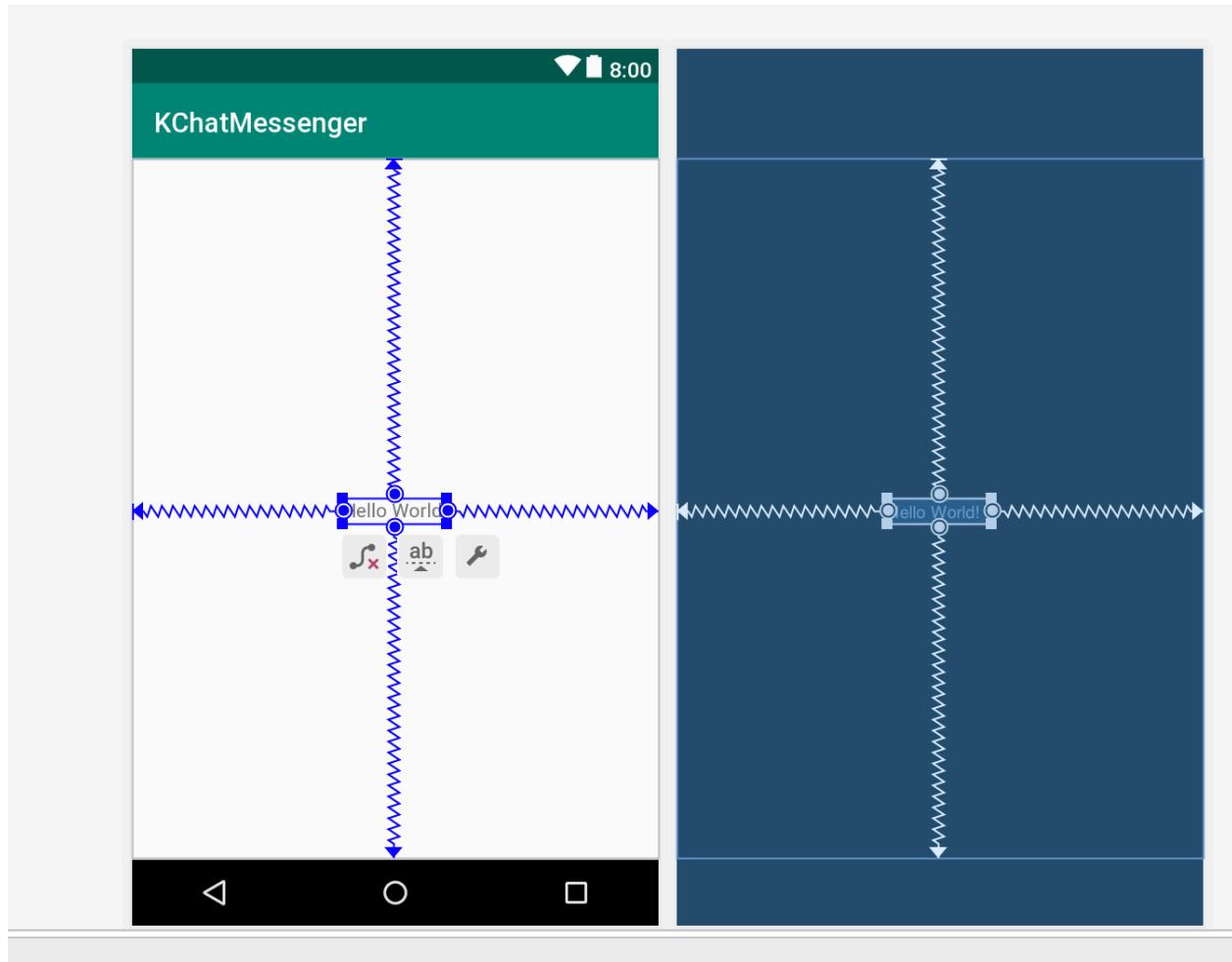


Figure 6: k008

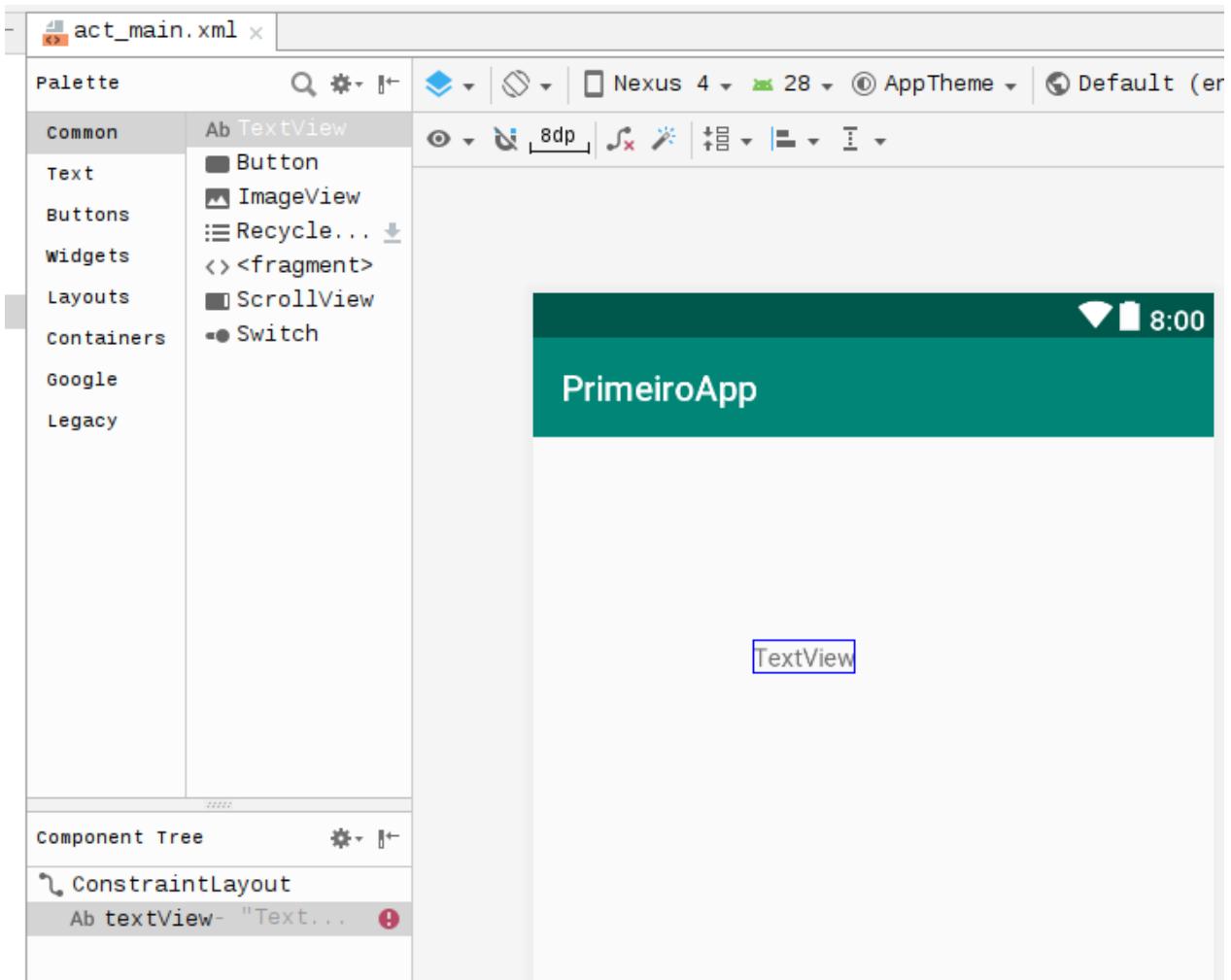


Figure 7: k009

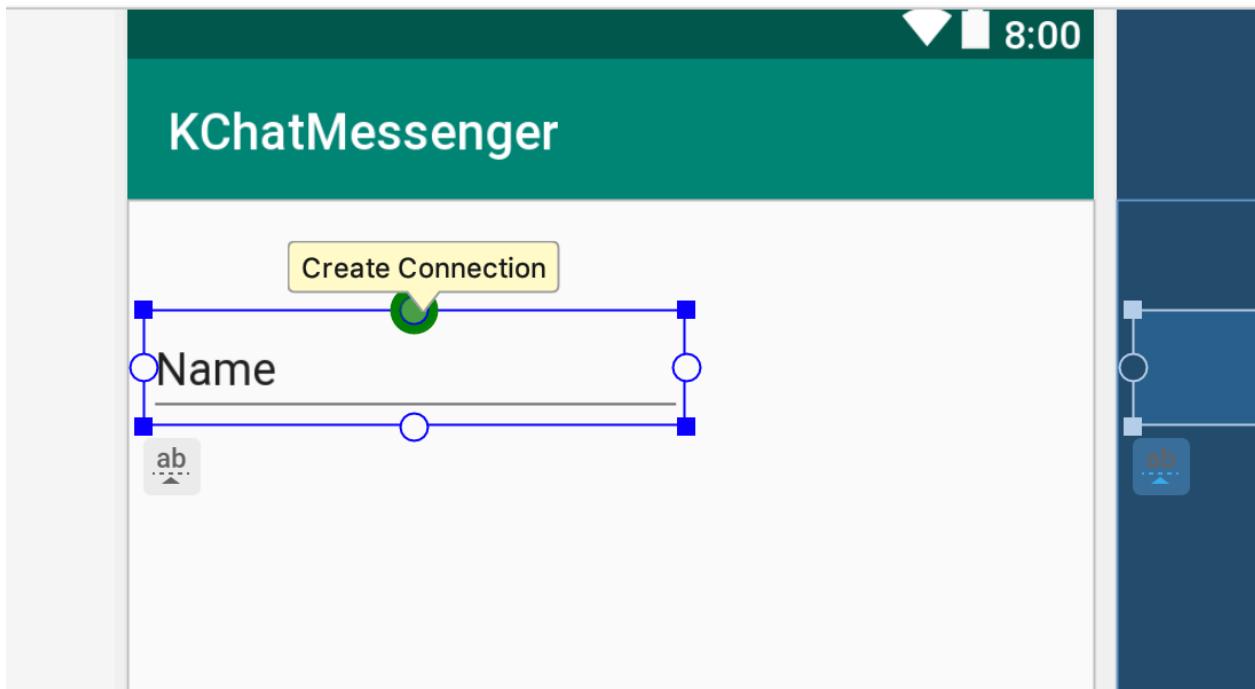


Figure 8: k011

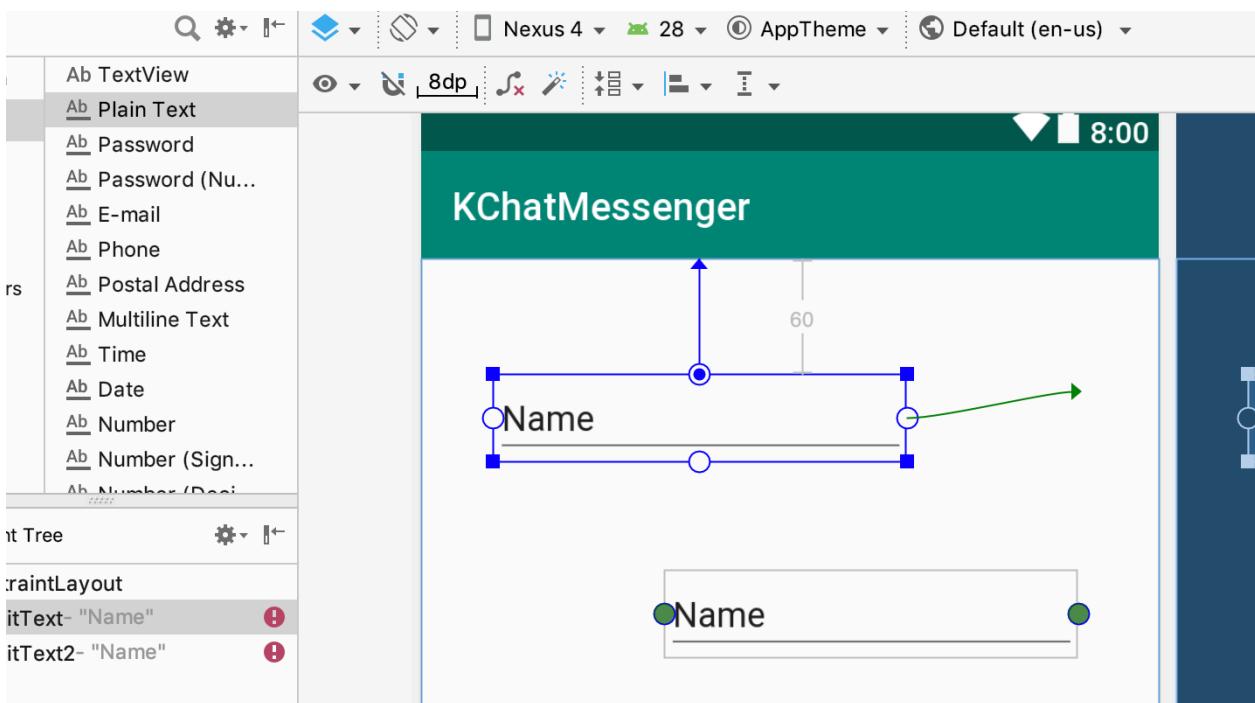


Figure 9: k012

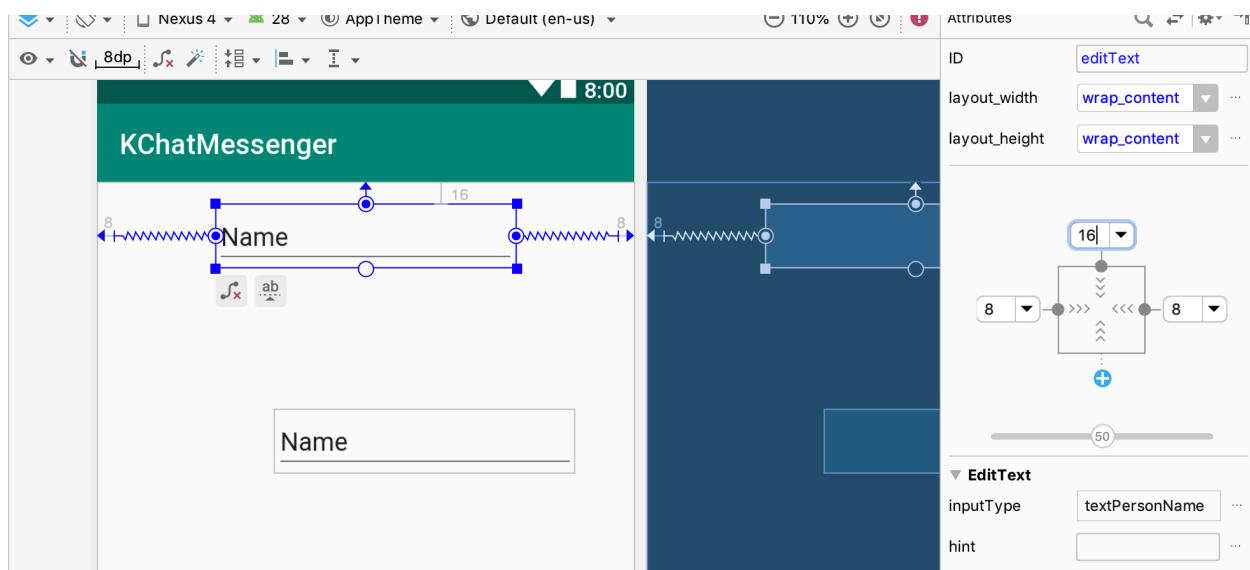


Figure 10: k013

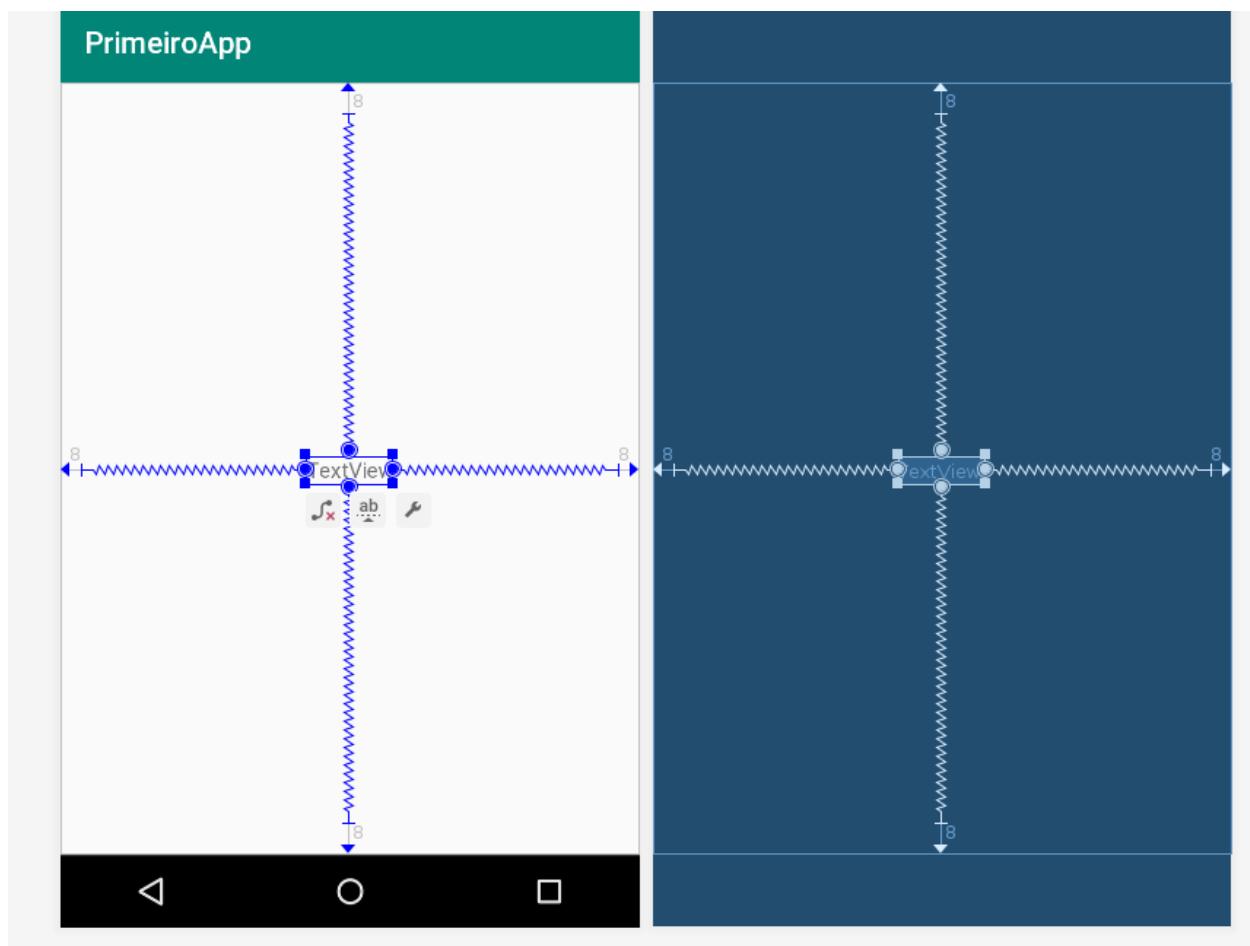


Figure 11: k013

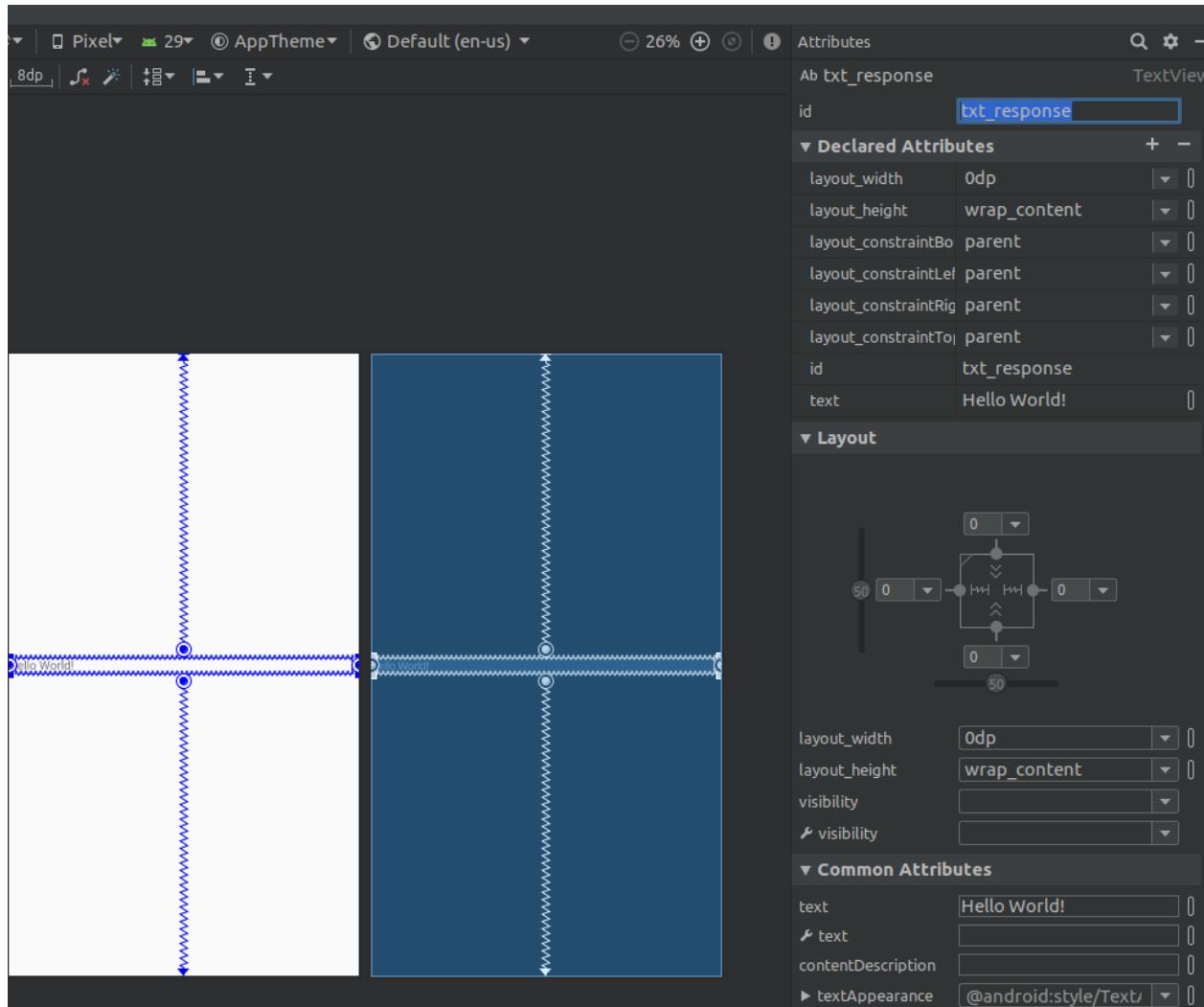


Figure 12: k014

Vamos analisar um pouco mais.

No canto superior a direita temos um painel com algumas informações:

1. ID: Indica uma referência única a este componente. Troque para **txt_response**.
2. Layout_width e layout_height: Esses campos possuem basicamente 2 opções.
 1. Wrap_content: Este valor indica que o elemento (neste caso a largura) deve respeitar o conteúdo do componente. Ou seja, ele encapsula a largura até o conteúdo aparecer. Geralmente usamos dessa forma quando temos um conteúdo dinâmico ou o layout do texto deverá respeitar a largura do texto em si..
 2. Match_constraint: Aqui é onde a constraint é efetivada. Altere o status e veja o que acontece. A largura irá se esticar até a extremidade da tela com margins onde será definida nesse “quadrado” abaixo.

Ao mudar o atributo da largura, perceba que as constraints foram ativadas e a partir de agora, você deve definir o quanto de margin ela deve ter em relação ao container pai (no caso, a tela).

Isso é feito pelo quadrante ao lado onde temos:

- Top (topo): 8
- Left (esquerda): 8
- Right (direita): 8
- Bottom (embaixo): 8

Clique em **Text** abaixo do editor Visual do Android Studio para visualizar a codificação do arquivo xml.

IMPORTANTE: Se você estiver usando o Androidx como mencionado no começo desse material, você terá as definições de pacotes mais recentes, por exemplo:

Sem Androidx: **android.support.constraint.ConstraintLayout**

Com Androidx: **androidx.constraintlayout.widget.ConstraintLayout**

Isso se repetirá em todo o material. Logo, aconselho a usar o Androidx para mantermos os pacotes atuais corretos com as novas versões do Android recente.

Seu arquivo deve estar semelhante a este abaixo:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:text="TextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:id="@+id/txt_response"
        android:layout_marginTop="8dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Novamente temos um arquivo XML. Perceba que o componente principal é o

`<androidx.constraintlayout.widget.ConstraintLayout`

onde é aberta a TAG no começo do arquivo e fechada logo abaixo da seguinte forma

`</androidx.constraintlayout.widget.ConstraintLayout>`.

Os arquivos XML trabalham com tags de abertura e fechamento sendo que, dentro delas, podemos ter outras tags que são componentes como Button ou TextView por exemplo.

Cada componente possuem muitos atributos que fazem o layout acontecer. Cores, margins, larguras, alinhamento de texto, fontes e etc.

Note que a tag `<TextView />` possue atributos como `android:text="TextView"` e `android:layout_marginTop="8dp"`. Para você praticar um pouco o XML, vamos alinhar o texto no centro da tela.

Escreva `android:textAlignment="center"` abaixo de qualquer atributo e note o que acontece com o editor.. ele centraliza o seu texto.

Não dá pra escrever todos os atributos aqui porque são muitos e alguns funcionam em tipos específicos de componentes. Mas no dia a dia você vai decorando.

Pois bem, tente agora por conta própria adicionar um Button com o ID **btn** e altere o texto do botão **Responder**. Abaixo vou colocar a “cola” para você comparar com o seu...

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:text="TextView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:id="@+id/txt_response"
        android:layout_marginTop="8dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        android:textAlignment="center"
        android:layout_marginRight="8dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent" />
```

```

<Button
    android:text="Responder"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/btn"
    android:layout_marginTop="8dp"
    app:layout_constraintTop_toBottomOf="@+id/txt_response"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Para deixar o layout do botão um pouco mais elegante vamos adicionar uma cor de fundo, uma cor do texto e um *padding* a este botão.

Os atributos são:

- android:background="@color/colorAccent"
- android:textColor="@android:color/white"
- android:padding="20dp"

Note que para usar cores precisamos de uma referência ao arquivo **colors.xml**. As referências em Android são feitas pela anotação **@color** ou **@string** ou **@style** e assim por diante. Dessa forma, o Android acessa esses arquivos e busca os valores definidos por lá.

Porém vemos que temos um **@android:color**, isso indica ao android para não buscar no nosso arquivo **colors.xml** e sim no **colors.xml** do SDK do Android, criado pelos Googlers.

Isso porque eu ainda não defini uma cor branca, então vou usar do SDK porque lá já tem isso pronto!

Como teste, tente alterar a cor de fundo do container.

Dica: o **androidx.constraintlayout.widget.ConstraintLayout** também é um componente customizável..

De volta ao editor visual, logo no menu de attributes, temos o seletor de *background*. Clique nele.

Essa alteração que fizemos poderá ser feita via interface visual.

E o que é esse tal de ID?

O ID usaremos futuramente para acessar esses componentes no nosso código Kotlin. É dessa forma que o Android pode trabalhar com o layout e definir lógicas de programação para eles como eventos de click no touch do celular e etc.

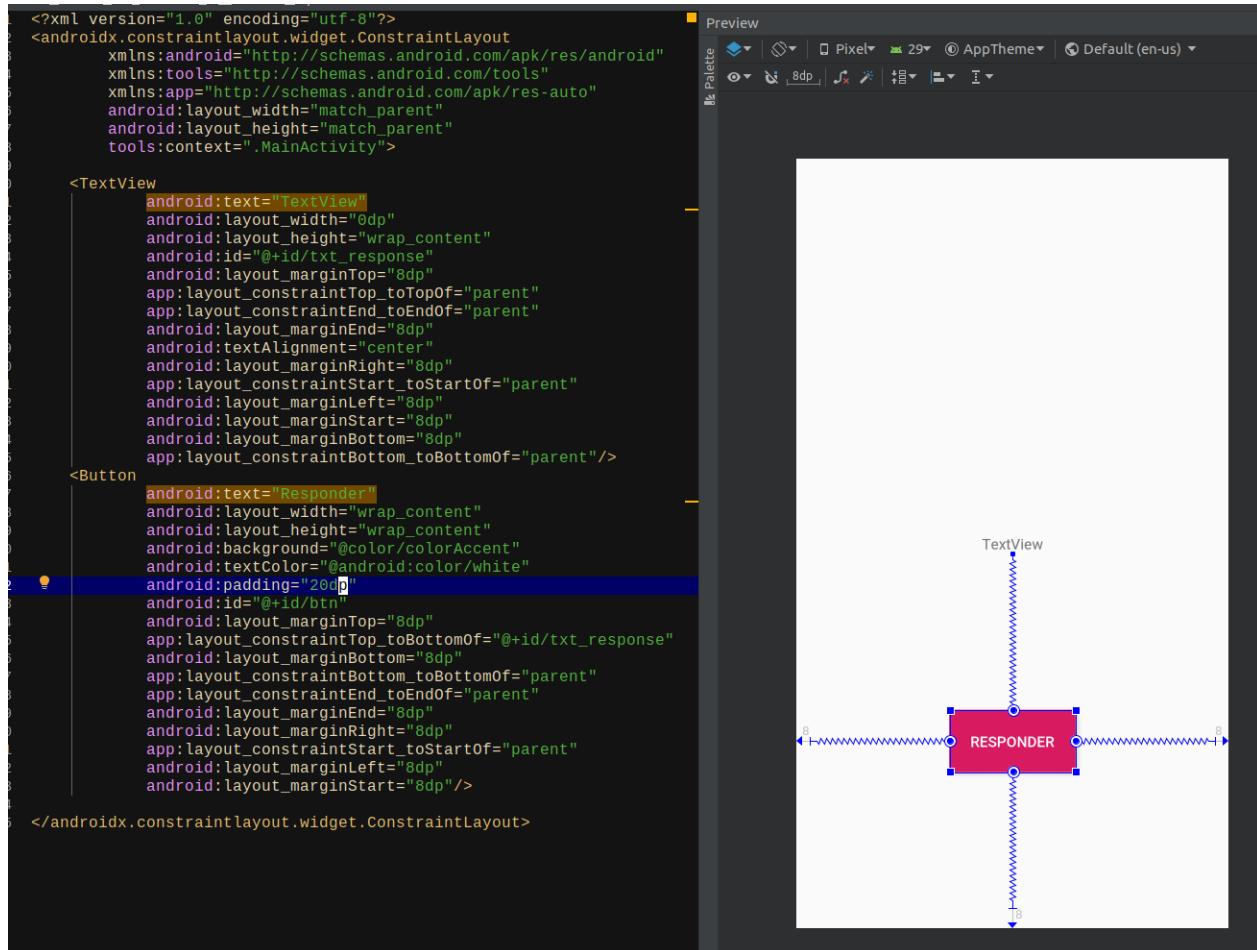


Figure 13: k014

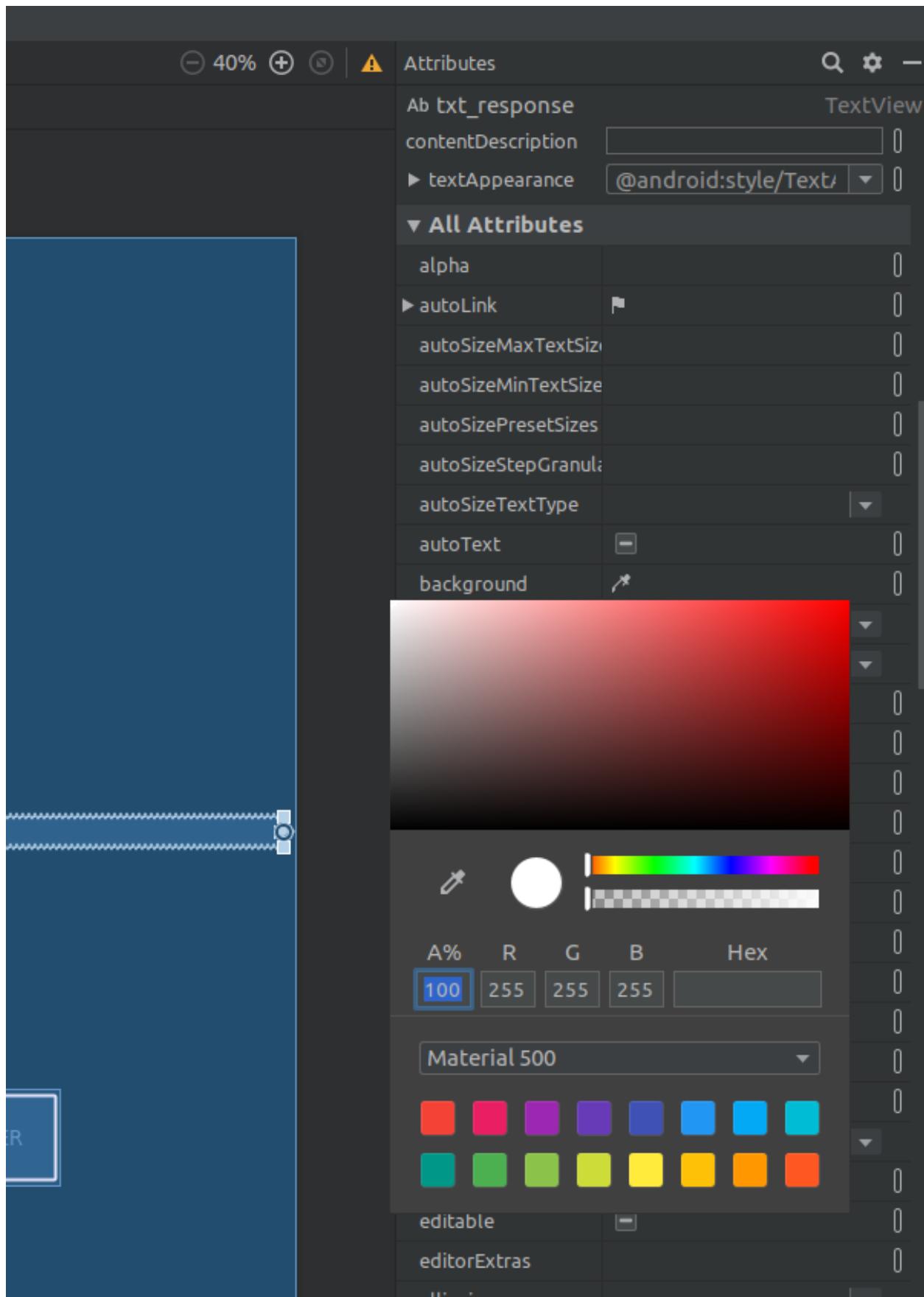


Figure 14: k014

Essa “conexão” entre Layout e Código Lógico (Kotlin/Java) se dá a um arquivo especial criado automaticamente (lembra da pasta **generatedJava** que não devemos mexer?).

Esse arquivo é o arquivo **R.java**. o R vem de Resources. Ou seja, esse é o arquivo automatico que literalmente cria referencias ao layouts e seus componentes para acessá-los no código Kotlin.

Agora vamos abrir nossa Activity para dar “vida” a esse aplicativo porque até agora somente criamos o layout que não funciona os eventos de clique ou os textos não são alterados.

7.1 Dando Vida a Activity

A Activity é onde vive a lógica do seu aplicativo, onde vive o código fonte que se conecta com o layout (usando os recursos do arquivo R).

Abra o Arquivo **MainActivity** e veja alguns pontos:

- **Package:** O diretório onde se encontra esse arquivo. Isso evita conflitos de códigos, geralmente o dominio invertido para ficar único, usado para organizar os arquivos em diretórios. Sempre que você criar um arquivo, ele deverá estar em um pacote para que quando outros arquivos precisarem utilizá-lo será fácil encontrá-lo.
- **Import:** usado para referenciar outras classes e arquivos para que possamos usar seus recursos no código atual aberto. Ou seja, após definir um package, podemos importá-lo em outros arquivos.
- **class MainActivity:** Indica a classe atual deste arquivo, um molde que representa a lógica da nossa activity. No próximo capítulo vamos entrar nesse conceito de classe e objeto. Por hora, entenda como sendo isso que define o que é uma Activity
- **: AppCompatActivity():** Indica que a MainActivity herda as funcionalidades da classe AppCompatActivity (que é uma classe dentro do SDK Android) com outros comportamentos e dados para serem acessados.
- **override fun onCreate():** é o ponto de entrada - entrypoint - de toda activity, a execução do código começa por aqui. Esta é um função que executa algum comportamento do objeto atual, neste caso, executa o comportamento de Criar algo do objeto MainActivity.
- **parâmetro - o parenteses depois de onCreate:** funções são geralmente verbos, como **tocar** guitarra, **andar**, **fazer** login, etc. Entao se tivermos uma função **tocar()** podemos dizer o que ele deve tocar... como parâmetros: **tocar(guitar)**.. mais a frente vamos entrar em detalhes o que é objeto, parâmetro e nome de parâmetro.
- **super :** usado para chamar outra função na classe pai. Falaremos mais sobre isso quando for aprender mais sobre a linguagem orientada a objetos.
- **setContentView(R.layout.act_main):** diz qual é o layout, a view que contém o conteúdo. Neste caso, o parâmetro dessa função **setContentView()** é a referencia pelo arquivo **R** do Layout **act_main.xml**. Dessa forma, quando precisarmos acessar o TextView ou o Button pelo seu ID, é desse arquivo que ele vai buscar.

As activities possuem ciclos de chamadas dependendo do que acontece, como quando você minimiza o app ou quando você fecha por completo o app. São métodos do ciclo de vida de uma activity.

Só para ilustrar, veja essa imagem que mostra o ciclo de vida de uma Activity:

Sempre use `onCreate` e `setContentView` antes de qualquer código para que você tenha a referencia dos componentes da tela no código e consiga aplicar as suas lógicas de programação.

Escreva o seguinte código:

```
package co.tiagoaguiar.primeiroapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.act_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_main)

        txt_response.text = "Oi TextView"
    }
}
```

Neste código acima, estamos importando um código para buscar a referencia do `TextView` `import kotlinx.android.synthetic.main.act_main.*` e estamos alterando o valor do texto para `Oi TextView`. Execute o aplicativo e veja o resultado.

Uma outra forma (mais antiga) de fazer isso era com o seguinte código:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_main)

        val textView = findViewById<TextView>(R.id.txt_response)
        textView.text = "Oi TextView"
    }
}
```

Aqui buscamos o `textView` pelo seu ID e ainda indicamos o **tipo de objeto** que irá retornar, no caso `TextView` `findViewById<TextView>(R.id.txt_response)`, guardamos ele em uma **variável** chamada `textView` e depois mudamos seu texto. Aqui escrevemos mais código mas temos o mesmo resultado.

Quando aprendermos programação orientada a objetos, vamos entender esse conceito melhor. Por hora, brinque com o `textView`, veja o que você pode alterar de texto e o mais importante,

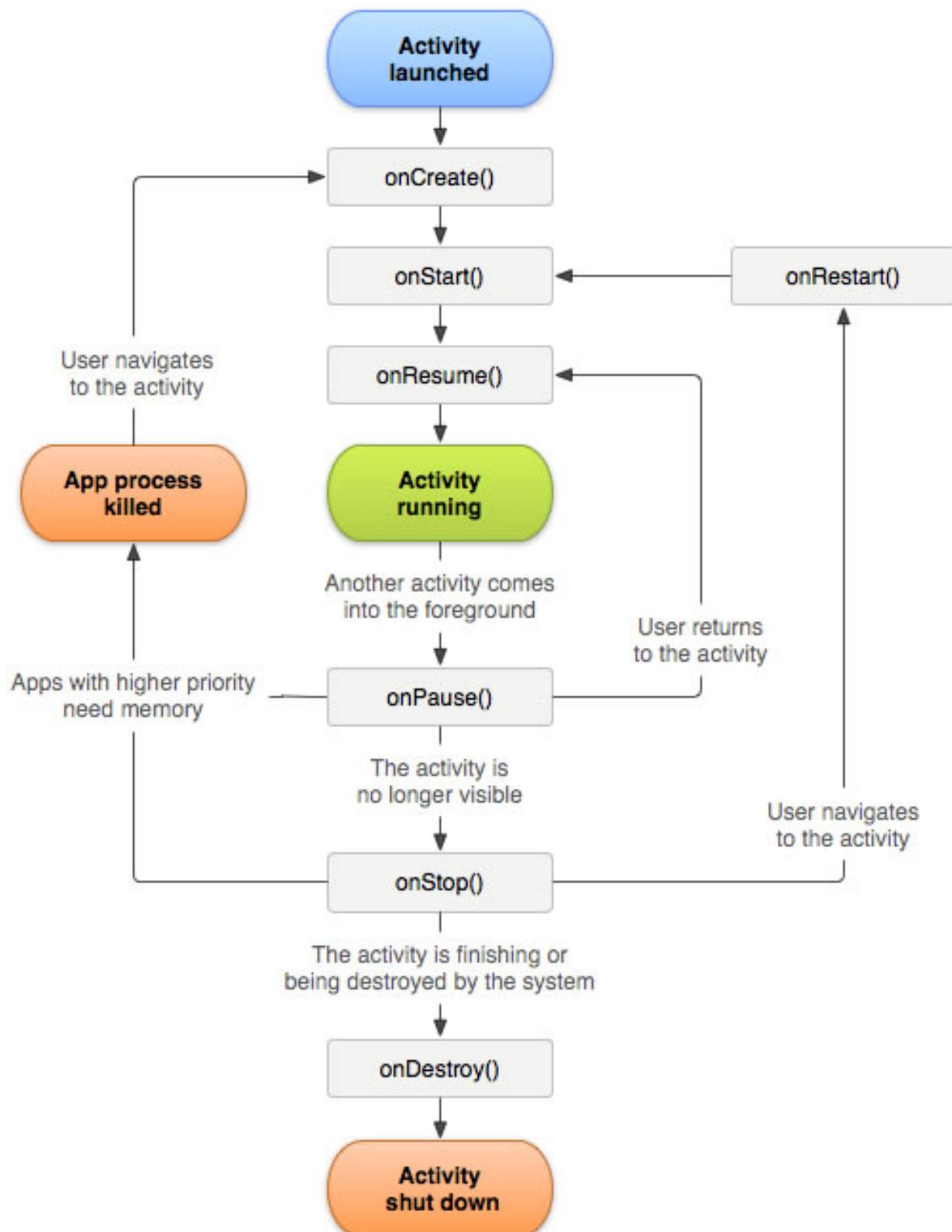


Figure 15: k018

comece a aprender a LER códigos. Você passará boa parte identificando e analisando a lógica do aplicativo.

Como exercício, tente importar o botão que criamos e alterar o seu texto para **Enviar**.

7.2 Dando Vida ao Botão

Para que os componentes do Android “escutem” os eventos que acontecem na tela como um touch (tap) ou a rolagem com os dedos na tela, precisamos declarar um bloco de código que chamamos de **Listeners**.

Escreva o seguinte código na sua Activity, clique em executar e aperte o botão na tela:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_main)

    btn.setOnClickListener {
        txt_response.text = "Botão Clicado"
    }
}
```

A mensagem que você digitou (**Botão Clicado**) é atribuída no TextView apenas quando o bloco interno do `setOnClickListener` é executado. E, essa execução acontece no evento de touch na tela.

Em outras palavras, existe um “ouvinte” ou listener que escuta e espera esse evento acontecer para executar o bloco de código dentro de `{ }`

Talvez você esteja se perguntando como isso realmente funciona por “debaixo do capô”. No próximo capítulo daremos inicio a explicação do que é orientação a objetos, o que significa esses blocos listeners e as classes.

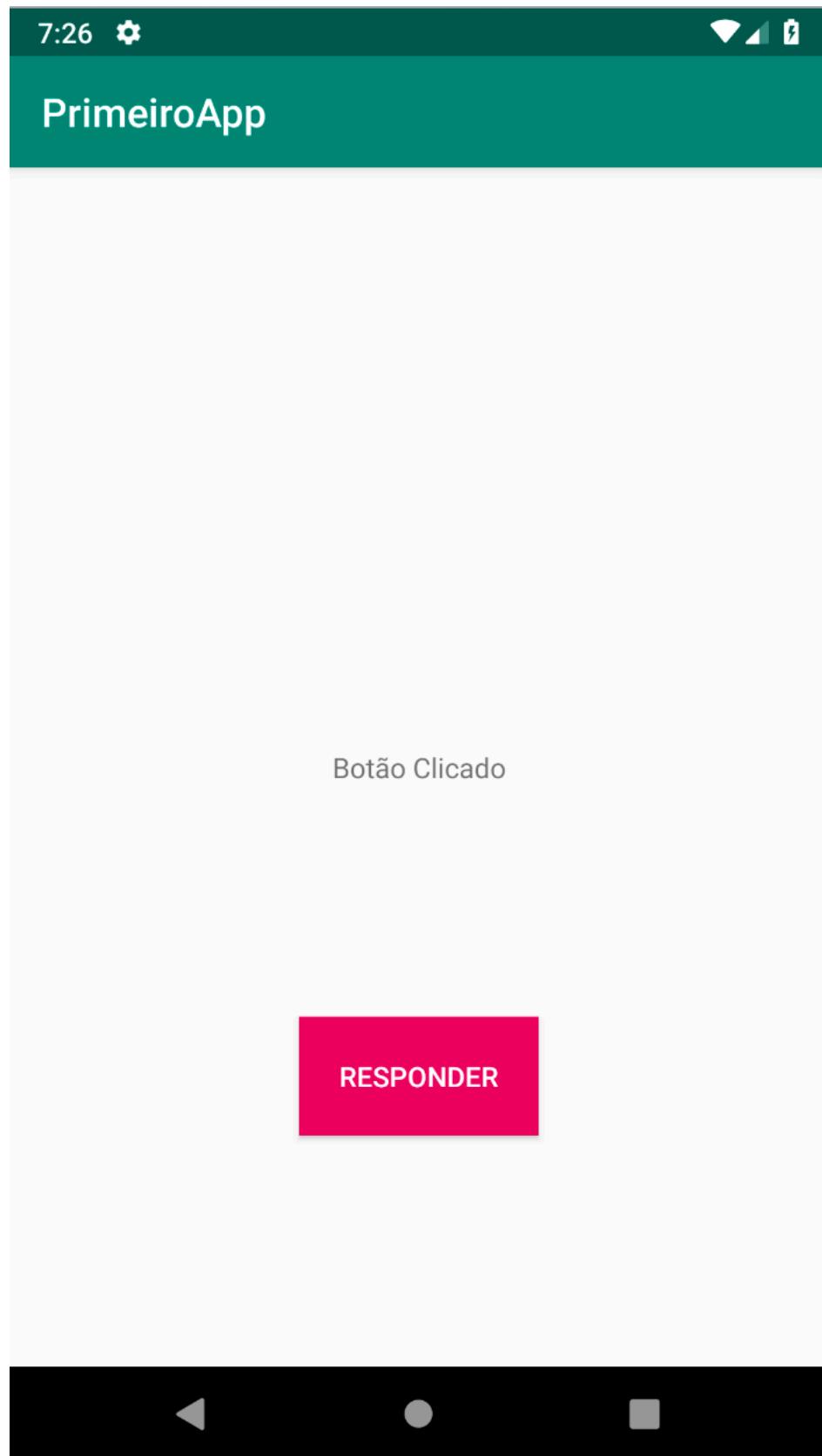


Figure 16: k018

8 Capítulo 4: Programação Orientada a Objetos em Kotlin

Para entendermos como as coisas funcionam no mundo da computação/programação, primeiro precisamos voltar um pouco no passado e entender como os programas eram construídos.

Programar nada mais é do que “dizer” ao computador através de instruções o que ele deve fazer. Exemplo: **some 2 números e me devolva o resultado** ou **faça login do usuário e redirecione ele para a tela de contatos** e assim por diante.

Tudo que fazemos é escrever instruções que são convertidas em uma linguagem de máquina para ser interpretada por esse computador (ou smartphone que nada mais é que um computador também).

Antigamente, essas instruções eram escritas em arquivos de forma linear. Ou seja, sequencial.

Os programas tinham um começo - ponto de entrada - um processamento no meio e uma resposta ao final dependendo do que acontecia no processamento e dos inputs de entrada.

Porém, conforme os programas começaram a crescer, ficou praticamente impossível manter a qualidade e a manutenção desses programas de forma linear. Isso porque algumas instruções dependiam de outras e acabava ficando confuso gerenciar todo esse ecossistema de forma sequencial.

Então, eis que surge a programação orientada a objetos para resolver esses problemas e para tornar a vida do programador mais fácil.

Ainda escrevemos instruções para o computador executar, porém desta vez, com outro paradigma e de forma quase não sequencial.

8.1 Primeiro Conceito: Objetos

A principal ideia desse conceito é tornar tudo que compõe o software, todos os problemas do mundo real em objetos no mundo virtual. Vou te dar um exemplo: pessoas, carros, usuários, botões, cores, campos de textos, cartão de crédito, telas, absolutamente tudo o que você precisa para criar um software pode ser tratado como um objeto.

No nosso aplicativo que estamos criando, você pode identificar alguns objetos como:

- Botão (Button)
- Campo de texto (TextView)
- Container - tela do celular (View)
- Gerenciador da lógica e da tela (Activity)
- e muito mais..

Conforme nosso projeto for crescendo, inevitavelmente teremos mais objetos para se comunicar e trabalharem em conjunto e o melhor, cada um com a sua **responsabilidade**.



Figure 17: k018

Desta forma, o problema do mundo real que precisamos resolver (criar um aplicativo de respostas) pode ser solucionado de forma clara e não diretamente sequencial.

É muito mais simples dar manutenção e identificar quem é responsável pelo o que e onde devemos alterar e corrigir eventuais erros no sistema.

Bom, essa a primeira parte - **todas as “coisas” que vivem no software orientado a objetos devem ser tratados como OBJETOS**.

Pegou esse primeiro conceito!? Se sim, ótimo! Se não leia novamente ;)

8.2 Segundo Conceito: Classes

Ok, já sei que devemos tratar tudo como objetos, mas como eu crio objetos!?

É o seguinte, para um objeto existir (ou tomar vida no mundo computacional) nós precisamos de uma espécie de **Molde** ou **Forma** para que ele exista.

É literalmente um molde para você criar esses objetos, semelhante a assar pão de queijo ou bolo de arroz!

Todo pão de queijo (objeto) precisa de uma forma/molde para indicar o tamanho, formato redondo e outras características únicas do pão de queijo. Esse “molde” chamamos de **classe** na programação.

Todo objeto pode ter 2 coisas principais:

1. Atributos / Propriedades
2. Métodos / Funções

Atributos ou propriedades como gostam de chamar são características que podem definir um objeto do outro. Vamos um exemplo:

O molde de um objeto Botão pode definir que cada botão criado deverá ter:

- Cor de fundo
- Tamanho
- Texto
- Cor do Texto

Ou seja, nossa **Class** que define a estrutura de um objeto **Button** terá essas propriedades.

Se formos escrever esse molde que irá gerar 1 ou mais botões, podemos escrever assim:

```
class Button {

    var bgColor = 0xFF
    var size = 16
    var text = "Texto do botao"
    var textColor = 0x00

}
```

Então o código acima significa exatamente isso:

Declarei uma classe (molde) do tipo Button onde esse Botão deverá ter 4 propriedades.

Dessa forma, cada objeto poderá ser diferente - um com cor branca e texto preto, outro com tamanho 18 e texto diferente e assim por diante - mas todos irão respeitar as propriedades definidas no molde (ou na classe).

Em resumo, **Classes** são moldes para definir a estrutura de um objeto. E cada objeto terá uma vida com propriedades diferentes - ou iguais - mas nunca serão o mesmo objeto.

O segundo item que uma classe pode definir além de propriedades são métodos ou funções.

Métodos definem **ações** que um objeto pode executar. Essas ações são geralmente verbos como: andar, enviar, tocar, redimensionar, etc.

Vamos a um exemplo: Um objeto do tipo **Guitar** pode ter propriedades como tamanho, número de cordas, cor, etc. E pode ter ações (métodos) como emitir som, afinar, etc.

No nosso exemplo do botão, além das propriedades poderíamos ter as funções:

- enviar contato
- alterar cor de fundo

- escutar eventos de click
- etc

Vamos criar um outro objeto para você entender melhor o que significa função, combinado!?

Imagine uma classe **Calculadora**. A classe calculadora é um molde para objetos do tipo Calculadora XD.

A calculadora tem a responsabilidade de calcular números. Logo, vamos escrever o seguinte:

*Definir uma classe calculadora com 1 propriedade que é resultado e 1 função que é somar.

```
class Calculator {

    var result = 0

    fun sum(x: Int, y: Int): Int {
        result = x + y
        return result
    }

}
```

Primeiro declaramos o tipo do Objeto, no nosso caso - **class Calculator**.

Nota: Aconselho sempre escrever em inglês para manter um padrão e ter mais qualidade de código. Assim, qualquer pessoa no mundo entenderá o que seu sistema faz.

Depois, declaramos uma variável - propriedade - inicializada com zero e com o nome de **result**.

Depois, declaramos uma função que **recebe 2 parâmetros**, x e y. Sendo que os 2 parâmetros são do tipo **Int**.

Perceba que Objetos começam com a primeira letra maiúscula. Logo, os 2 parâmetros dessa função, são 2 objetos do tipo Int que, eventualmente também tenha uma definição de comportamento e propriedades lá no SDK. Isso indica que alguém escreveu **class Int** e definiu como ele deve funcionar. Eu disse que tudo é objeto ;)

E depois declaramos que essa função deverá retornar um valor do tipo **Int**.

Dentro do bloco **** { } **** executamos a lógica (que é somar x + y) e atribuimos o resultado em **result** terminando no retorno - **return** - do tipo Int.

Escreva o código a seguir e execute o aplicativo:

```

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_main)

        btn.setOnClickListener {
            val calculator = Calculator()
            val resultOfCalculator = calculator.sum(2, 3)
            txt_response.text = "resultado é ${resultOfCalculator}"
        }
    }
}

class Calculator {

    var result = 0

    fun sum(x: Int, y: Int): Int {
        result = x + y
        return result
    }
}

```

Veja que o resultado no TextView aparece **5**.

Aqui damos vida a um objeto `val calculator = Calculator()`. O ato de criar um objeto, na programação chamamos de **instanciar um objeto**.

Com a instância em mãos, chamamos a função `sum()` passando 2 e 3. E, como essa função tem um retorno, guardamos o seu retorno em uma nova variável chamada `resultOfCalculator: val resultOfCalculator = calculator.sum(2, 3)`

Conclusão, funções são **ações** que o seu objeto pode executar para realizar lógicas de programação. Algumas funções **podem** retornar algum valor, outras apenas alteram o estado das propriedades do seu objeto - como no caso de uma função `alterarCor...` em tese, ela iria apenas alterar a propriedade `color` e não retornar nada.

Outro ponto é que funções podem receber parâmetros para serem manipulados dentro da função - como no nosso caso a variável X e Y - que foram utilizadas para para somar.

Altere os parâmetros e veja que sempre é retornado uma soma. Querendo ou não, ainda mandamos instruções para o computador, só que dessa forma, cada um como objeto e com uma responsabilidade diferente.

8.3 Tipos de dados

Para trabalhar com variáveis e propriedades devemos definir o tipo de dados dela.

Isso garante a consistência do projeto. Ou seja, se uma variável está declarada como **Int** que é um número inteiro, você só pode alterar o seu estado para outro número inteiro e não para um texto - como uma **String**.

No exemplo anterior, não podemos fazer isso aqui: `result = "novo valor"`. Porque o tipo da propriedade é **Int** **var result: Int**.

Existem 2 tipos de variáveis:

- Mutáveis (que podem sofrer alteração)
- Imutáveis (que não podem sofrer alteração)

Quando escrevemos **var** dizemos que mais a frente podemos alterar:

```
var result = 0
result = 10
```

Já **val** não podemos alterar:

```
val result = 0
result = 10 // <- ERROR
```

Isso garante a consistência da propriedade. Se ela não precisa ser alterada, declare-a como **val**.

Em alguns lugares você verá uma lista enorme com os tipos de dados ou estruturas de dados que você pode estar criando que já é fornecida pela linguagem de programação. Entretanto, prefiro que você já aprenda a usá-los quando precisar usá-los!

Ficar tentando decorar o que cada coisa faz não é um bom caminho a se seguir. Ao contrário disso, é melhor ir aprendendo enquanto já está aplicando os conceitos teóricos.

Dito isto, este não será um material técnico - um manual - da linguagem de programação. Meu objeto é te dar o caminho das pedras para criar aplicativos já na prática.

Então, vamos voltar ao nosso código inicial e entender o que cada linha faz (de novo) só que olhando na perspectiva de orientação a objetos pois agora você já sabe o que é classe, objeto, função e propriedade.

8.4 De Volta a MainActivity

De volta na **MainActivity**, vamos entender melhor como ela funciona. Vamos enumerar as linhas para exemplificar.

```

package co.tiagoaguiar.primeiroapp // 1

import android.os.Bundle // 2
import androidx.appcompat.app.AppCompatActivity
import kotlinx.android.synthetic.main.act_main.* // 3

class MainActivity : AppCompatActivity() { // 4

    override fun onCreate(savedInstanceState: Bundle?) { // 5
        super.onCreate(savedInstanceState) // 6
        setContentView(R.layout.act_main) // 7

        btn.setOnClickListener { // 8
            txt_response.text = "Olá" // 9
        }
    }
}

```

1. **package**: indica o local onde se encontra o arquivo da classe corrente que pode ser importada futuramente em outros arquivos
2. **import**: importa classes de terceiros como classes do SDK Android
3. **import ***: importa ao invés de uma classe, uma série de propriedades de uma classe
4. **class MainActivity**: declara uma classe chamada MainActivity que será responsável por gerenciar a tela bem como eventos. Note que quem cria esse objeto do tipo MainActivity é o próprio Android SDK. Já a linha `: AppCompatActivity()` indica que a nossa classe MainActivity herdará as propriedades e funções da classe pai - AppCompatActivity. Somente propriedades como **private** não poderemos acessar.
5. **override fun onCreate**: como nossa classe é filha de AppCompatActivity podemos **sobrescrever** a função dela e definir o que quiser aqui; Já **fun** indica que é uma função e **onCreate** é o nome da função; `(savedInstanceState: Bundle?)` indica que essa função espera um parâmetro do tipo Bundle chamado de savedInstanceState; o `?` no Bundle, indica que quem chamar essa função poderá passar um objeto do tipo Bundle OU não passar nada como **null**. De novo, quem chama onCreate é o Android SDK que esta é a primeira função a ser executada. Veja o ciclo de vida da activity na imagem anterior.
6. **super.onCreate**: indica que mesmo depois de sobrescrever a função da classe pai, vamos chamar o que foi declarado na classe pai e, só depois, executar a linha 7 em diante.
7. **setContentView(R.layout.act_main)**: chama a função da super class (AppCompatActivity) setContentView onde passamos a referencia do layout que a Activity deverá gerenciar. Ou seja, passamos a propriedade **act_main** da classe Layout que está na classe R.
8. **btn.setOnClickListener**: executa a função do objeto Button que atribui um objeto “ouvinte” - nesse caso, o objeto é o bloco de código `{ }` que foi resumido. Assim, quando clicarmos no botão, dentro da classe Button, há uma chamada para esse “ouvinte”;
9. **txt_response.text**: atribui a propriedade de texto do objeto TextView chamado de txt_reponse o valor **Olá** que é uma String (conjunto de caracteres)

Dica: Abra os arquivos dos códigos no pacote do android e comece a lê-los para entender o conceito de orientação a objetos. Você precisará de muita interação com esses elementos para dominar o paradigma.

Vamos deixar o aplicativo mais dinâmico, já respondendo nossas eventuais perguntas.

Debaixo do `setContentView(R.layout.act_main)` declare uma variável imutável `val` do tipo array que pode armazenar uma sequência de valores e que são acessadas por índices.

```
val lists = arrayOf("Sim", "Talvez", "Não conte com isso!")
```

Para acessar cada frase, precisamos buscar pelo índice desta forma `lists[0]`, `lists[1]` ou `lists[3]`. Na programação, índices começam sempre em zero.

Agora no bloco listeners que é executado quando há eventos de touch, vamos criar um objeto da classe **Random** que possui a responsabilidade e função de randomizar número até certo índice.

```
val random = Random()
val index = random.nextInt(3)
```

Agora, a variável `index` sempre terá um valor de 0 a 2 (totalizando 3) sempre que apertarmos o botão.

Por fim, vamos buscar a frase com o índice dinâmico para que ela fique aleatória.

```
btn.setOnClickListener {
    val random = Random()
    val index = random.nextInt(3)
    txt_response.text = lists[index]
}
```

O código completo deverá ficar assim:

```
package co.tiagoaguiar.primeiroapp

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import kotlinx.android.synthetic.main.act_main.*
import java.util.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_main)
```

```
    val lists = arrayOf("Sim", "Talvez", "Não conte com isso!")

    btn.setOnClickListener {
        val random = Random()
        val index = random.nextInt(3)
        txt_response.text = lists[index]
    }
}
```

Execute o programa e veja o que o aplicativo pode te “responder” dependendo da sua “pergunta” ;)



```

14     val lists = arrayOf("Sim", "Talvez", "Não conte com isso!")
15
16     btn.setOnClickListener { it: View!
17         val random = Random()
18         val index = random.nextInt( bound: 3 )
19         txt_response.text = lists[index]
20     }
21
22
23 }
```

Figure 18: 001

9 Capítulo 5: Bug e Debugging

Um dos recursos mais poderosos que uma IDE como Android Studio pode nos fornecer é a capacidade de depuração (debugging) de uma aplicação.

Geralmente queremos visualizar em tempo de execução o que está dentro de cada variável.

9.1 Breakpoints

Para realizar esse processo, precisamos clicar no canto esquerdo, ao lado da linha de código para adicionar um **breakpoint** - ponto de parada - do código.

Depois, basta clicar no ícone de inseto que fica ao lado do botão Play para iniciar o aplicativo em modo Debug.

Execute o aplicativo e clique no botão para ver o código parando na linha definida.

Aperte **F8** para seguir para o próximo linha:

Note que logo abaixo do editor temos o campo onde podemos abrir o objeto atual **this** e visualizar variáveis locais como o **index**.

Aperte **F9** para deixar o programa prosseguir até o fim.

Você também pode adicionar mais de 1 breakpoint e visualizar o ciclo de vida e a ordem que as funções são chamadas.

9.2 LogCat

Além disso, durante o desenvolvimento do aplicativo, também podemos exibir valores de variáveis no Console chamado de **LogCat**.

Adicione essa linha logo abaixo do `val index = random.nextInt(3)`.

```
Log.i("Teste", "O index é ${index}")
```

Agora, no menu abaixo você tem acesso ao LogCat. Clique nele, execute o aplicativo e aperte novamente o botão no app.

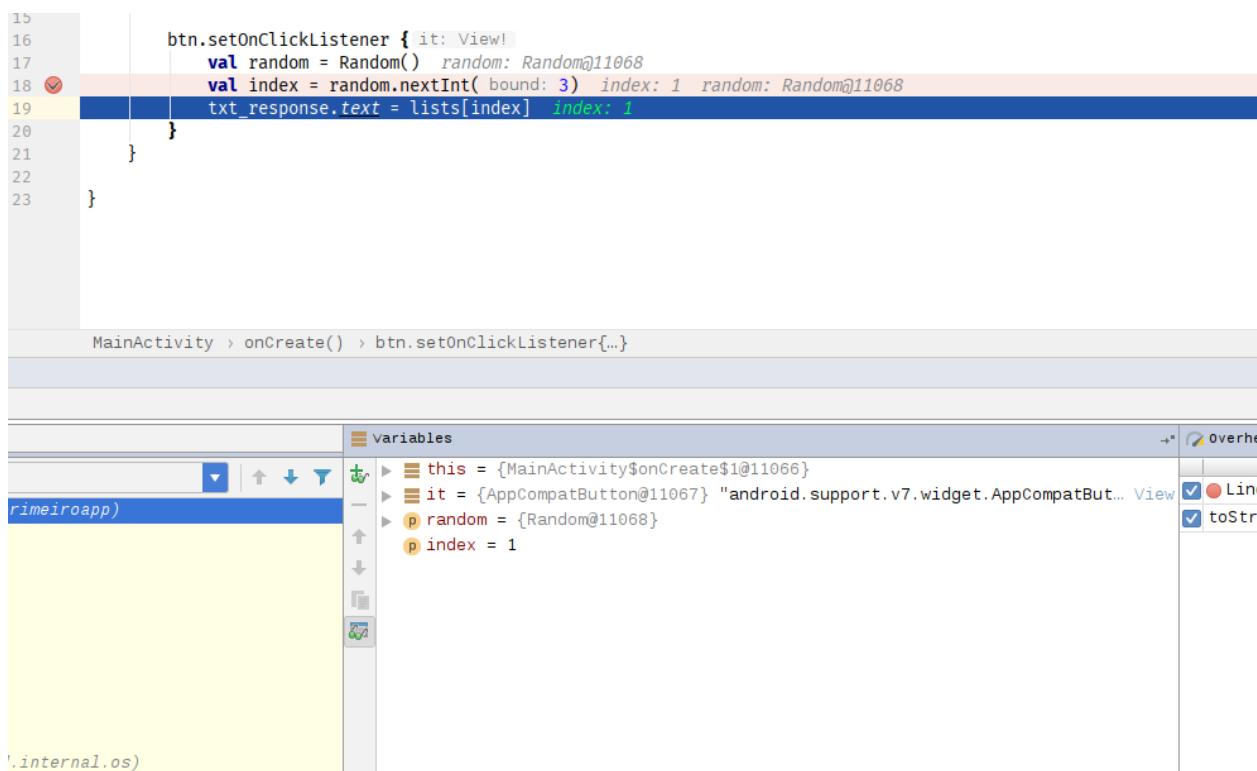


Figure 19: 001

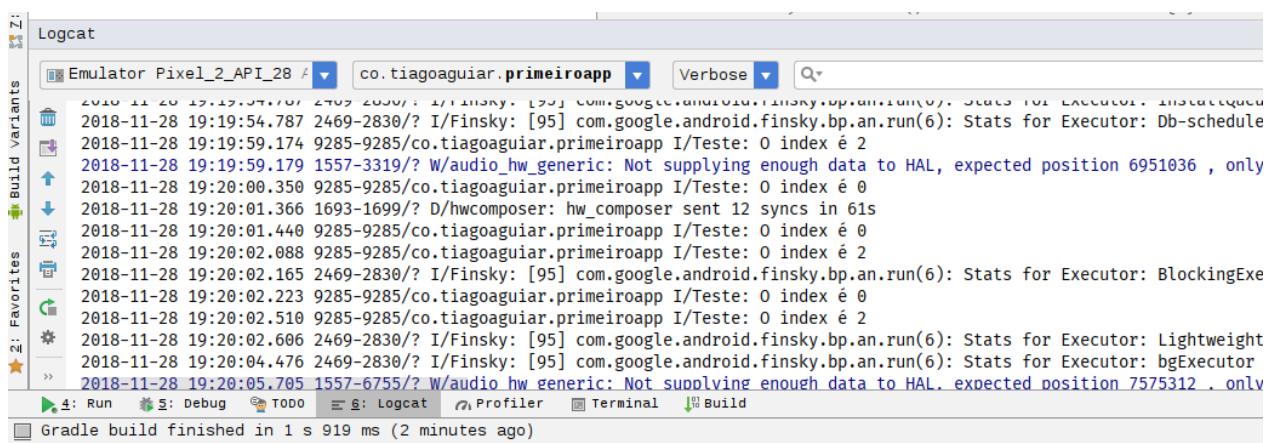


Figure 20: 001

Você verá a saída da variável no LogCat como: **I/Teste: O index é 2.**

10 Capítulo 6: UX Design

Vamos explorar um pouco mais dos recursos do Android e adicionar algumas animações para deixar o aplicativo um pouco mais elegante.

Além disso, vamos explorar recursos de menus e PopUps que são muito úteis em dezenas de milhares de aplicativos.

10.1 Animations

As animações - assim como layouts - são definidos em arquivos xmls na pasta `/res/anim/`.

Crie esse diretório anim dentro da pasta res. Depois, crie um arquivo de recurso de animação **Animation resource file** com o nome de `button_anim.xml`.

Esse arquivo deverá ser:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:fillAfter="true"
      android:interpolator="@android:anim/bounce_interpolator">
    <scale
        android:duration="600"
        android:fromXScale="1.5"
        android:fromYScale="1.5"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="1.0"
        android:toYScale="1.0" />
</set>
```

Esse arquivo define:

- Quanto tempo durará a animação
- A origem da animação: neste caso, iniciar 1.5x o tamanho do componente
- Tudo a partir do centro (pivot)
- Até voltar a estado normal

Agora no código Kotlin, vamos atribuir essa animação no nosso objeto Button. Primeiro vamos precisar da classe **AnimationUtils** e executar a função **loadAnimation** para buscar a referência do nosso arquivo xml.

Por fim, atribuir no botão pela fun:

```
btn.setOnClickListener {
    val animation = AnimationUtils.loadAnimation(this, R.anim.button_anim)
    btn.startAnimation(animation)
```

```

val random = Random()
val index = random.nextInt(3)
Log.i("Teste", "O index é ${index}")
txt_response.text = lists[index]
}

```

Execute o aplicativo, clique no botão e veja a animação por completo.

10.2 Menus

Para adicionar menus no Android também usamos arquivos XML. Crie uma pasta **/res/menu** e adicione o arquivo **menu.xml**.

Arraste o componente **Menu Item** do editor visual do Android Studio e atribua um ID para ele como **sobre**.

Altere o **title** para Sobre também.

De volta na Activity, vamos sobrescrever uma função da classe AppCompatActivity chamada **onCreateOptionsMenu**. Basta digitar esse texto e apertar Ctrl + Espaço para o Android Studio completar para você.

Ele deve gerar o seguinte código:

```

override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    return super.onCreateOptionsMenu(menu)
}

```

Neste código, vamos alterar para criar o menu xml.

```

override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu, menu)
    return true
}

```

Feita a alteração, agora aparecerá os três pontinhos no canto superior direito do aplicativo que servirá de menu.

Vamos adicionar um evento para escutar os cliques no menu.

Sobrescreva a função **onOptionsItemSelected**:

```

override fun onOptionsItemSelected(item: MenuItem?): Boolean {
    return super.onOptionsItemSelected(item)
}

```

Como nosso menu pode ter vários botões, teremos que distinguir qual será o botão clicado. Para fazer isso, vamos criar e testar uma condição chamada **when** para que **quando** algo for clicado, identificar pelo ID.

```
override fun onOptionsItemSelected(item: MenuItem?): Boolean {
    when (item?.itemId) {
        R.id.sobre -> {
            Log.i("Teste", "Sobre clicado")
        }
    }
    return super.onOptionsItemSelected(item)
}
```

Execute o aplicativo. Abra o LogCat e adicione o Filtro **Teste**. Agora clique no menu e veja a saída no console.

As linhas `when (item?.itemId) {` e `R.id.sobre -> {` é a condição para que quando o `item?.itemId` for igual ao ID do sobre, entre no bloco seguinte para imprimir a mensagem no Log.

Nesses capítulos você aprendeu:

- Mais sobre a linguagem Kotlin
- A construir Layouts
- A trabalhar com lógica de programação
- Criar animação
- Criar Menus
- Utilizar Debug
- E deixar o aplicativo dinâmico

Agora chegou o grande momento de criar um aplicativo mais completo, do começo ao fim.

Vamos aprender a criar um **Chat Messenger em Tempo Real** para você conversar com outras pessoas.

Iremos criar o aplicativo do absoluto zero até a publicação do Google Play.

Ao final desse material, você terá um aplicativo real em produção.

Então, sem mais delongas, **vamos codificar!**

11 Capítulo 7: Criando o Segundo Aplicativo Android - Chat Messenger

Neste capítulo iremos aprender a criar um novo aplicativo, só que desta vez, iremos criar algo mais tangível. Um aplicativo de Messenger - um Chat - onde poderemos criar contas, conversar com outros usuários e muito mais utilizando recursos em tempo real.

Vamos entrar em detalhes de como um aplicativo mais robusto é construído para que você possa terminar esse material já dominando o Android Studio, o SDK e outras tecnologias usadas por diversas empresas.

Para seguir esse mesmo tutorial em video com a [linguagem Java](#). Assista a playlist

Então sem mais conversas, vamos por a “mão na massa” e começar a codificar nosso novo aplicativo Android.

11.1 Configurando o Projeto Inicial

Se você já chegou até aqui muito provavelmente já tenha instalado o [Android Studio](#).

Então abra o seu Android Studio para começarmos a criar um novo projeto do zero.

Com o Android Studio aberto crie um novo projeto clicando em **Start a new Android Studio project** com o nome de KChatMessenger.

Antes de 2017, todos os aplicativos eram desenvolvidos usando a linguagem de programação **Java**. É interessante você também dominar Java porque existe muitos projetos em Java e empresas que buscam profissionais completos com esse conhecimento.

Outro ponto é que o próprio SDK Android está escrito em Java, então, para melhorar o seu entendimento, eu te recomendo aprender tanto Java quanto o Kotlin.

Kotlin é uma linguagem mais moderna, segura e que aumenta drasticamente a produtividade do desenvolvedor, mas por outro lado, omite muitas coisas que somente com o Java você entenderia.

Partindo do princípio de produtividade e modernidade, vamos começar a aprender a criar um aplicativo Android mais completo já usando a linguagem mais recente adotada pelo Google nesse eBook.

Para seguir esse mesmo app e dominar essas linguagens por completo, tem uma [playlist no Youtube](#) feito totalmente em Java. Assista lá.

Neste próximo passo sugiro selecionar a opção de API que atenda 100% dos dispositivos Android. Isso pode depender muito de quando você estiver lendo este tutorial. Enquanto escrevo, a última versão que está atendendo todos os dispositivos é a **API 15: Android 4.0.3 (IceCreamSandwich)**

Após escolher o mínimo SDK Android, vamos escolher uma atividade - Activity - vazia para começarmos do zero mesmo, sem templates prontos (ainda).

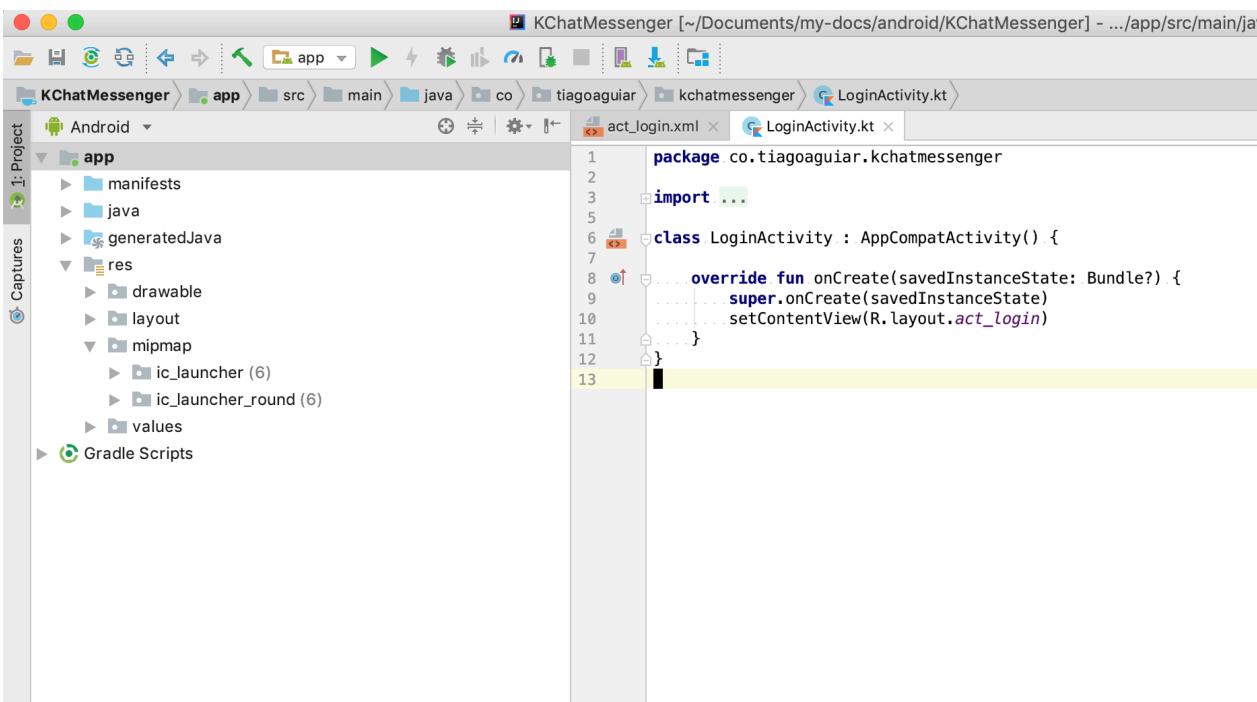


Figure 21: k002

Com o modelo de atividade em mãos (neste caso vazio) iremos definir tanto o nome da Activity quanto o nome do layout. Eu, particularmente, gosto de definir o nome da activity que indique o principal objetivo daquele conteúdo - neste caso, efetuar login.

E para o layout, geralmente defino **act_** para activities.

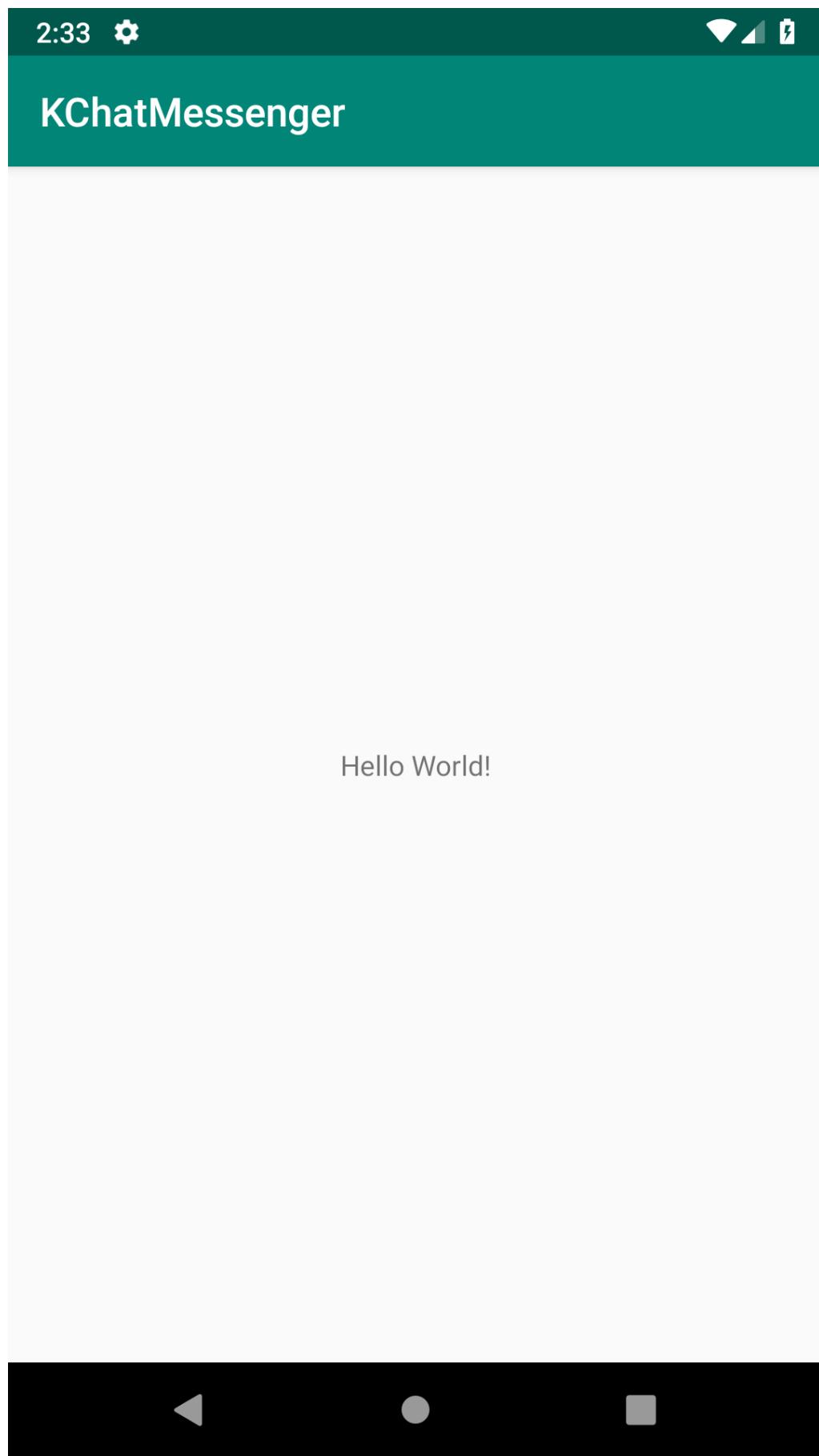
Se você chegou até aqui provavelmente já tem um projeto configurado e pronto para começar a brincar com o Kotlin e o Chat.

Por último, verifique se você já criou um emulador Android pelo **AVD Manager**, então clique no botão **Play** e execute o seu aplicativo para garantir que tudo foi construído como o esperado.

Antes de escrever qualquer linha de código, vamos primeiro modelar a UI - Interface do usuário - com os componentes do Kit de Desenvolvimento Android (SDK). Componentes como Editores de texto e botões. Dessa forma, já garantimos como o usuário final irá visualizar nosso aplicativo Android.

Além disso, vamos aprender como usar os recursos de “regras” de componentes para mantê-los alinhados e posicionados na tela de forma organizada. Esse recurso é conhecido como **constraints**.

E por fim, vamos melhorando aos poucos o Design do aplicativo Android para deixá-lo mais elegante e agradável, por mais simples que o aplicativo de Chat Messenger possa parecer ;)



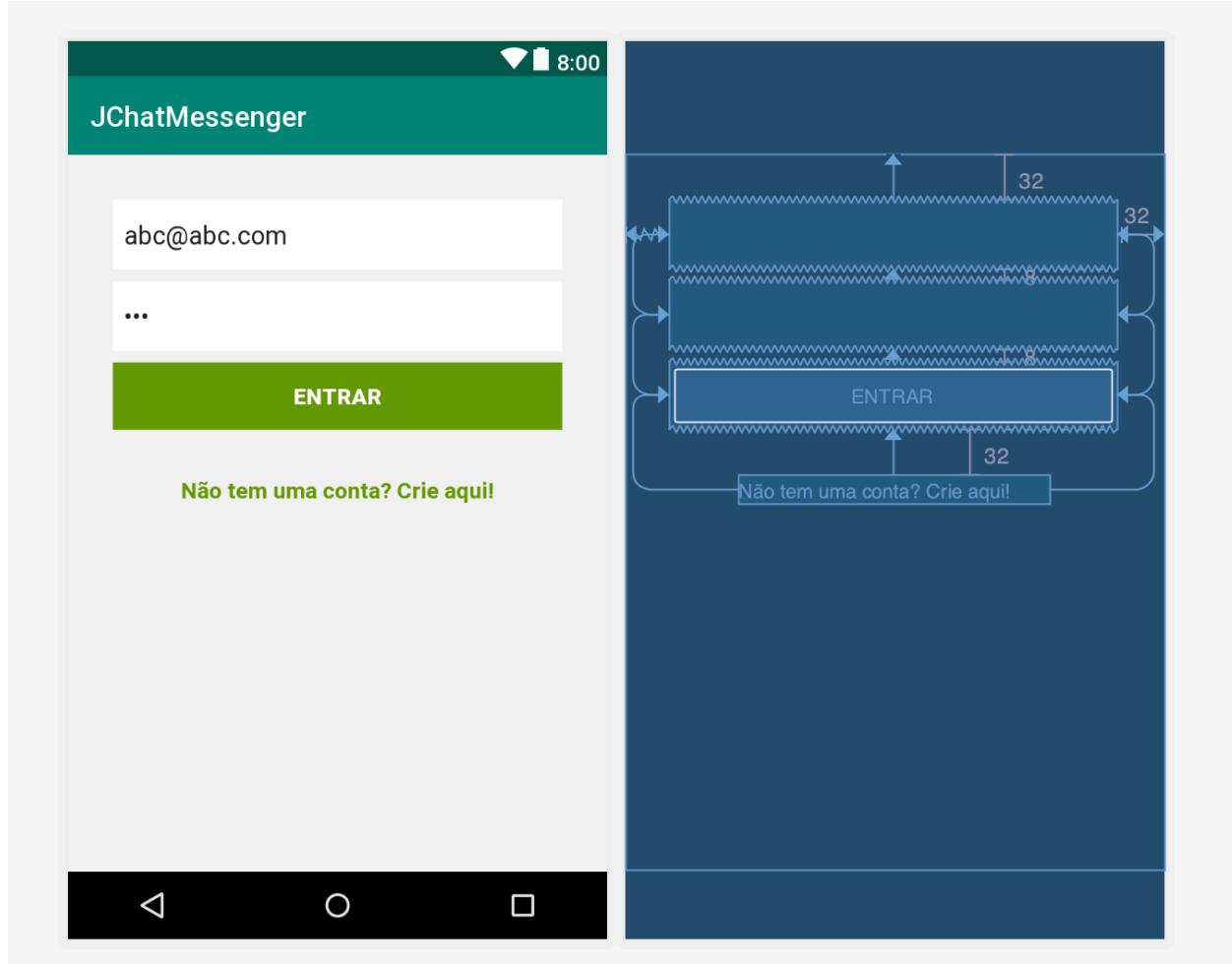


Figure 23: 008

11.2 Interface do Usuário: Primeiros Passos Para Criar Telas e Activities

Vamos iniciar a construção da tela para efetuar Login e Cadastro de um novo usuário no Chat. Quando precisamos adicionar uma nova funcionalidade em um aplicativo ou quando vamos começar do zero, precisamos antes de escrever quaisquer linha de código Kotlin, “desenhar” nossa tela e definir os componentes que vamos utilizar.

Isso facilita na construção da funcionalidade e também nos ajuda definir o que realmente é importante. O que realmente precisamos exibir para o usuário final do aplicativo.

Tela de login do chat

Tela de cadastro de usuário do chat

Pois bem, vamos primeiramente abrir o nosso arquivo de layout principal criado pelo Android Studio no passo anterior - o **act_login.xml**.

Neste ponto, você já deve entender como funciona a estrutura de pastas de um projeto Android. Todos os recursos como mídias, layouts, cores e, eventualmente, textos fixos que aparecem no aplicativo ficam no diretório de recursos conhecido como **res**.

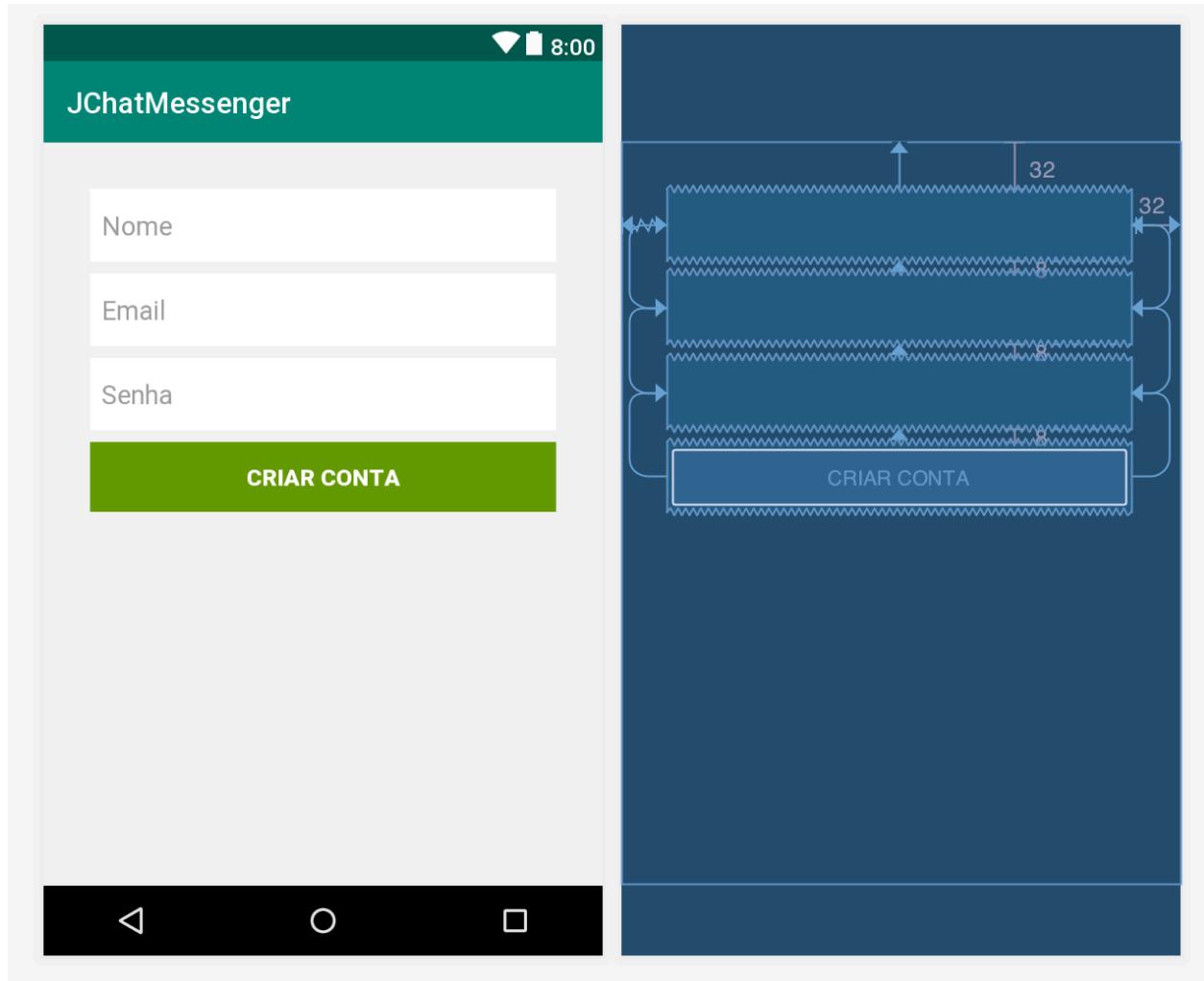


Figure 24: 009

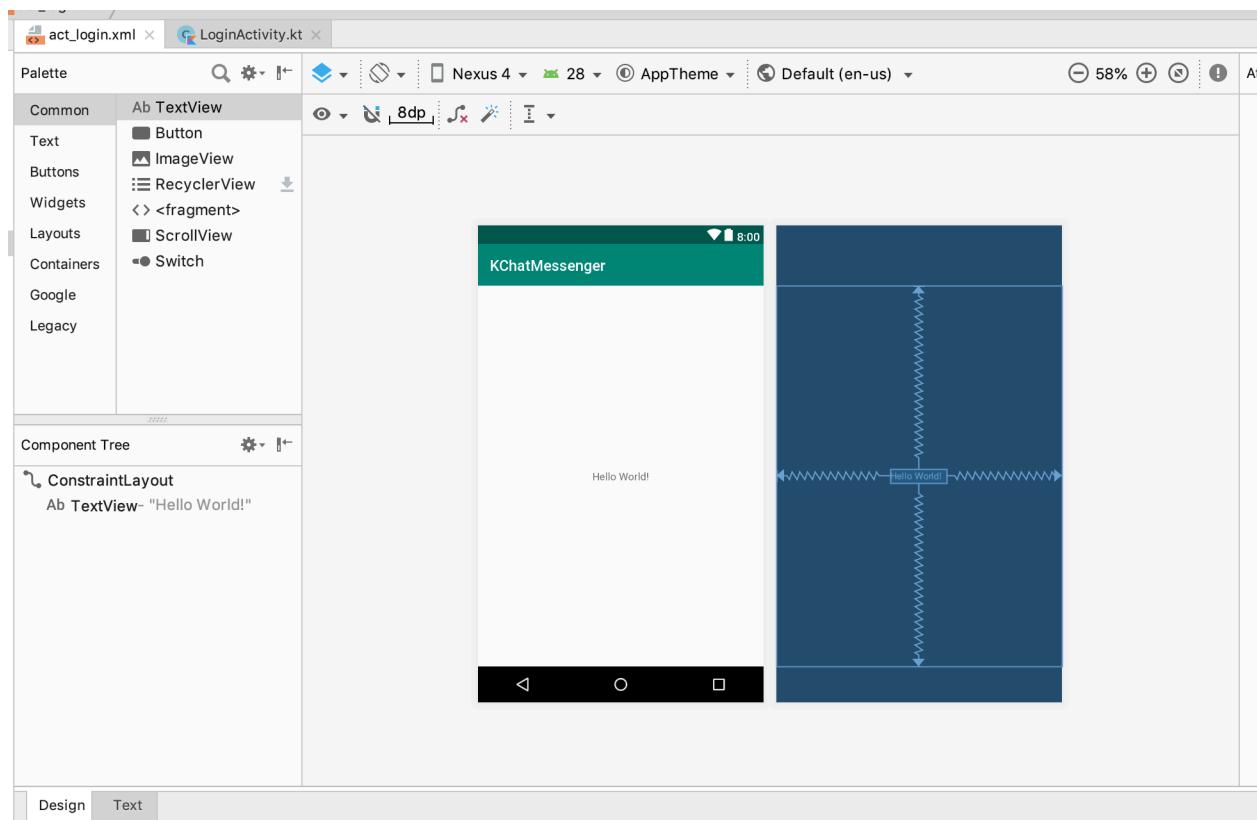


Figure 25: k004

Abra o diretório res/layout e procure pelo primeiro arquivo de layout **act_login.xml**.

Ao abrir o arquivo, um editor visual do Android Studio irá aparecer.

Precisaremos criar um layout para o formulário de login e depois para o de cadastro. A imagem a seguir mostra exatamente como ficará o layout no final do processo. Um Design com componentes mais elegantes e customizados.

De volta no arquivo **act_login.xml**, utilize o **zoomIn** que se encontra no canto superior direito para melhorar a visão do editor. Selecione o campo de texto padrão *Hello World* e apague-o.

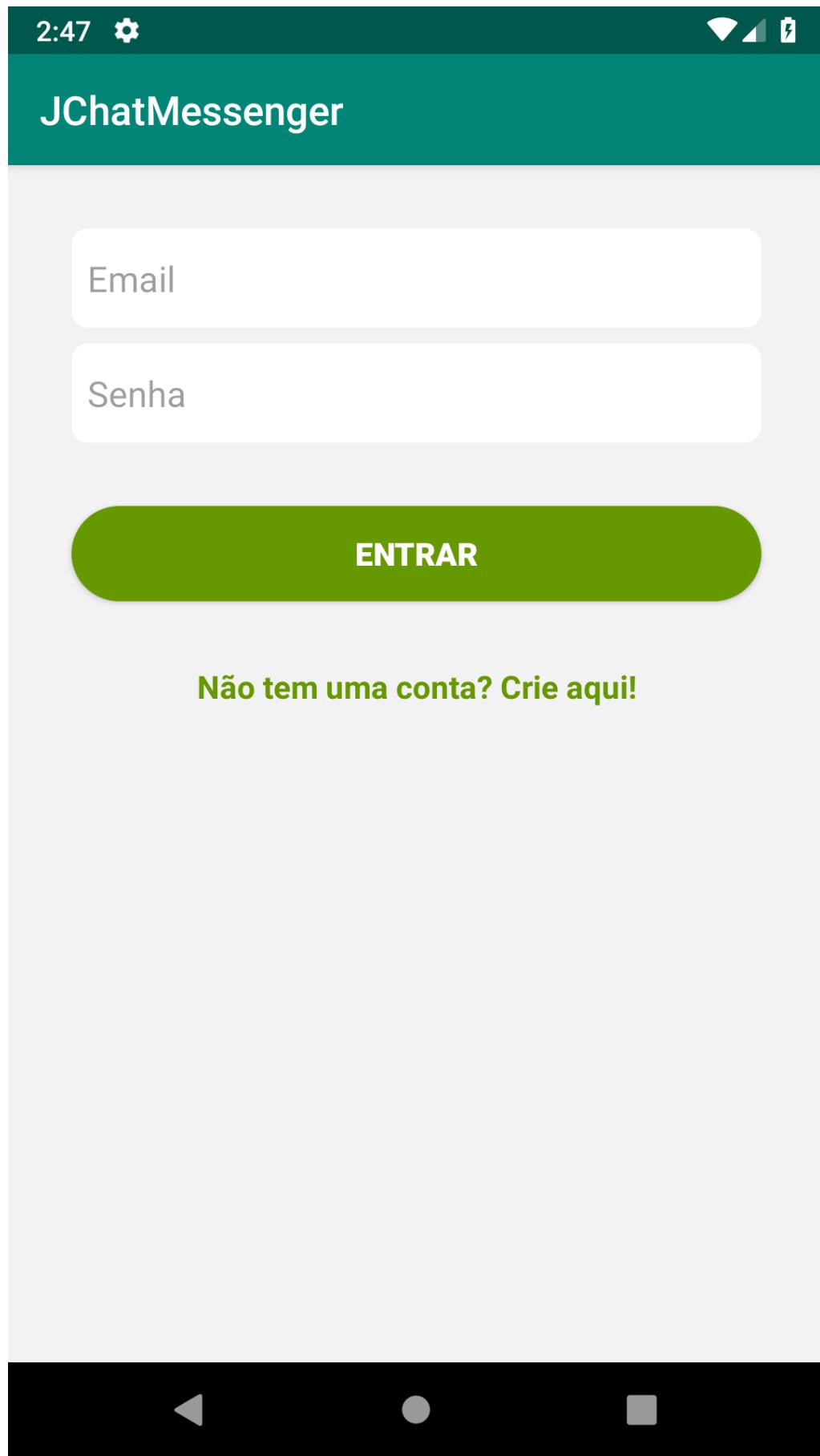
Agora selecione no painel ao lado o editor de texto **Plain Text** e arraste-o para dentro da tela do aplicativo.

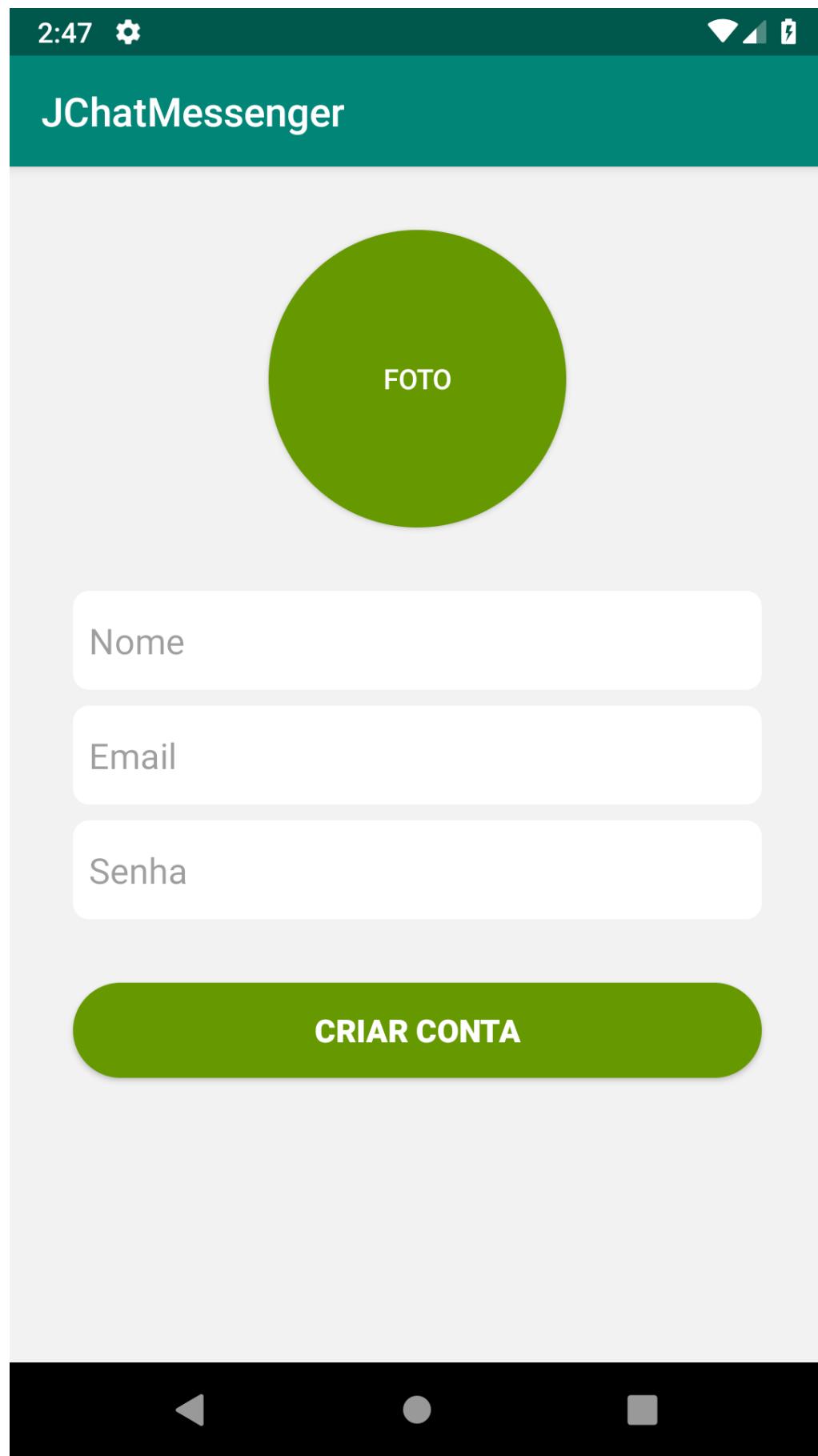
O próximo passo é muito importante. É aqui onde vamos definir as posições na tela de onde cada elemento irá aparecer.

No Android, os elementos são tratados por “regras” definidas em cada elemento e seu *pai*. Ou seja, um editor de texto pode depender de um botão ou de outro elemento para se posicionar.

Como estamos colocando nosso primeiro elemento, nós iremos fazer com que as posições laterais, superior e inferior se dimensione de acordo com o tamanho total da tela.

Faça o seguinte teste: adicione mais um editor de texto (**do tipo Password**) em qualquer lugar da tela e execute o aplicativo. Você vai notar que eles não respeitaram nenhuma posição porque ainda não definimos uma “regra” para cada elemento. Essas regras chamamos de





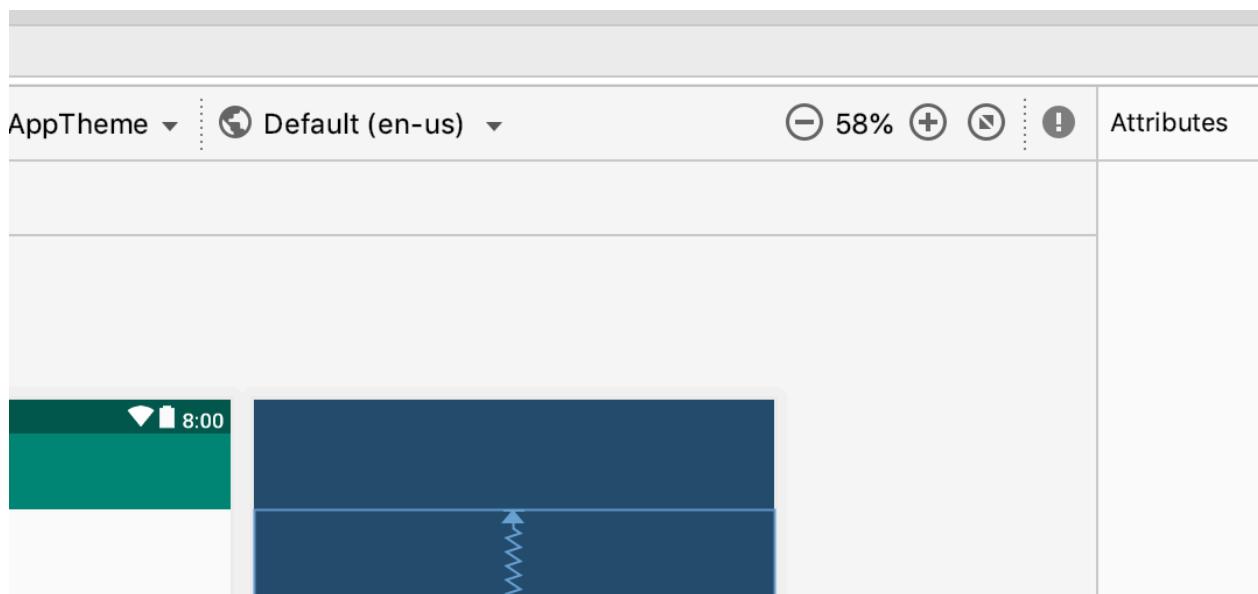


Figure 28: k007

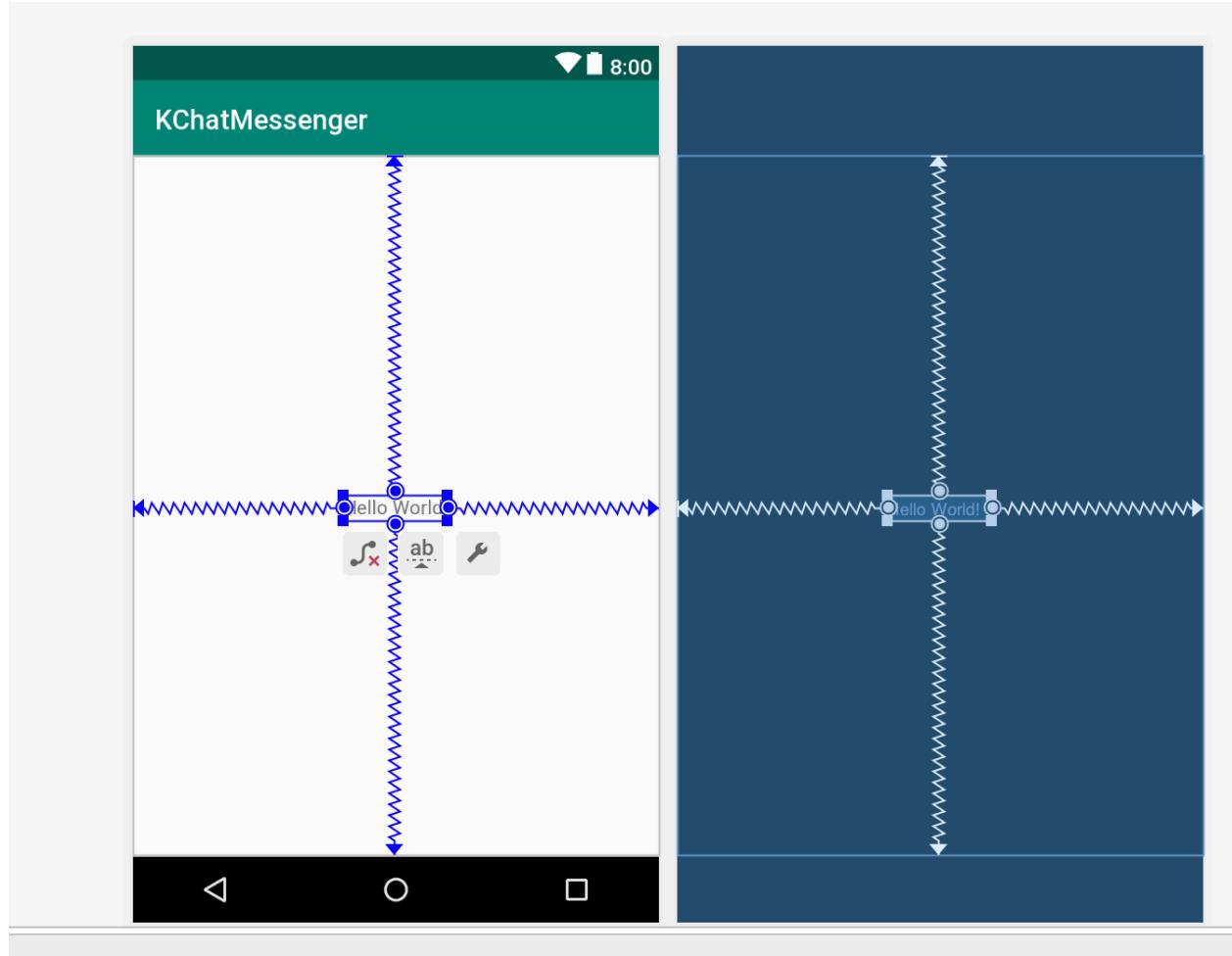


Figure 29: k008

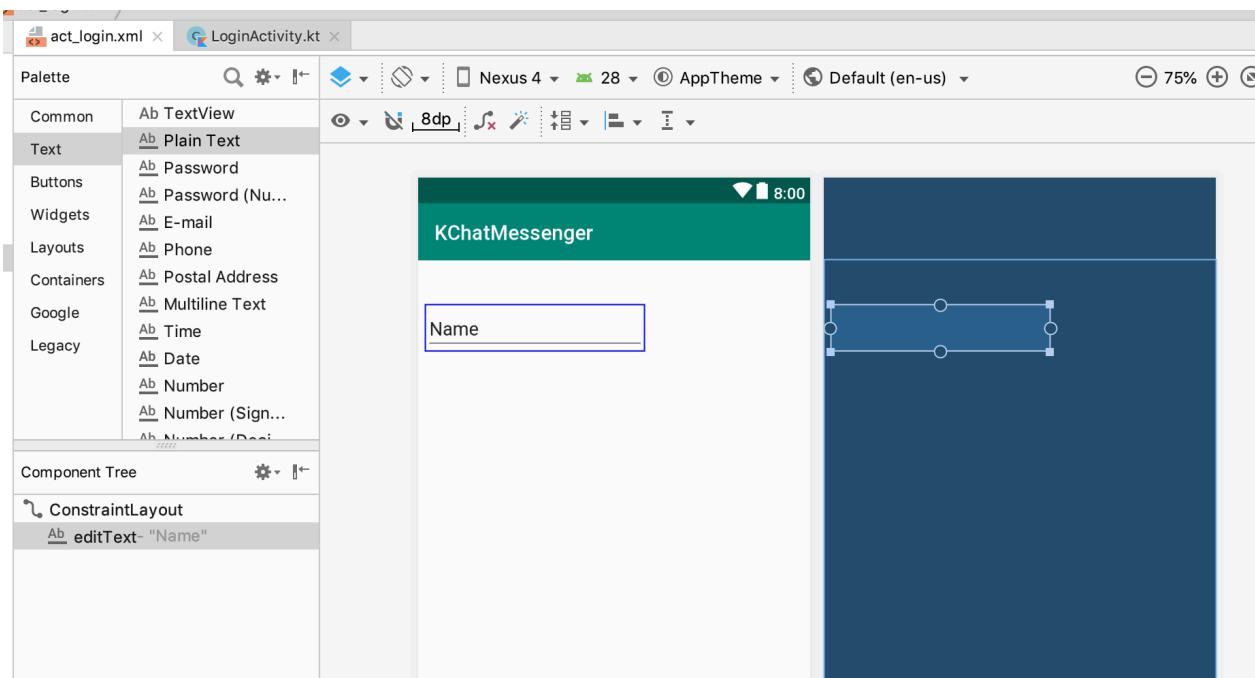


Figure 30: k009

constraints.

Para dizermos ao aplicativo que o editor de texto número 1 terá o alinhamento conforme mostramos no começo do capítulo, devemos criar as constraints. Segure e arraste o click nas “bolinhas” que aparecem nas extremidades do componente.

Ao segurar e arrastar a parte superior, esquerda e direita teremos algo parecido com isso.

Vamos analisar um pouco mais.

No canto superior a direita temos um painel com algumas informações:

1. ID: Indica uma referência única a este componente. Troque para **edit_email**.
2. Layout_width e layout_height: Esses campos possuem basicamente 2 opções.
 1. Wrap_content: Este valor indica que o elemento (neste caso a largura) deve respeitar o conteúdo do componente. Ou seja, ele encapsula a largura até o conteúdo aparecer. Geralmente usamos dessa forma quando temos um conteúdo dinâmico ou quando não queremos apenas usar constraints.
 2. Match_constraint: Aqui é onde a constraint é efetivada. Altere o status e veja o que acontece.

Ao mudar o atributo da largura, perceba que as constraints foram ativadas e a partir de agora, você deve definir o quanto de margin ela deve ter em relação ao container pai (no caso, a tela).

Isso é feito pelo quadrante ao lado onde temos:

- Top (topo): 16

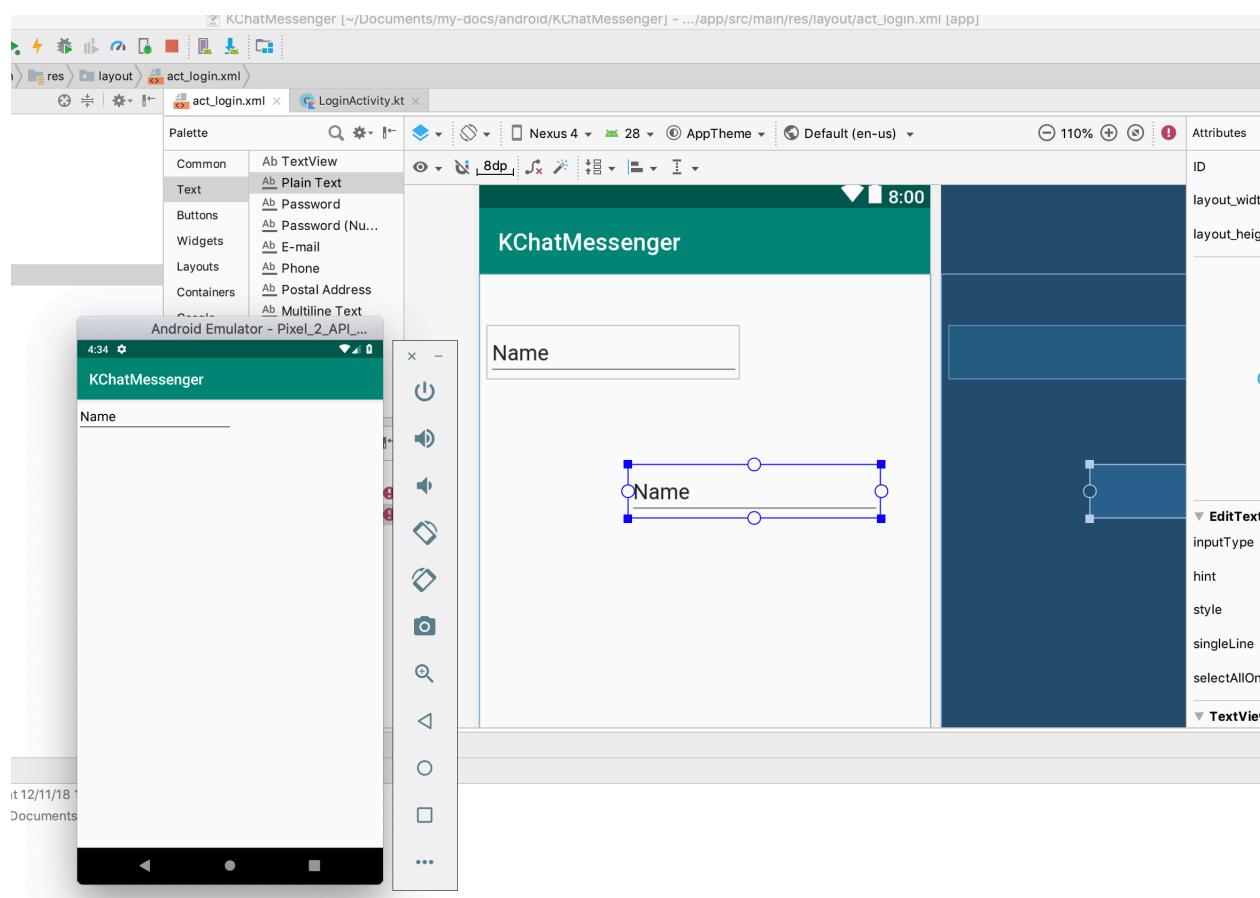


Figure 31: k010

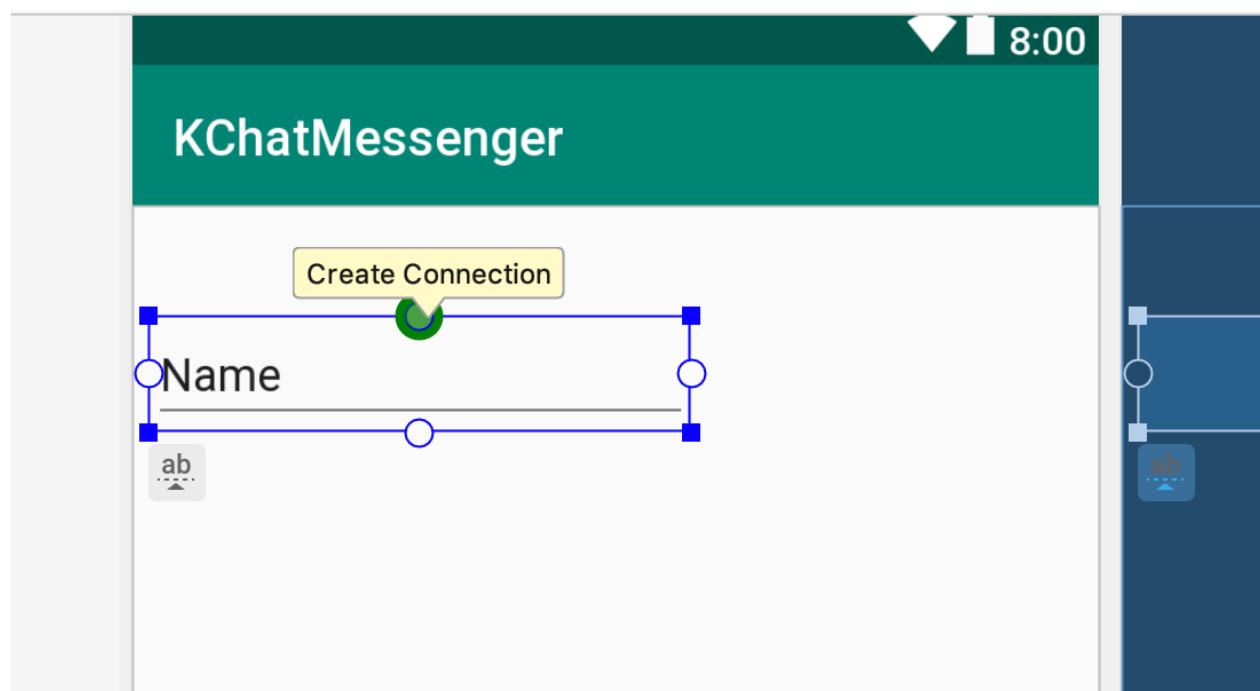


Figure 32: k011

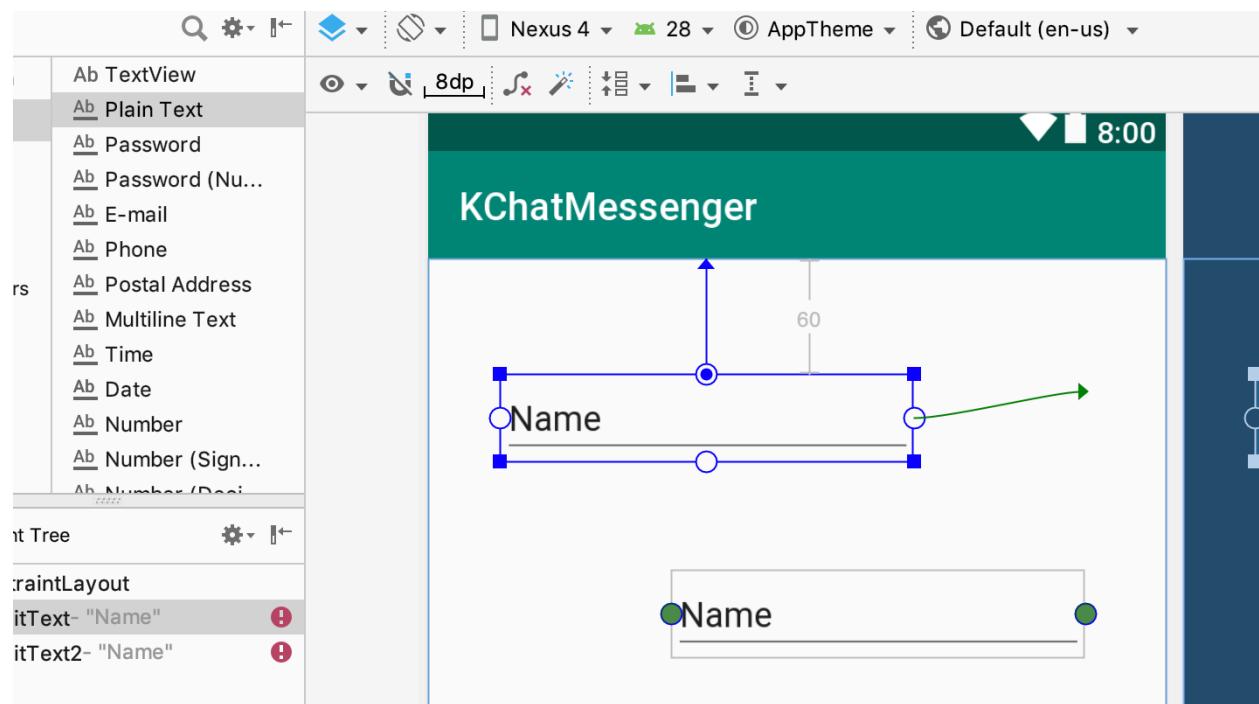


Figure 33: k012

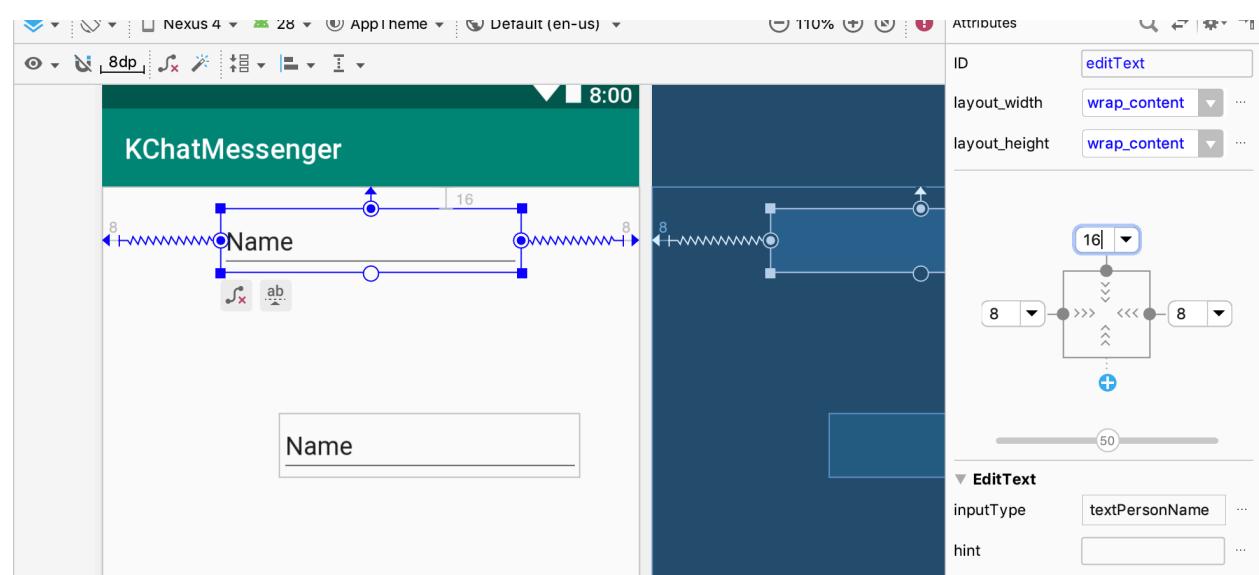


Figure 34: k013

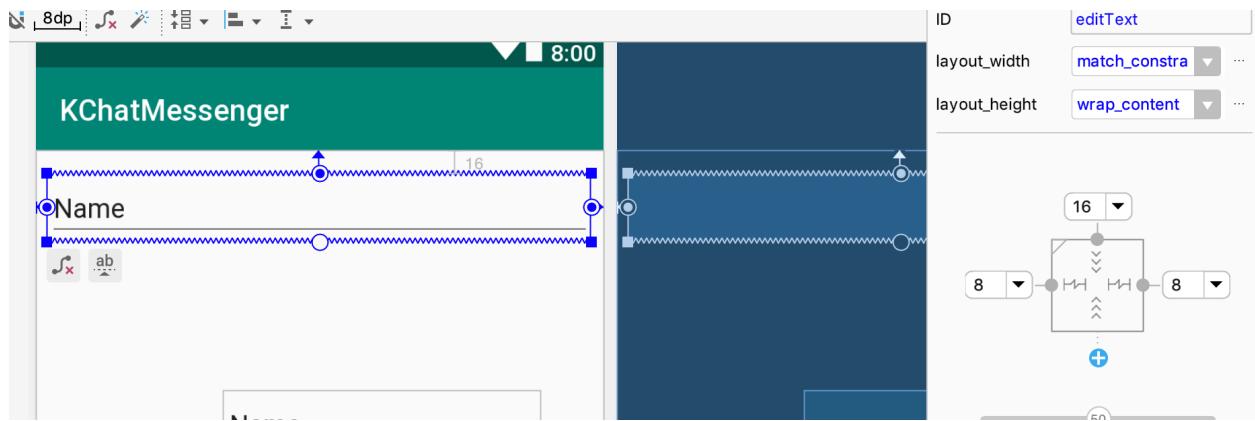


Figure 35: k014

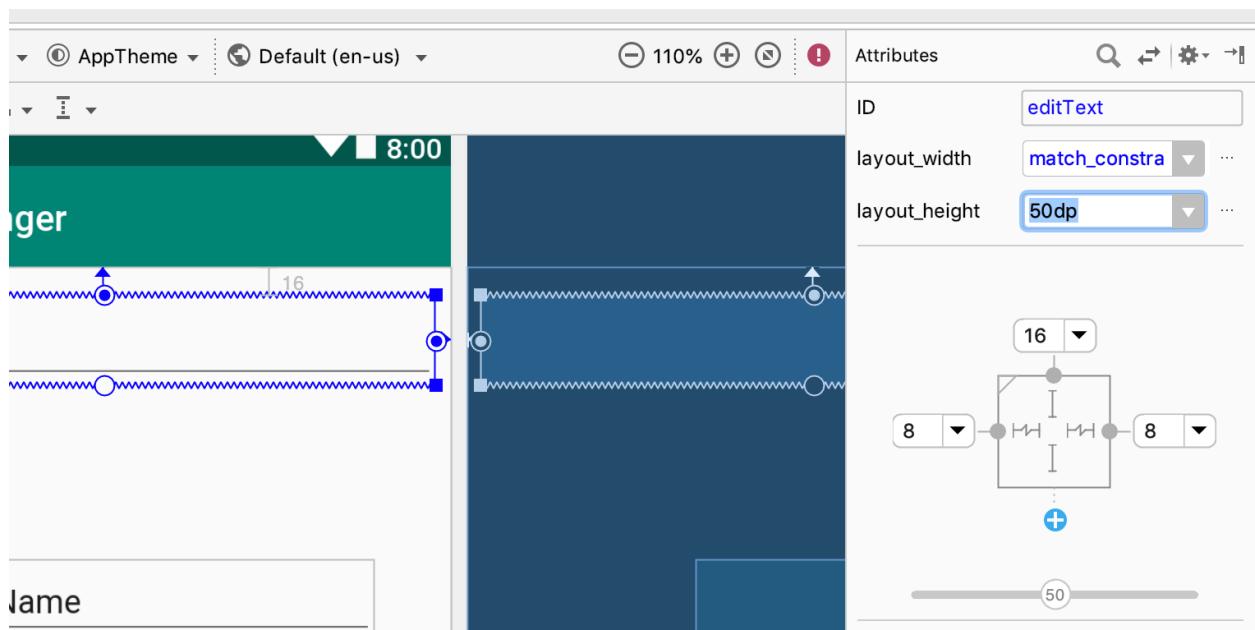


Figure 36: k015

- Left (esquerda): 8
- Right (direita): 8

Embaixo não temos, porque será o outro componente que irá se conectar a ele. O que podemos alterar agora é a altura **layout_height**, podemos colocar uma altura fixa como **50dp**.

DP é a unidade de medida em densidade de pixel do Android. Ele funciona desta forma para atender todo tipo de celular. Alguns com alta definição, outros menos.

Faça o mesmo com o próximo editor de texto.

Conecte o:

- topo do editor 2 embaixo do editor 1.
- esquerda do editor 2 na esquerda do editor 1 (margin 0 em relação ao editor 1).

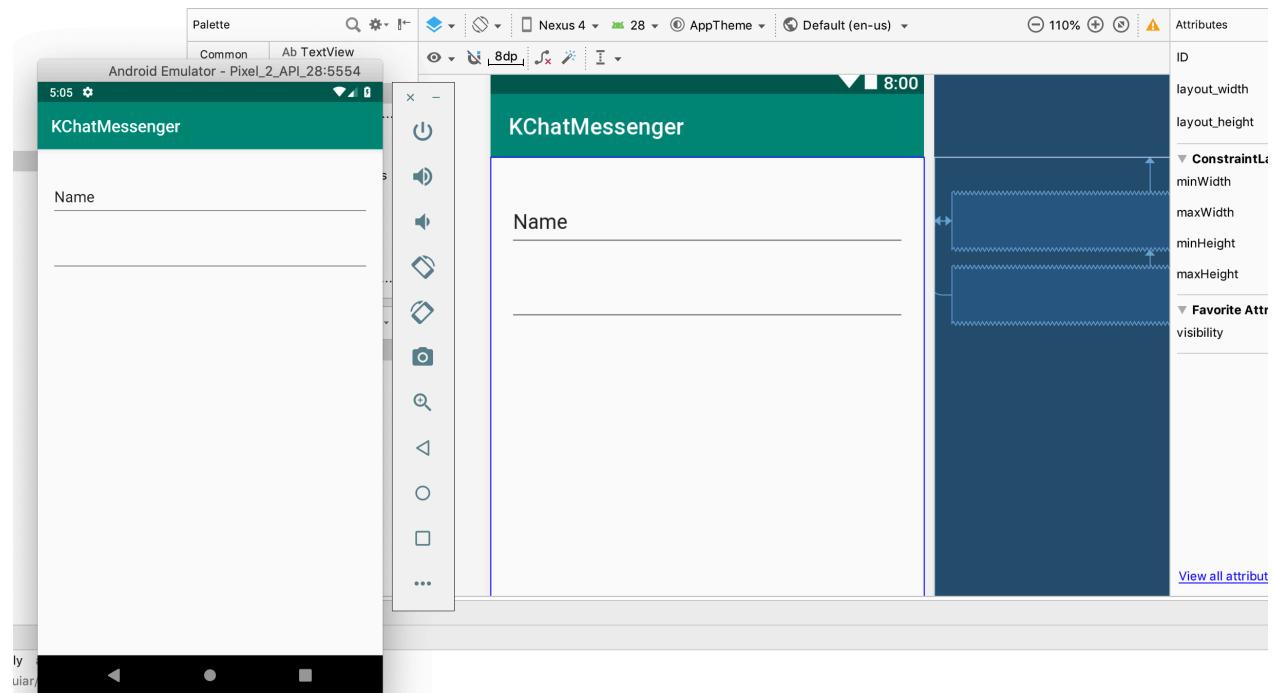


Figure 37: k016

- direita do editor 2 na direita do editor 1 (margin 0 em relação ao editor 1).
- largura (width) como *match_constraint*
- altura (height) como 50dp.

Agora altere o ID do editor 2 para ficar **edit_password**.

No fim, para deixar mais elegante, mantenha o editor de email com:

- Top: 32
- Left: 32
- Right: 32

E o editor de senha com:

- Top: 16
- Left: 0
- Right: 0

Execute o aplicativo e veja como ficou os dois campos de texto e email.

Clique em **Text** abaixo do editor Visual do Android Studio para visualizar a codificação do arquivo xml.

Seu arquivo deve estar semelhante a este abaixo:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity">

    <EditText
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:inputType="textPersonName"
        android:text="Name"
        android:ems="10"
        android:id="@+id/edit_email"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginEnd="32dp"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginRight="32dp"
        android:layout_marginStart="32dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="32dp"/>

    <EditText
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:inputType="textPassword"
        android:ems="10"
        android:id="@+id/edit_password"
        android:layout_marginTop="16dp"
        app:layout_constraintTop_toBottomOf="@+id/edit_email"
        app:layout_constraintEnd_toEndOf="@+id/edit_email"
        app:layout_constraintStart_toStartOf="@+id/edit_email"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Este arquivo de xml basicamente trabalha com componentes aninhados um dentro do outro. Veja que o container principal é o **ConstraintLayout** e dentro dele há 2 **EditText** no mesmo nível. Os atributos de largura, id, margins e todos os outros atributos do nosso componente ficará na TAG <Componente />.

Vamos alterar a cor de fundo do container.

De volta ao editor visual selecione o container clicando em qualquer lugar da tela do aplicativo e então nas duas setas:

Logo abaixo você verá o seletor de *background*. Clique nele.

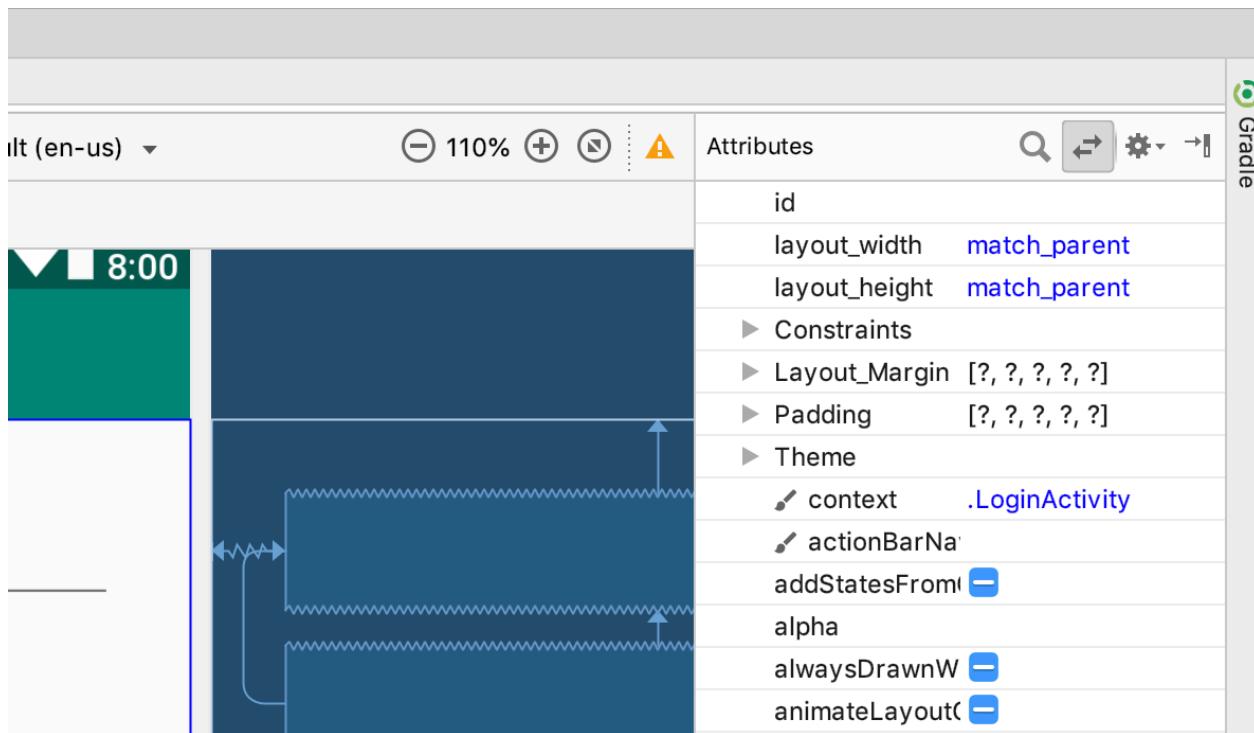


Figure 38: k017

Selecione **Add new resource** e então **New Color Value**. Defina um nome para sua nova cor como **bgColor**.

Feito isso, o fundo da tela ficará com a cor @color/bgColor.

Volte ao editor de código do arquivo xml e veja que foi adicionado uma referência ao arquivo de recursos **colors.xml** no container.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity" android:background="@color/bgColor">
```

Abra o arquivo `res/values/colors.xml` e veja que uma nova cor foi adicionada no padrão hexadecimal de cores.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="bgColor">#f3f3f3</color>
</resources>
```

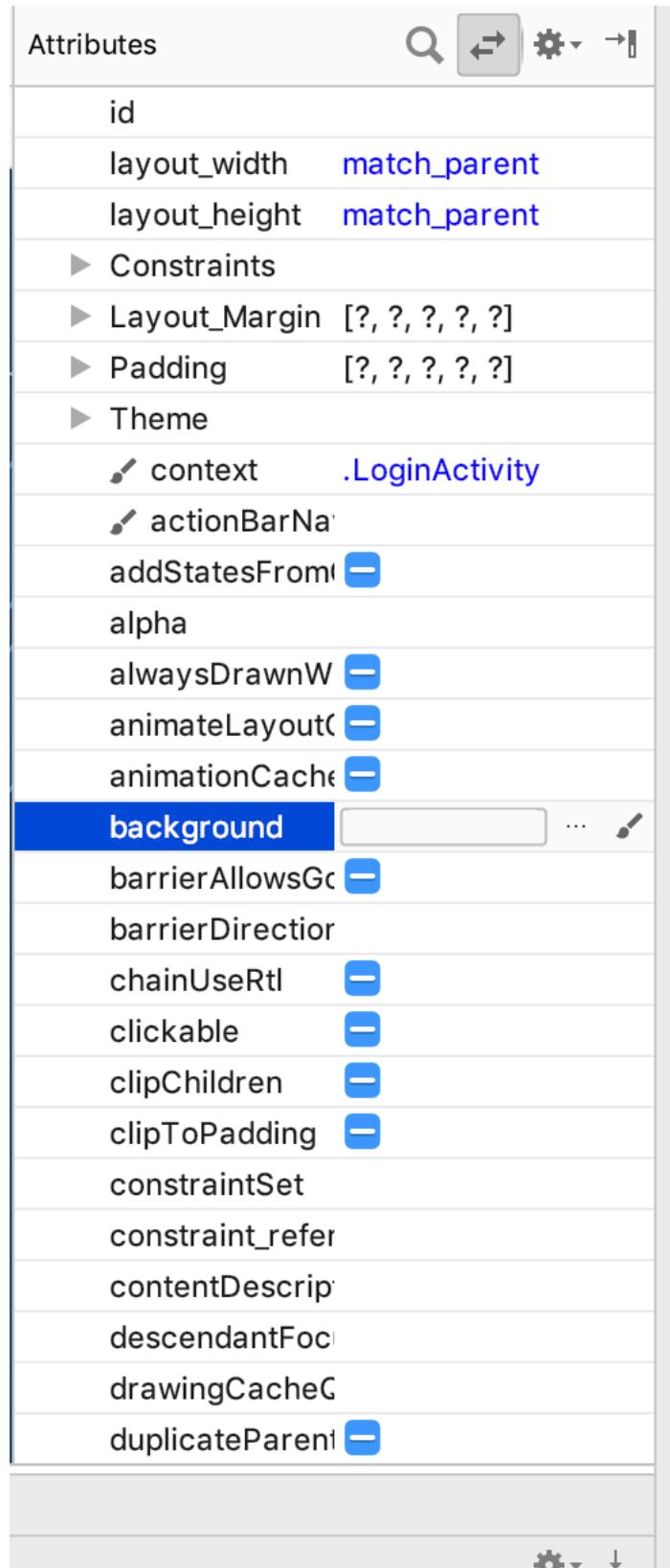


Figure 39: k018

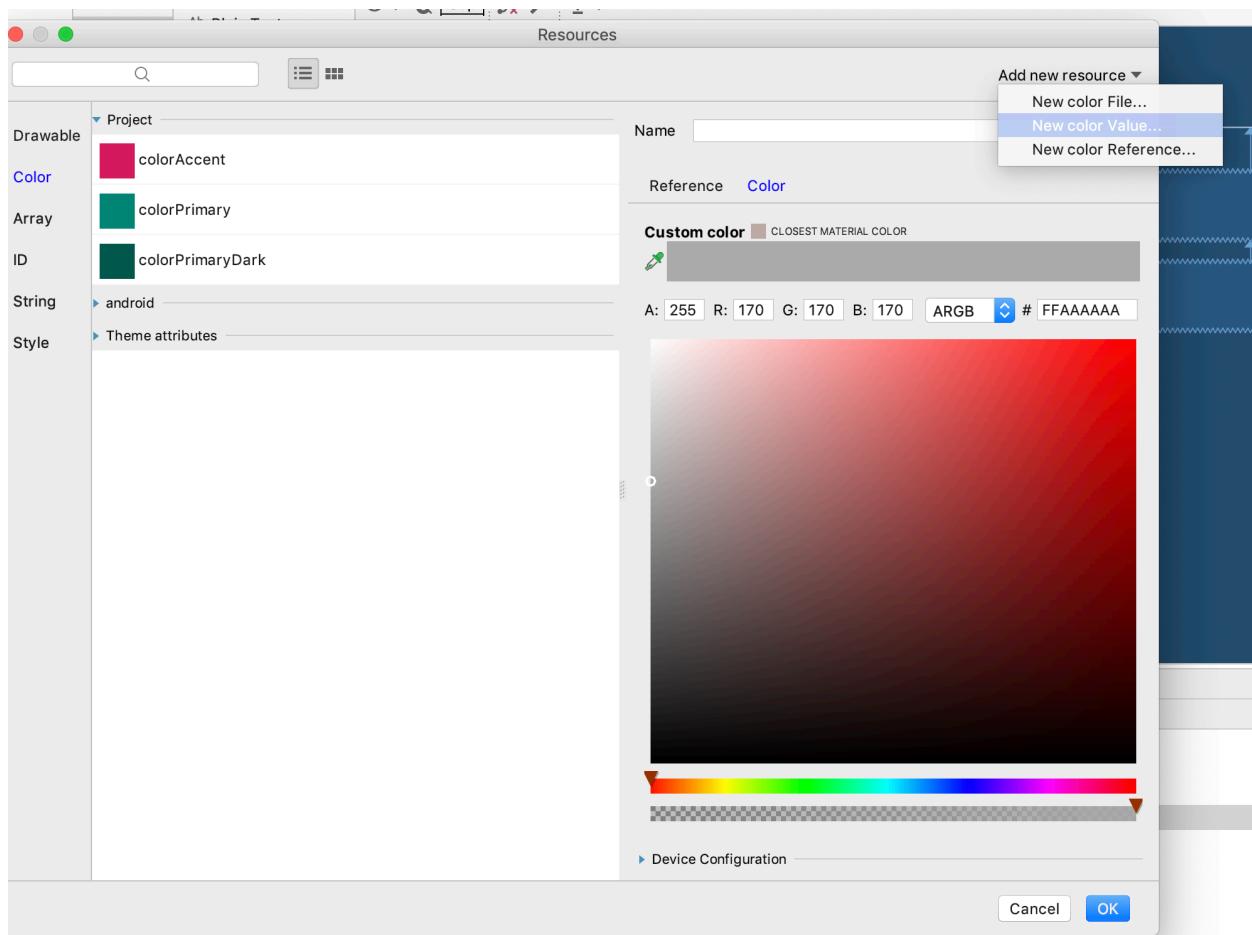


Figure 40: k020

Faça o mesmo para o fundo dos editores de textos. Só que desta vez, ao invés de criar uma cor nova, você pode usar uma cor já definida no SDK do Android como a cor branca.

Uma alternativa ao editor visual é criar o hábito de fazer alterações direto no código.

Altere o fundo do editor definindo o `android:background="@android:color/white"`.

Isso faz com que o seu aplicativo “olhe” para o arquivo de recursos de cores do SDK e não o que você cria.

Exemplo de como deve ficar o seu editor de texto.

<EditText

```
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:inputType="textPersonName"
    android:text="Name"
    android:ems="10"
    android:id="@+id/edit_email"
    android:layout_marginTop="32dp"
    app:layout_constraintTop_toTopOf="parent"
    android:layout_marginEnd="32dp"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginRight="32dp"
    android:background="@android:color/white"
    android:layout_marginStart="32dp"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginLeft="32dp" />
```

Agora vamos adicionar um *placeholder* que indique o que aquele campo espera de valor.

Adicione a tag **hint** para definir o *placeholder*. O Texto também deverá ser definido no arquivo de recursos de texto chamado **strings.xml**. E por fim, deve-se adicionar uma referência para esse recurso dentro do componente.

Quando você escrever `android:hint="@string/email"` ele mostrará um erro porque ainda não criamos esse textos. Uma forma rápida de fazer isso é a tecla de atalho Alt / Option (Mac) + Enter. Ele abrirá um PopUp para você criar o texto.

<resources>

```
<string name="app_name">KChatMessenger</string>
<string name="email">Email</string>
</resources>
```

Apague o texto que foi digitado `android:text="Name"`.

Faça o mesmo processo para a senha `android:hint="@string/password"`.

Só que os *placeholders* ficaram um pouco estranhos... muito próximos do início do editor de texto.

Vamos adicionar um **padding** lateral a esquerda para corrigir esse layout `android:paddingLeft="8dp"`.

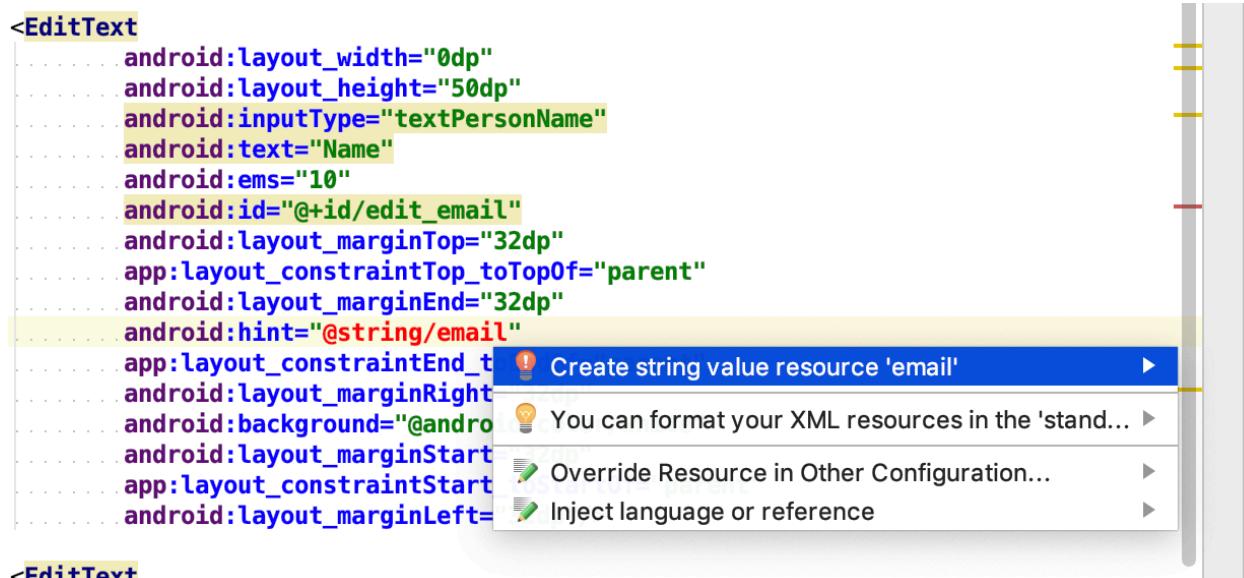


Figure 41: k021

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity"
    android:background="@color/bgColor">

    <EditText
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/edit_email"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginEnd="32dp"
        android:hint="@string/email"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginRight="32dp"
        android:background="@android:color/white"
        android:layout_marginStart="32dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="32dp"/>

```

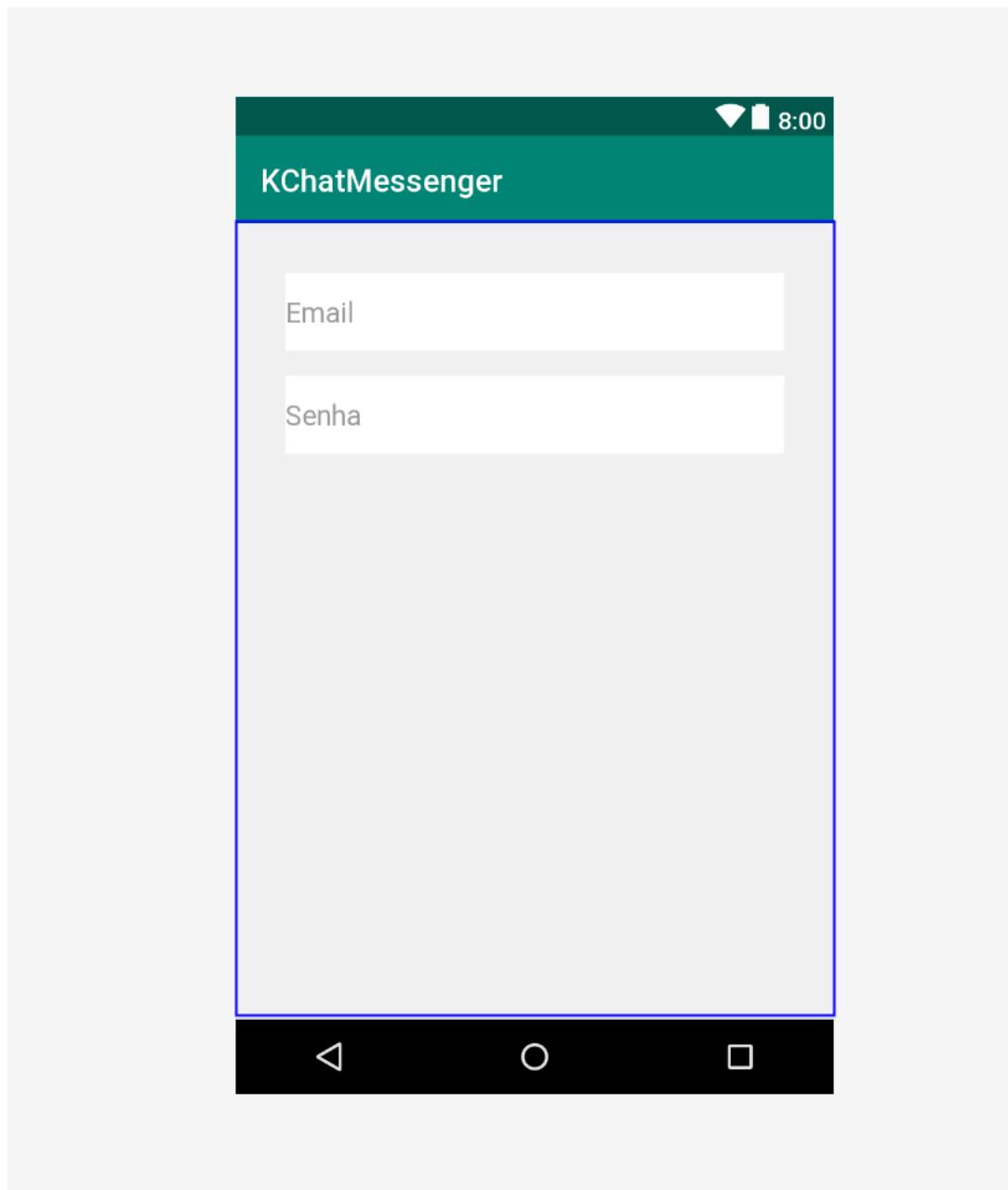


Figure 42: k022

```

<EditText
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:inputType="textPassword"
    android:hint="@string/password"
    android:ems="10"
    android:id="@+id/edit_password"
    android:background="@android:color/white"
    android:layout_marginTop="16dp"
    android:paddingLeft="8dp"
    app:layout_constraintTop_toBottomOf="@+id/edit_email"
    app:layout_constraintEnd_toEndOf="@+id/edit_email"
    app:layout_constraintStart_toStartOf="@+id/edit_email" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Adicione o botão embaixo do editor de texto de senha para depois conseguirmos efetuar o login.

O processo é o mesmo:

1. Arrastar o componente
2. Alterar o ID: **btn_enter**
3. Criar as constraints: Top 32, Left 0, Right 0
4. Alterar o fundo do botão
5. Alterar o texto do botão

```

<Button
    android:text="Button"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:id="@+id	btn_enter"
    android:layout_marginTop="32dp"
    android:background="@android:color/holo_green_dark"
    android:textColor="@android:color/white"
    app:layout_constraintTop_toBottomOf="@+id/editText3"
    app:layout_constraintEnd_toEndOf="@+id/editText3"
    app:layout_constraintStart_toStartOf="@+id/editText3" />

```

Segue abaixo a primeira versão final do layout da tela de login.

act_login.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

```

```

xmlns:app="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".LoginActivity"
android:background="@color/bgColor">

<EditText
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:inputType="textPersonName"
    android:ems="10"
    android:id="@+id/edit_email"
    android:layout_marginTop="32dp"
    app:layout_constraintTop_toTopOf="parent"
    android:layout_marginEnd="32dp"
    android:hint="@string/email"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginRight="32dp"
    android:paddingLeft="8dp"
    android:background="@android:color/white"
    android:layout_marginStart="32dp"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginLeft="32dp"/>

<EditText
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:inputType="textPassword"
    android:hint="@string/password"
    android:ems="10"
    android:id="@+id/edit_password"
    android:background="@android:color/white"
    android:layout_marginTop="16dp"
    android:paddingLeft="8dp"
    app:layout_constraintTop_toBottomOf="@+id/edit_email"
    app:layout_constraintEnd_toEndOf="@+id/edit_email"
    app:layout_constraintStart_toStartOf="@+id/edit_email"/>

<Button
    android:text="@string/enter"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:id="@+id/btn_enter"
    android:layout_marginTop="32dp"
    android:background="@android:color/holo_green_dark"
    android:textColor="@android:color/white"
    app:layout_constraintTop_toBottomOf="@+id/edit_password"
    app:layout_constraintEnd_toEndOf="@+id/edit_password"

```

```

        app:layout_constraintStart_toStartOf="@+id/edit_password" />

</androidx.constraintlayout.widget.ConstraintLayout>

string.xml

<resources>
    <string name="app_name">KChatMessenger</string>
    <string name="email">Email</string>
    <string name="password">Senha</string>
    <string name="enter">Entrar</string>
</resources>

```

Tente agora adicionar o componente **TextView** com a mensagem: **Não tem uma conta? Crie aqui!**.

O processo é muito semelhante ao que fizemos até agora. Ao final irei mostrar como ficará o arquivo final.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity" android:background="@color/bgColor">

    <EditText
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/edit_email"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginEnd="32dp"
        android:hint="@string/email"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginRight="32dp"
        android:paddingLeft="8dp"
        android:background="@android:color/white"
        android:layout_marginStart="32dp"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="32dp" />

    <EditText

```

```

        android:layout_width="0dp"
        android:layout_height="50dp"
        android:inputType="textPassword"
        android:hint="@string/password"
        android:ems="10"
        android:id="@+id/edit_password"
        android:background="@android:color/white"
        android:layout_marginTop="16dp"
        android:paddingLeft="8dp"
        app:layout_constraintTop_toBottomOf="@+id/edit_email"
        app:layout_constraintEnd_toEndOf="@+id/edit_email"
        app:layout_constraintStart_toStartOf="@+id/edit_email"/> >

<Button
        android:text="@string/enter"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:id="@+id/btn_enter"
        android:layout_marginTop="32dp"
        android:background="@android:color/holo_green_dark"
        android:textColor="@android:color/white"
        app:layout_constraintTop_toBottomOf="@+id/edit_password"
        app:layout_constraintEnd_toEndOf="@+id/edit_password"
        app:layout_constraintStart_toStartOf="@+id/edit_password"/> >

<TextView
        android:text="@string/account"
        android:textColor="@android:color/holo_green_dark"
        android:textStyle="bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txt_account"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toBottomOf="@+id/btn_enter"
        app:layout_constraintEnd_toEndOf="@+id/btn_enter"
        app:layout_constraintStart_toStartOf="@+id/btn_enter"/> >

</androidx.constraintlayout.widget.ConstraintLayout>

```

Perceba o conceito de **wrap_content**. Isso faz a largura respeitar o limite do conteúdo (que neste caso são os caracteres do texto).

11.3 Criando Bordas Arredondadas

Para criar bordas arredondadas precisamos criar estilos customizados para nossos botões e editores de textos. Estes arquivos customizados também são criados via xml. Só que

desta vez vamos adicionar um novo arquivo na pasta **res/drawable** - estes arquivos são **bg_edittext_rounded.xml** e **bg_button_rounded.xml**.

bg_edittext_rounded.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">

    <solid android:color="@android:color/white" />
    <corners android:radius="8dp" />

</shape>
```

bg_button_rounded.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">

    <solid android:color="@android:color/holo_green_dark" />
    <corners android:radius="25dp" />

</shape>
```

Estes arquivos *drawable* são úteis para manipularmos alguns atributos e formas de um componente. A tag **shape** nos indica o tipo de atributo que vamos customizar - no caso, a forma do componente.

Solid e **Corners** nos permite atribuir cores de fundo e manipular as bordas do componente, respectivamente. Os corners estão diretamente relacionados a altura do componente. Note que os botões possuem mais *radius* para garantir bordas mais arredondadas.

Agora, basta trocarmos o atributo **android:background="@android:color/holo_green_dark"** para uma referência do recurso *drawable*

android:background="@drawable/bg_button_rounded".

<Button

```
    android:text="@string/enter"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:id="@+id	btn_enter"
    android:layout_marginTop="32dp"
    android:background="@drawable/bg_button_rounded"
    android:textColor="@android:color/white"
    app:layout_constraintTop_toBottomOf="@+id/edit_password"
    app:layout_constraintEnd_toEndOf="@+id/edit_password"
    app:layout_constraintStart_toStartOf="@+id/edit_password"/>
```

11.4 Tela de Cadastro de Usuário

Você já aprendeu como criar um novo layout, trabalhar com constraints, adicionar componentes a tela e customizar componentes com bordas arredondadas e cores usando drawables. Isso te da o poder de criar praticamente qualquer formulário de um aplicativo.

Dito isto, faça o exercício de criar seu próximo layout - **tela de cadastro de usuário**.

Sei que a tela de cadastro possui a foto do perfil do usuário, mas não se preocupe com isso agora, apenas adicione os campos:

- Nome
- Email
- Senha
- Botão Cadastrar

Basicamente o processo de criação é identico a tela de login. Por isso, tente criar por conta própria. Logo abaixo, estará disponível a versão da tela de login (sem a foto).

act_register.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/bgColor"
    tools:context=".LoginActivity">

    <EditText
        android:id="@+id/edit_name"
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:layout_marginStart="32dp"
        android:layout_marginLeft="32dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:layout_marginRight="32dp"
        android:background="@drawable/bg_edittext_rounded"
        android:ems="10"
        android:hint="Nome"
        android:inputType="textPassword"
        android:paddingLeft="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
    >
```

```
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/edit_email"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    android:background="@drawable/bg_edittext_rounded"
    android:ems="10"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:paddingLeft="8dp"
    app:layout_constraintEnd_toEndOf="@+id/edit_name"
    app:layout_constraintStart_toStartOf="@+id/edit_name"
    app:layout_constraintTop_toBottomOf="@+id/edit_name" />

<EditText
    android:id="@+id/edit_password"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    android:ems="10"
    android:background="@drawable/bg_edittext_rounded"
    android:hint="Senha"
    android:inputType="textPassword"
    android:paddingLeft="8dp"
    app:layout_constraintEnd_toEndOf="@+id/edit_email"
    app:layout_constraintStart_toStartOf="@+id/edit_email"
    app:layout_constraintTop_toBottomOf="@+id/edit_email" />

<Button
    android:id="@+id/btn_register"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:background="@drawable/bg_button_rounded"
    android:text="@string/create_account"
    android:textAllCaps="true"
    android:textColor="@android:color/white"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="@+id/edit_password"
    app:layout_constraintStart_toStartOf="@+id/edit_password"
    app:layout_constraintTop_toBottomOf="@+id/edit_password" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

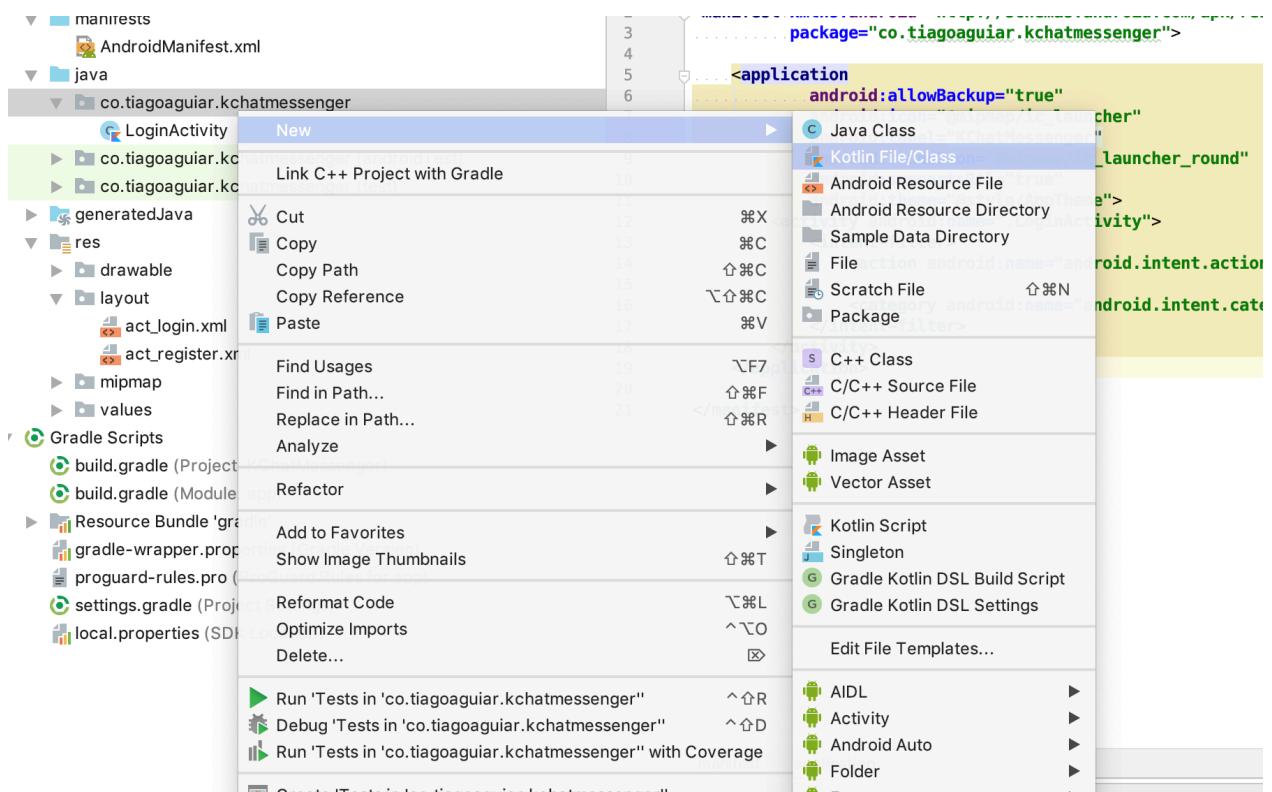
Neste momento, você ainda não conseguirá ver o resultado final da tela de cadastro do usuário porque não conseguimos chegar até essa tela. Para que ela funcione, precisamos criar eventos de click no touch do celular como por exemplo, quando clicamos no texto **Não tem uma conta? Crie aqui!**.

Nos próximos passos, vamos dar vida a tela de cadastro de usuário que, até então, temos apenas o *preview* pelo Android Studio. E também vamos por a mão na massa em códigos Kotlin!

11.5 Dando Vida a Nossa Atividade de Login

Abra o pacote java onde se encontra a Activity de Login e crie uma nova Activity chamada **RegisterActivity**.

Clique com o direito no pacote > New > Activity > Empty Activity



Tire a opção **generate layout file** porque já criamos um arquivo de layout. O que vamos fazer agora é vincular essa nova classe da atividade de registro ao layout que criamos anteriormente.

Se você ainda não comprehende muito bem o conceito de orientação a objetos, classes e etc, volte nos primeiros capítulos desse material para dominar esse paradigma o quanto antes.

Feito isso, você terá uma nova classe em Kotlin para a atividade de cadastro de usuário. As Activities representam a parte lógica por trás de cada tela e eventos/ação do usuário no

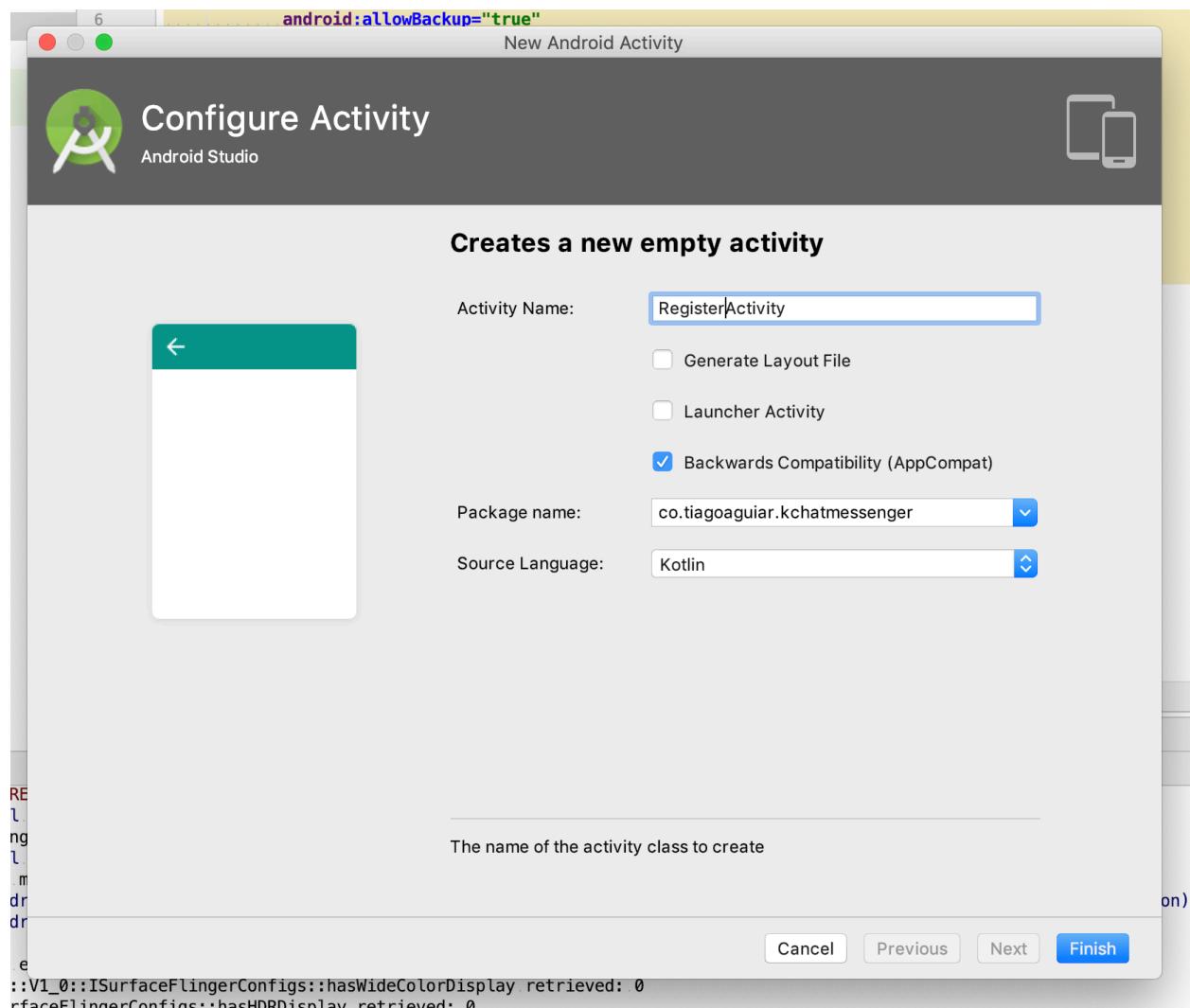


Figure 43: k024

smartphone. É ela que controla para onde o aplicativo deve ir - caso mudar de tela - ou quais mensagens mostrar ao usuário dependendo da regra e lógica definida.

Como usamos o Android Studio para nos ajudar a criar uma nova activity, podemos esquecer o passo de registrá-la no **AndroidManifest.xml** - porque o Android Studio já fez isto para nós.

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="co.tiagoaguiar.kchatmessenger">

    <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">
        <activity android:name=".LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".RegisterActivity">
        </activity>
    </application>

</manifest>

```

Sempre que uma nova Activity é criada, ela deve ser registrada no **AndroidManifest.xml**. Este é o arquivo de configuração do aplicativo que define várias coisas a respeito de como o aplicativo deve funcionar.

Nosso próximo passo é definir o layout que criamos e atribuir a nossa nova classe como sendo responsável pelo layout de registro de cadastro.

Para fazer isso, vamos abrir o arquivo **RegisterActivity.kt** e adicionar o layout com o **setContentView**.

```

package co.tiagoaguiar.kchatmessenger

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class RegisterActivity : AppCompatActivity() {

```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // adicionar Layout
    setContentView(R.layout.act_register)
}
}

```

A Classe **R** é a classe responsável por ter a referência de todos os recursos do Android como layout, menu, drawable, anim (Animações), raw (arquivos de mídia), etc. E ela que conecta a View com a Lógica - que chamamos de *Controllers*.

O próximo passo é fazer com que, quando o usuário clicar no **TextView**, o aplicativo execute a lógica de abrir essa nova Activity de Cadastro de usuário - **RegisterActivity**.

11.6 Interagindo Entre Activities

Para navegar entre activities usamos um recurso chamado de evento de clicks (ou *Listeners*). São objetos e eventos disparados quando ocorre algum tipo específico de evento com o componente.

Veja o seguinte código:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_login)

    txt_account.setOnClickListener {
        Log.i("Teste", "funciona")
    }
}

```

Neste código temos a referência do componente que está na tela **txt_account**. Essa referência é bem mais simples de importar com Kotlin pelo recurso de extensibilidade pelo import: **import kotlinx.android.synthetic.main.act_login.***. Dessa forma, as referências já estão disponíveis para usarmos aqui no código e trabalharmos com os componentes.

Com os componentes em mãos, podemos definir blocos de códigos que serão executados quando ocorrer algum tipo de evento. Neste primeiro caso, quando houver o evento de click que, será “escutado” pelo bloco **setOnClickListener {}** iremos imprimir no console de log a palavra **funciona..**

Execute o código acima, clique na frase **Não tem uma conta? Crie aqui!** e veja o que acontece...

Você verá uma mensagem clicando no LogCat do Android Studio.

Em resumo, podemos ter objetos que anônimos que ficam escutando e fazendo o papel de *Listeners* para diversos eventos do aplicativo.

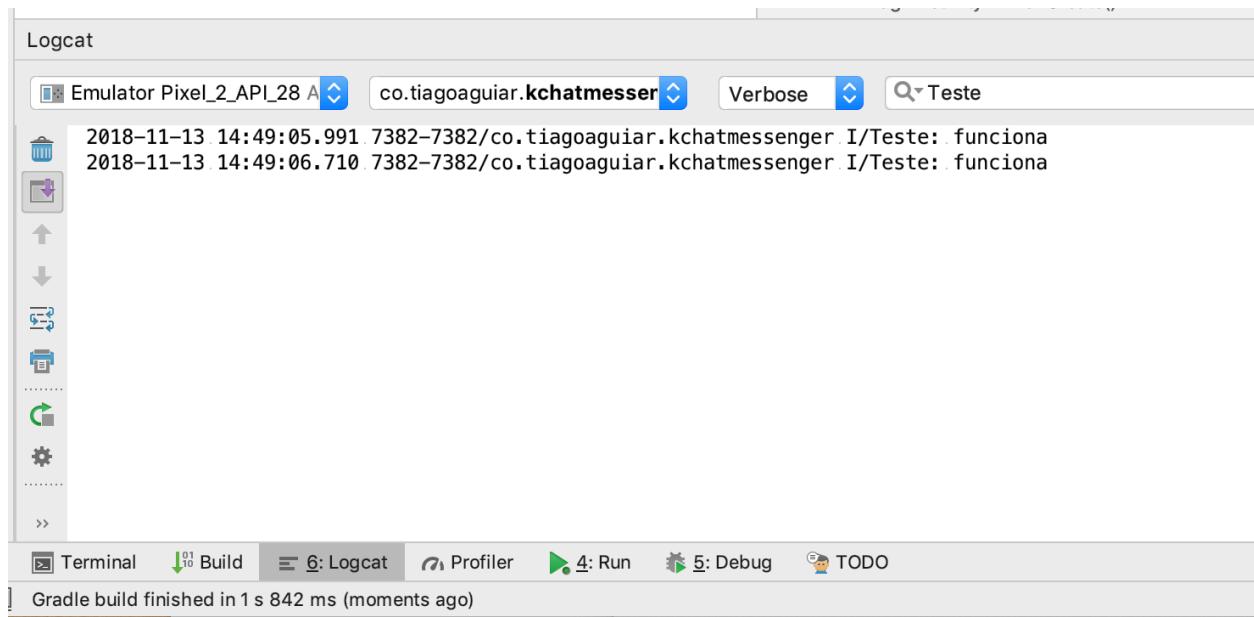


Figure 44: k025

Agora, vamos trocar o código de Log para um código onde abrirá uma nova Activity.

Para abrir uma nova Activity, precisamos usar o recurso de **Intenção - ou Intent**. Uma intenção de exibir um novo contexto para o aplicativo.

Digite o seguinte código e execute o seu aplicativo:

```
package co.tiagoaguiar.kchatmessenger

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.act_login.*

class LoginActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_login)

        txt_account.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java)
            startActivity(intent)
        }
    }
}
```

O que esse código realmente faz!?

Basicamente criamos uma nova Instância do objeto **Intent** que recebe dois parâmetros:

1. O Contexto atual que está executando, neste caso, seria a instância de um objeto **LoginActivity**.
2. A classe da Activity de destino que deve ser instanciada (pelo Android) e exibida como novo contexto atual.

Para que tudo funcione, basta executar o método fornecido em todas as activities - `startActivity(intent)`. Dessa forma, uma nova Activity é inicializada e a atual sai de cena.

No capítulo anterior, detalhamos a fundo o que cada linha de código representa na programação orientada a objetos. Se precisar, volte lá..

Na tela de registro, vamos pegar as referências dos campos editores de textos e imprimir o que o usuário preenche na tela.

Abra o arquivo **RegisterActivity**, e digite o seguinte código:

```
package co.tiagoaguiar.kchatmessenger

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import kotlinx.android.synthetic.main.act_register.*

class RegisterActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_register)

        btn_register.setOnClickListener {
            val email = edit_email.text.toString()
            val password = edit_password.text.toString()

            Log.i("Teste", "email: ${email} | senha: ${password}")
        }
    }
}
```

Da mesma forma como fizemos ao clicar no botão de login, podemos imprimir no console os valores que estão dentro dos editores de texto usando a sintaxe `edit_email.text.toString()` e assim por diante.

Podemos adicionar uma variável dentro de uma String para imprimir formatado, evitando concatenar Strings. `"email: ${email} | senha: ${password}"`.

Finalizamos o primeiro passo da construção do nosso Chat. Ainda temos alguns passos pela frente, mas é importante ressaltar o quanto evoluímos em nosso projeto: Design, Activities, Eventos de Listener e muito mais.

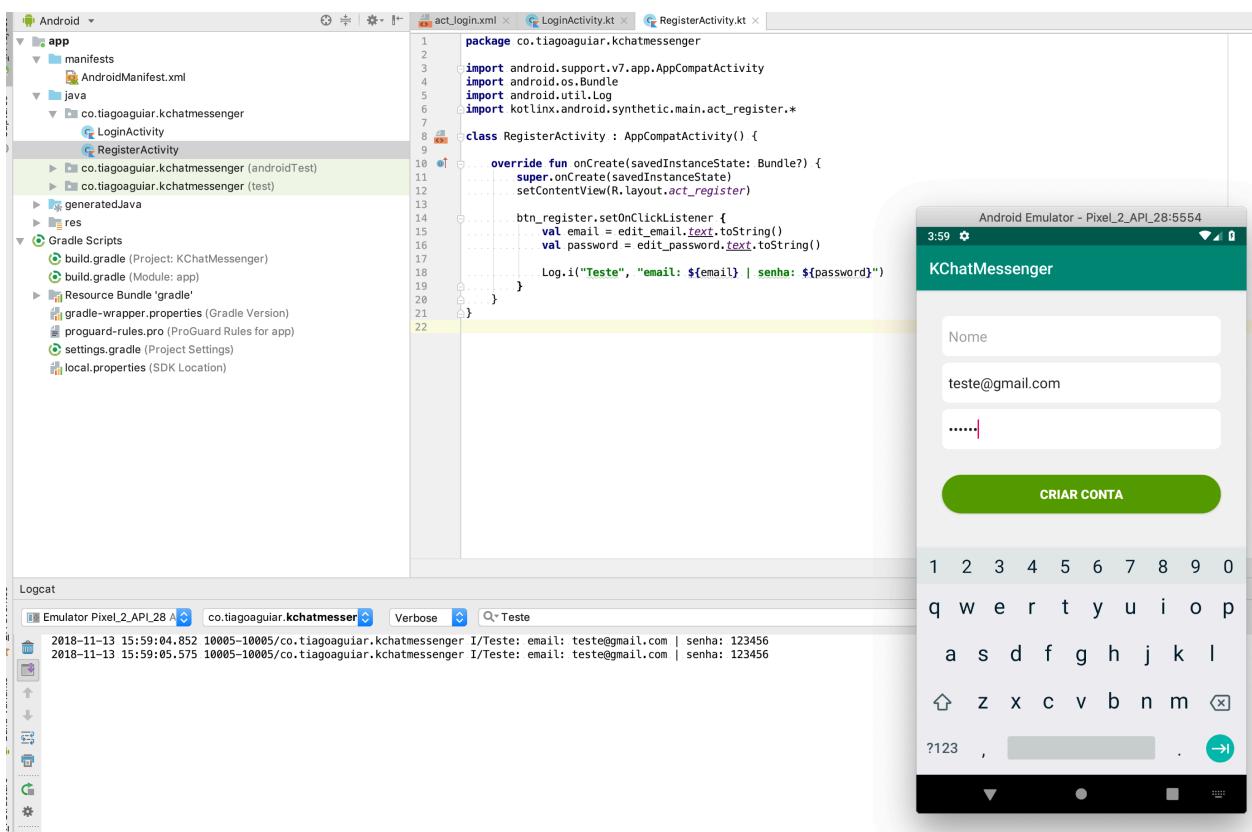


Figure 45: k026

Agora vamos começar a configurar nosso banco de dados em tempo real para criar as conversas e usuários do chat. A tecnologia que vamos utilizar será o **Firebase**. Um poderoso recurso criado pelo Google para facilitar o desenvolvimento de aplicativos sem ter, necessariamente, um servidor backend hospedados em algum lugar.

O próprio Firebase fará essa camada de prover os dados para nós. Então, vamos codificar!

11.7 Firebase: Inicialização e Configuração

Se tivesse que resumir o Firebase em uma frase eu diria que o **Firebase te ajuda a construir aplicativos rapidamente sem se preocupar com infraestrutura**.

Isso porque ele fornece alguns recursos como:

- Poderoso banco de dados em tempo real
- Autenticação com email, telefone e as principais redes sociais
- Funções de eventos de disparo
- Recursos de inteligência artificial voltado aos aplicativos
- Hospedagem
- Espaço de disco para armazenamento de mídias
- Crashlytics: Ferramenta para análise e relatório de Bug's no aplicativo
- E a lista continua...

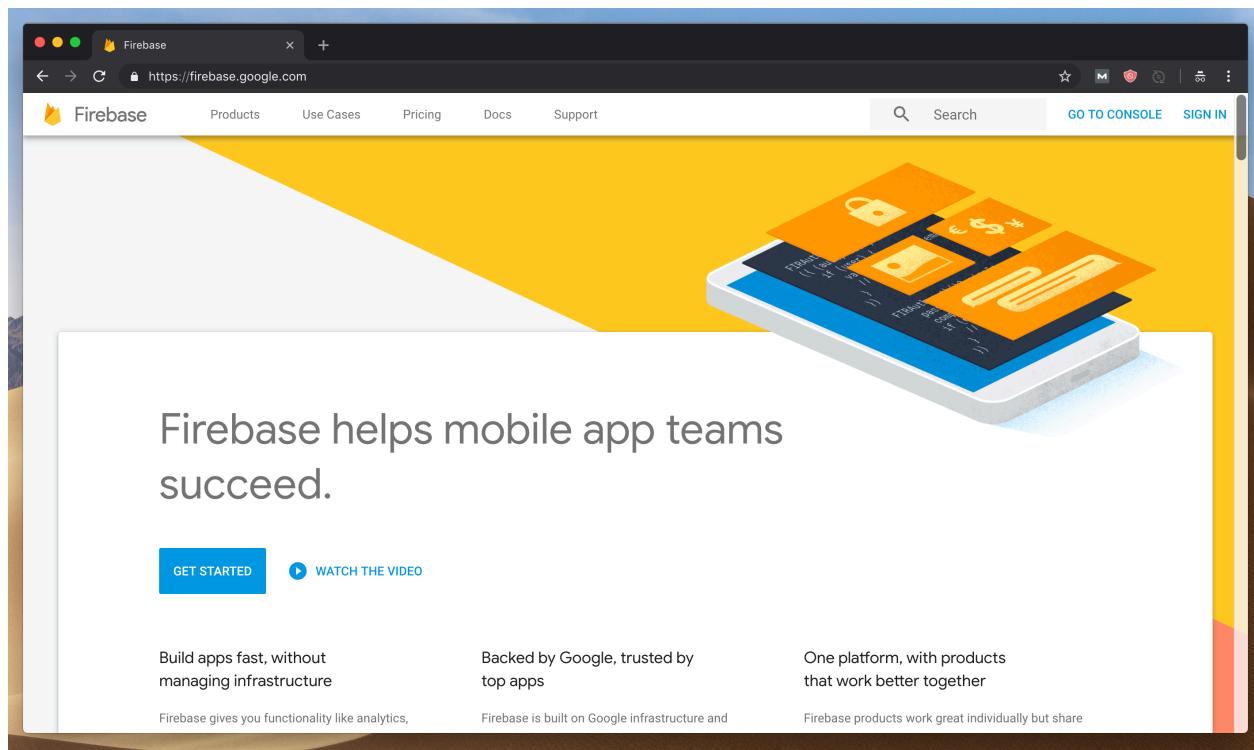


Figure 46: k027

Para conhecer tudo o que o Firebase oferece, acesse <https://firebase.google.com/products/>.

Por hora, vamos focar apenas na infraestrutura que o nosso aplicativo precisa. Um recurso de banco de dados em tempo real + Autenticação de usuários (Login/Cadastro).

Para começar a vincular a nova “infraestrutura” com o nosso projeto, primeiramente precisaremos de uma conta no Firebase.

Acesse a página <https://firebase.google.com>.

Clique em **Get Started** e efetue o login com uma conta Google. Caso não tenha, é bem simples criar um <https://gmail.com>.

Dentro do painel do Firebase, clique em **+ Add project** para começar um novo projeto e montar a infraestrutura.

Preencha o formulário com o nome do projeto, o project ID e aceite os termos de uso da plataforma.

Após clicar em **Continue** você será redirecionado para o *Dashboard* do Firebase.

Neste painel você encontrará todos os recursos que expliquei anteriormente. Alguns recursos são pagos, outros são cobrados sob demanda, mas não se preocupe porque o recurso que vamos usar não terá custo (até porque o número de acessos ao aplicativo será bem pequeno).

O Firebase nos ajuda (e muito) a criar um projeto novo e colocar na Google Play ou App Store. Porque basicamente só pagamos o serviço quando crescemos o negócio e isso é fantástico!

Welcome to Firebase!

Tools from Google for developing great apps,
your users, and earning more through mobile

[Learn more](#) [Documentation](#) [Support](#)

Recent projects

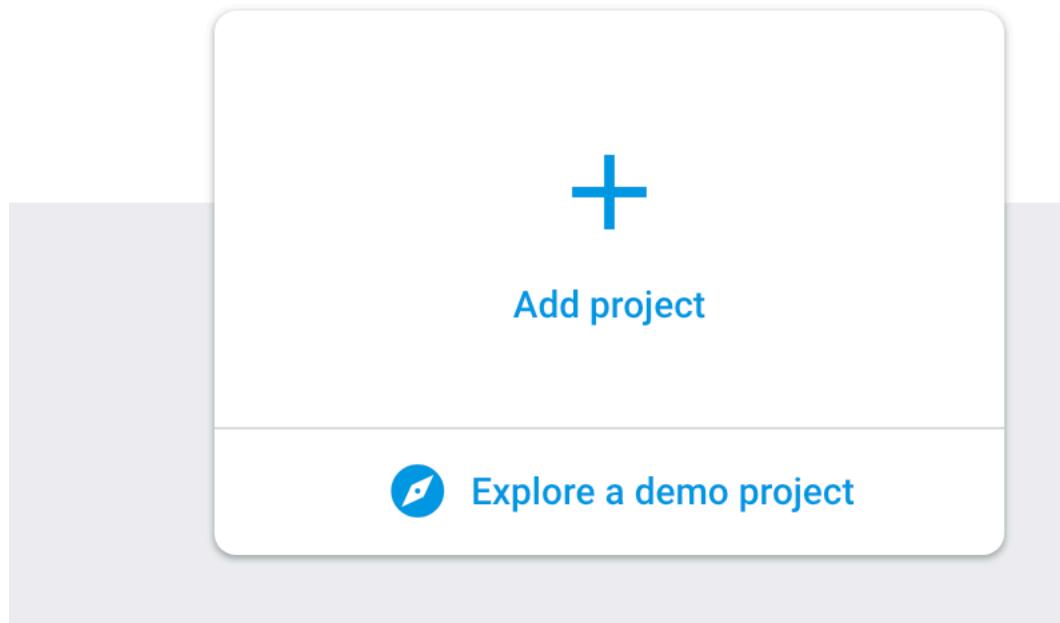


Figure 47: k028

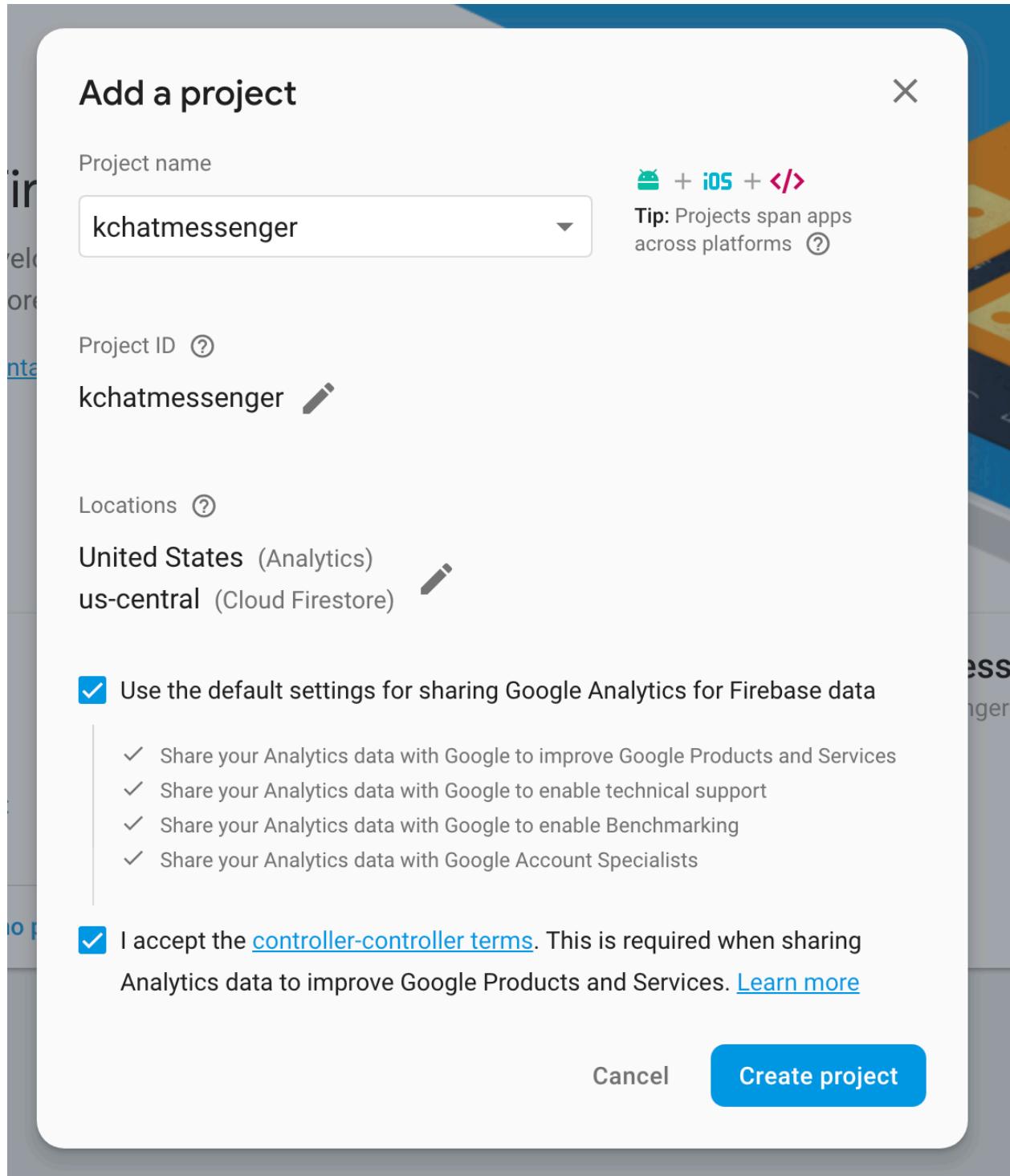


Figure 48: k029

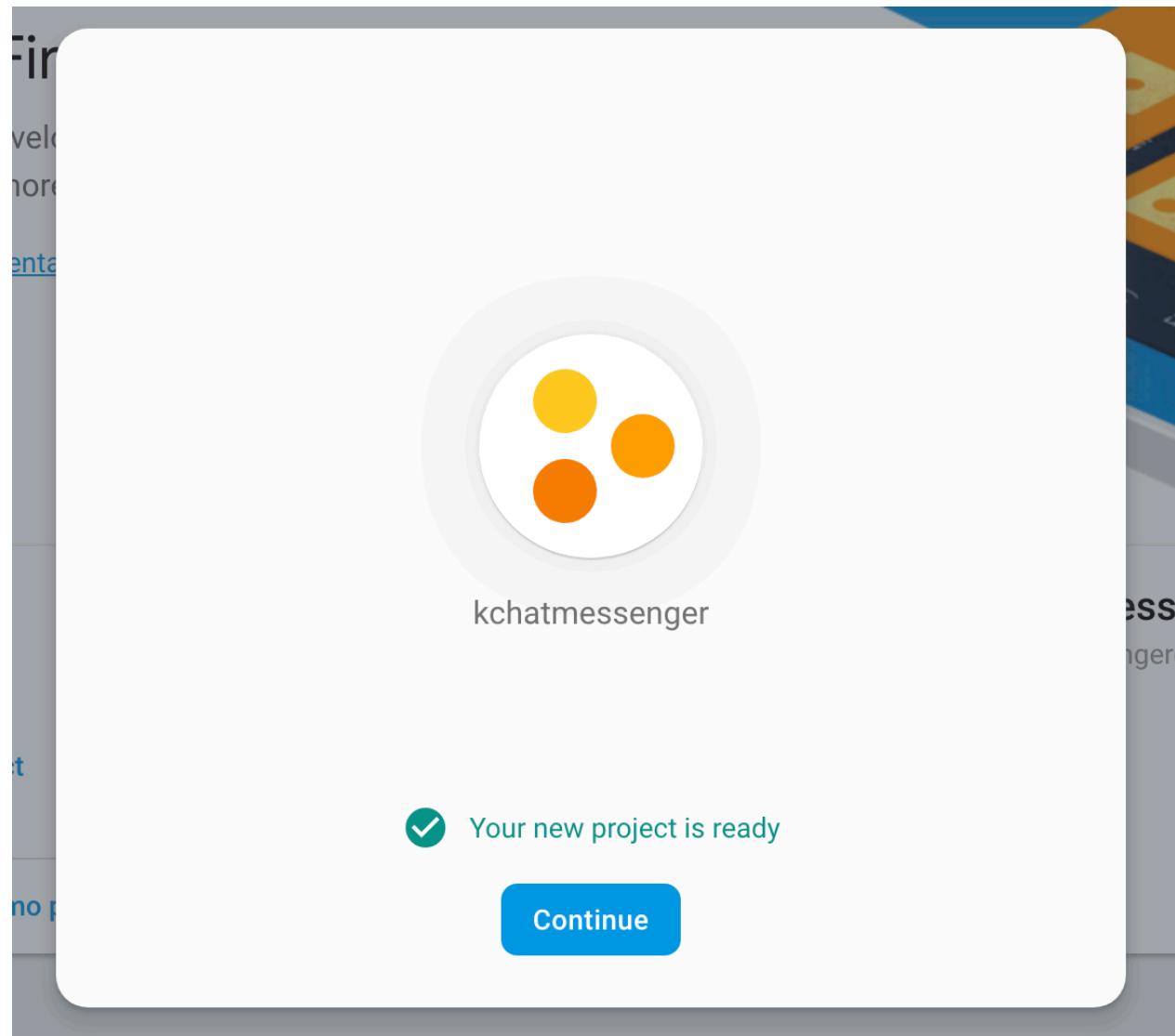


Figure 49: k030

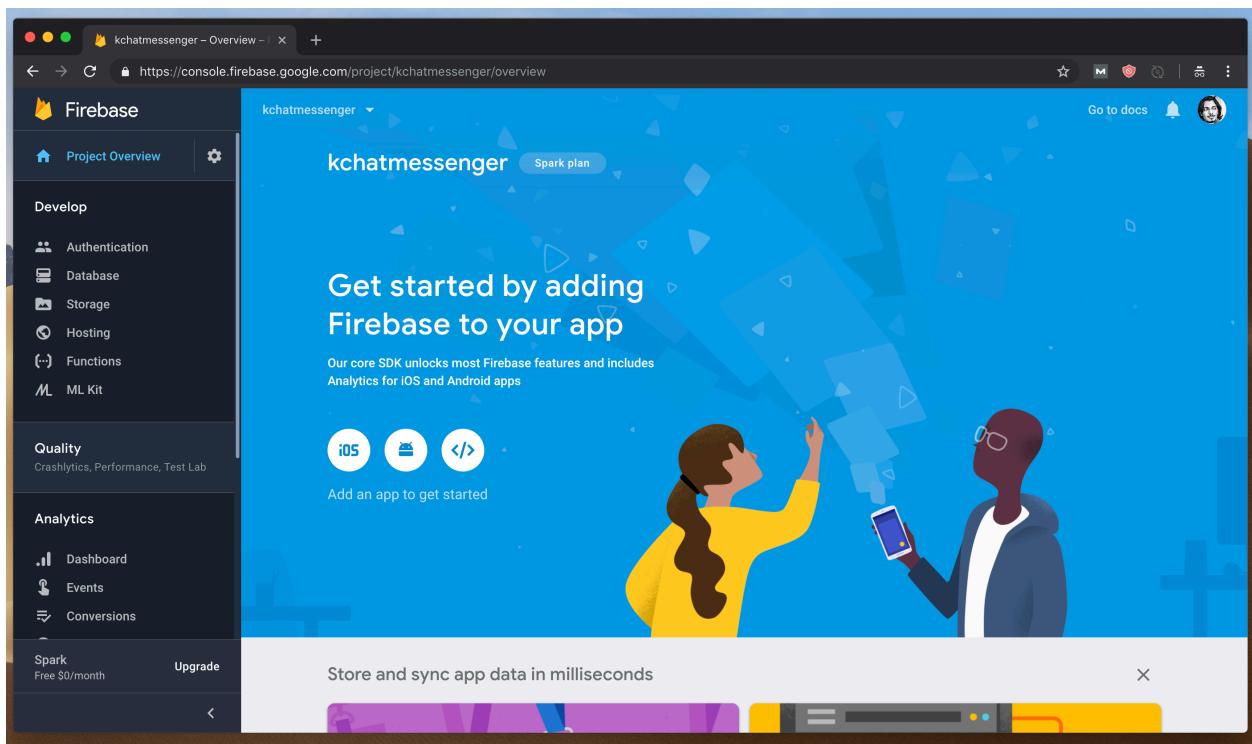


Figure 50: k031

Esse é o painel do Firebase:

Primeiramente vamos definir o tipo de plataforma que iremos trabalhar. Existe 3 opções por hora:

- iOS
- Android
- Web

Claro que vamos clicar em Android.

Na tela seguinte, vamos preenchendo os dados e sincronizando com o Android Studio. Os dois trabalham super-bem juntos e é bem intuitivo a configuração.

11.7.1 Registrando o Aplicativo no Firebase

Para registrar o aplicativo, na verdade o identificador único do aplicativo devemos abrir nosso **AndroidManifest.xml** e copiar o **package**. `co.tiagoaguiar.kchatmessenger`.

Depois, adicionar um apelido ao aplicativo. Pode ser o nome do aplicativo mesmo.

O próximo campo podemos deixar em branco que é um recurso para efetuarmos login via Google e outras coisas um pouco mais avançadas - não se preocupe com isso agora.

Clique em **Register app**. Para começar a sincronização.

O próximo passo é baixar um arquivo que o Firebase gerou e adicioná-lo ao projeto no Android Studio.

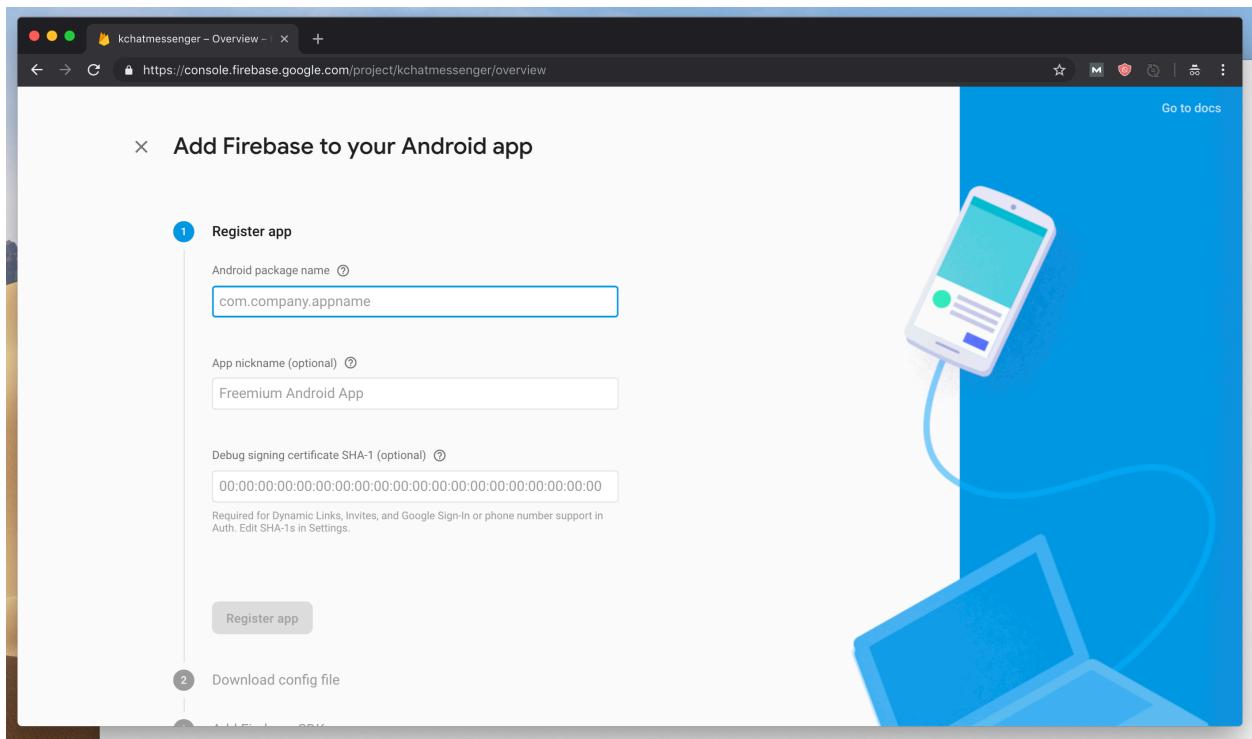


Figure 51: k032

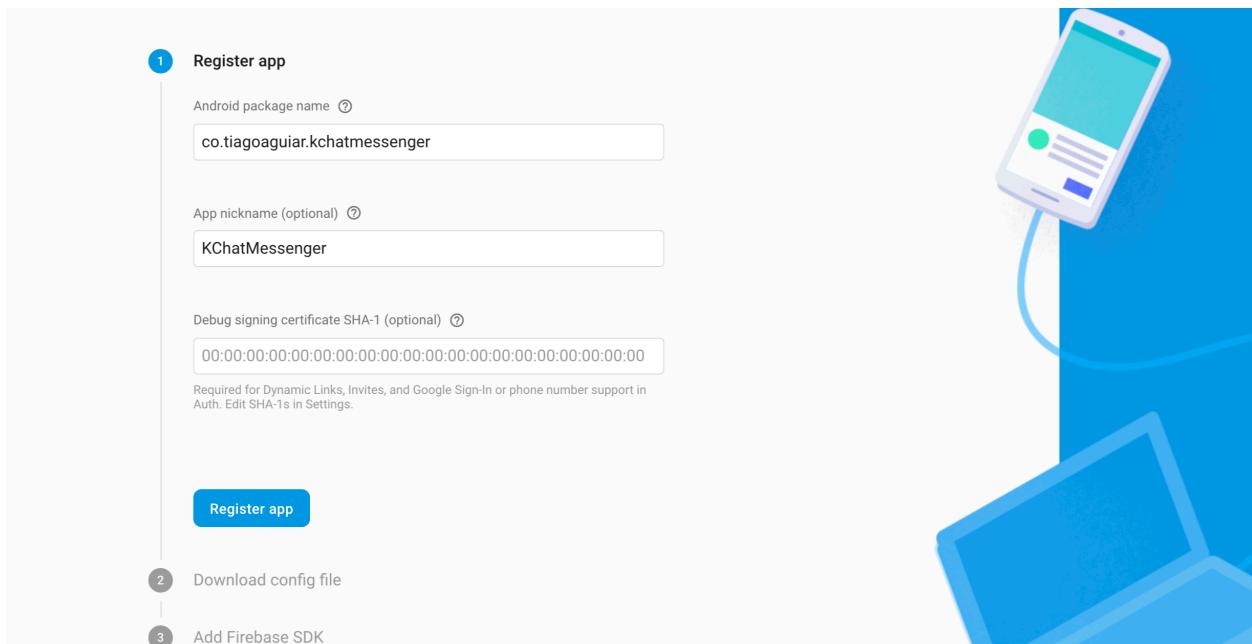


Figure 52: k033

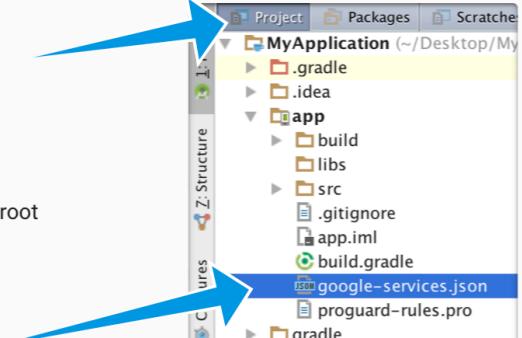
1 Register app
Android package name: co.tiagoaguiar.kchatmessenger, App nickname: KChatMessenger

2 Download config file [Download google-services.json](#) Instructions for Android Studio below | [Unity](#) [C++](#)

Switch to the Project view in Android Studio to see your project root directory.

Move the google-services.json file you just downloaded into your Android app module root directory.





Previous [Next](#)

Figure 53: k034

3 Add Firebase SDK

Instructions for Gradle | [Unity](#) [C++](#)

The Google services plugin for [Gradle](#) loads the google-services.json file you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
    dependencies {
        // Add this line
        classpath 'com.google.gms:google-services:4.0.1'
    }
}
```

App-level build.gradle (<project>/<app-module>/build.gradle):

```
dependencies {
    // Add this line
    implementation 'com.google.firebaseio:firebase-core:16.0.1'
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

Includes Analytics by default [?](#)

Finally, press "Sync now" in the bar that appears in the IDE:



Previous [Next](#)

Figure 54: k035

No Android Studio, troque a visão para **Project** e adicione esse arquivo **google-services.json** no **diretório app**. Clique em **Next** após importar o arquivo.

Agora, com o arquivo no lugar certo, devemos adicionar o SDK do Firebase. Da mesma forma que o Android possui o seu kit de desenvolvimento de software (SDK), o Firebase possui o seu.

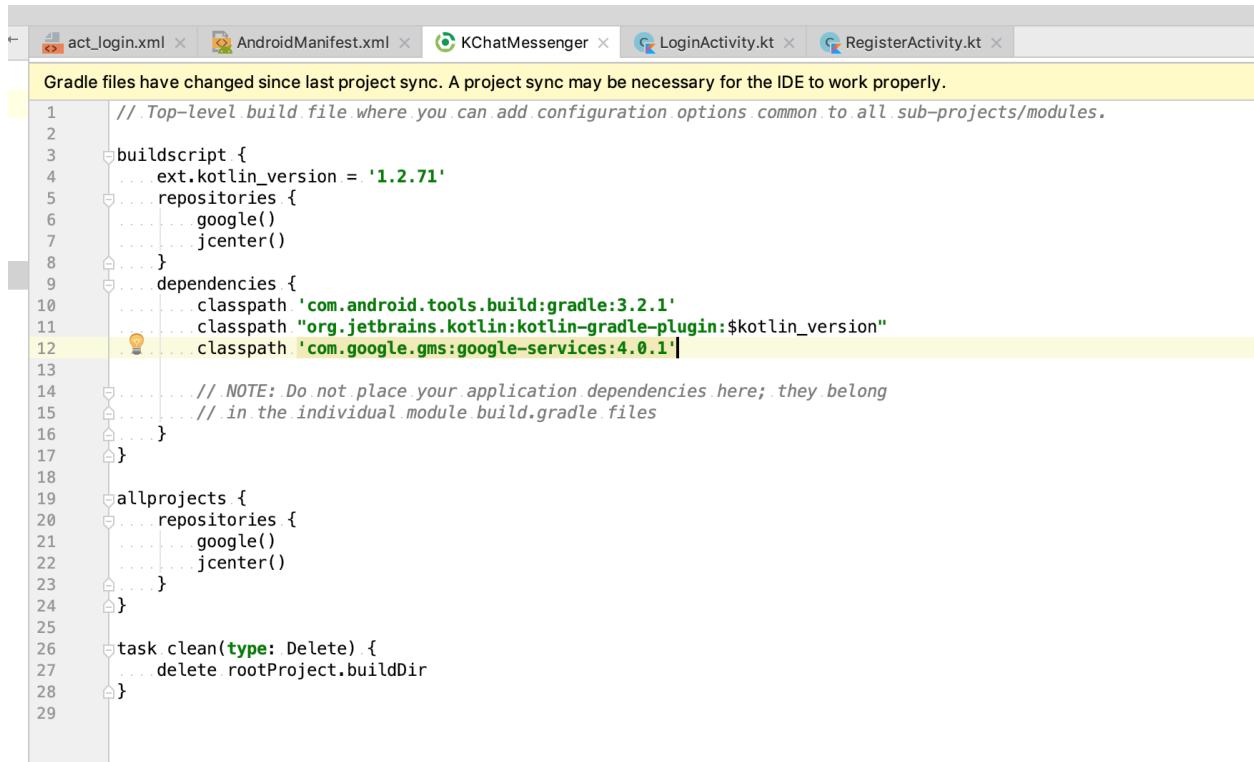
Só que ao invés de baixar vários arquivos, apenas com algumas linhas de configuração em nosso arquivo de construção (**build.gradle**) podemos fazer esta tarefa.

Adicione a linha `classpath 'com.google.gms:google-services:4.0.1'` no arquivo **KChatMessenger/build.gradle**. Dentro do bloco **dependencies**.

Ao clicar no texto, lá no Firebase, automaticamente será copiada essa linha. Agora só usar o **Ctrl/Cmd + V**

Clique em **Sync Now** para começar a sincronizar as dependências de códigos Google.

Da mesma forma, adicione a dependência ao módulo **app**. Esses últimos ficam no



```

1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2
3 buildscript {
4     ext.kotlin_version = '1.2.71'
5     repositories {
6         google()
7         jcenter()
8     }
9     dependencies {
10        classpath 'com.android.tools.build:gradle:3.2.1'
11        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
12        classpath 'com.google.gms:google-services:4.0.1'
13
14        // NOTE: Do not place your application dependencies here; they belong
15        // in the individual module build.gradle files
16    }
17 }
18
19 allprojects {
20     repositories {
21         google()
22         jcenter()
23     }
24 }
25
26 task clean(type: Delete) {
27     delete rootProject.buildDir
28 }
29

```

Figure 55: k036

KChatMessenger/app/build.gradle.

```

dependencies {
    // outras definições

    implementation 'com.google.firebaseio:firebase-core:16.0.1'
}

apply plugin: 'com.google.gms.google-services'

```

No último passo, você deverá executar o aplicativo para que as alterações sejam efetivadas e o Firebase comece a receber sinal do seu aplicativo.

Pronto! Seu aplicativo está configurado com o Firebase.

Agora vamos criar os 3 recursos que usaremos no Chat:

1. Autenticação
2. Banco de dados
3. Espaço em disco

Cada componente deste deve ser configurado. Para o *Authentication*, clique no menu ao lado e então em **Set up sign-in method**.

Ative a opção de autenticação por **Email**.

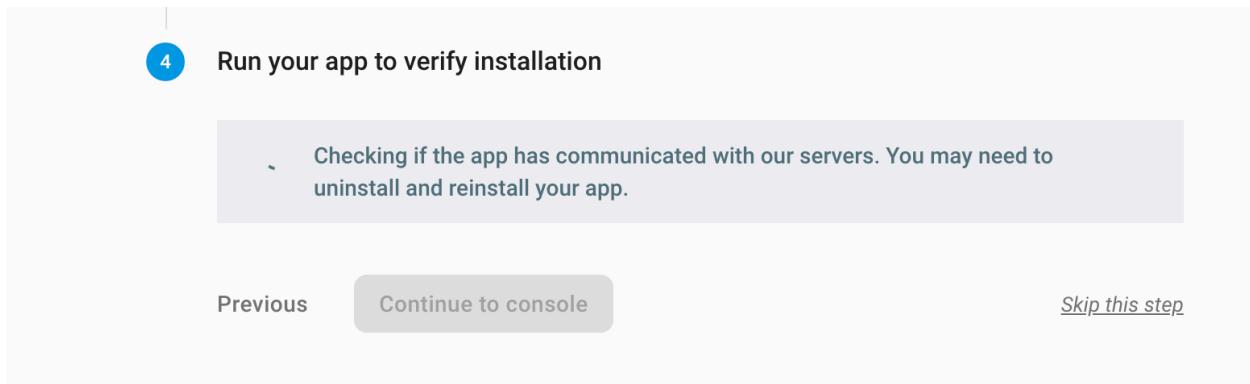


Figure 56: k037

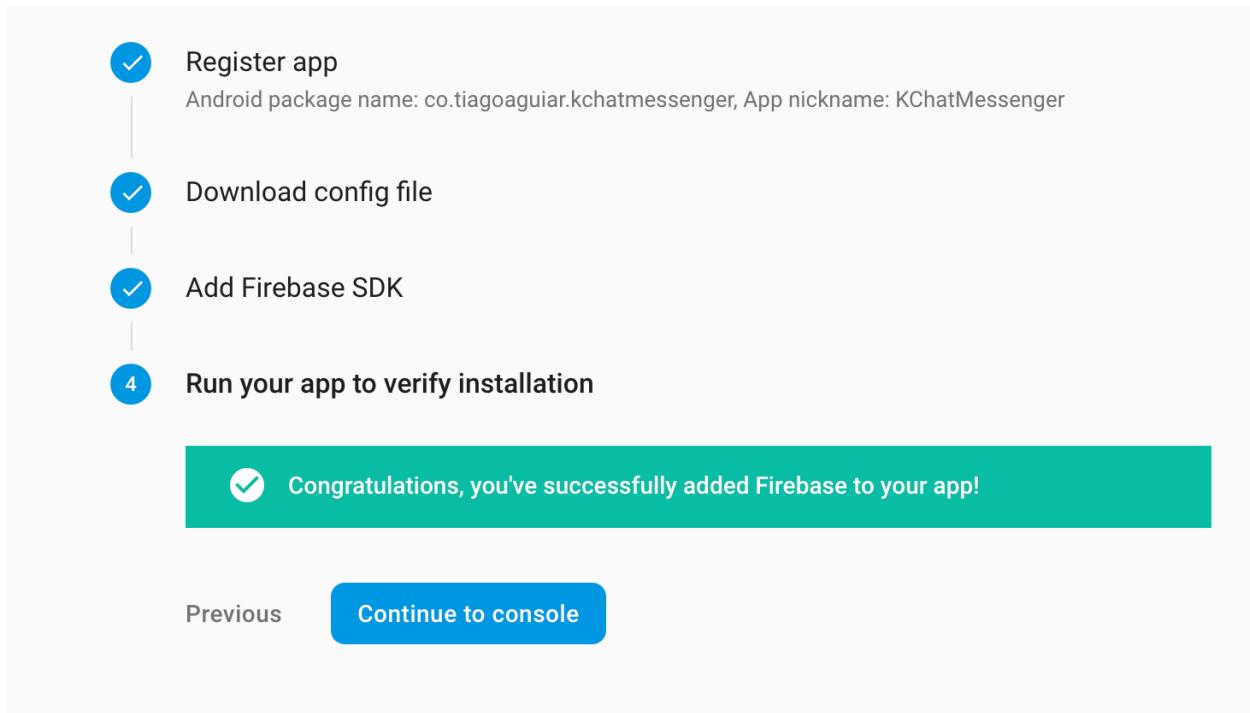


Figure 57: k038

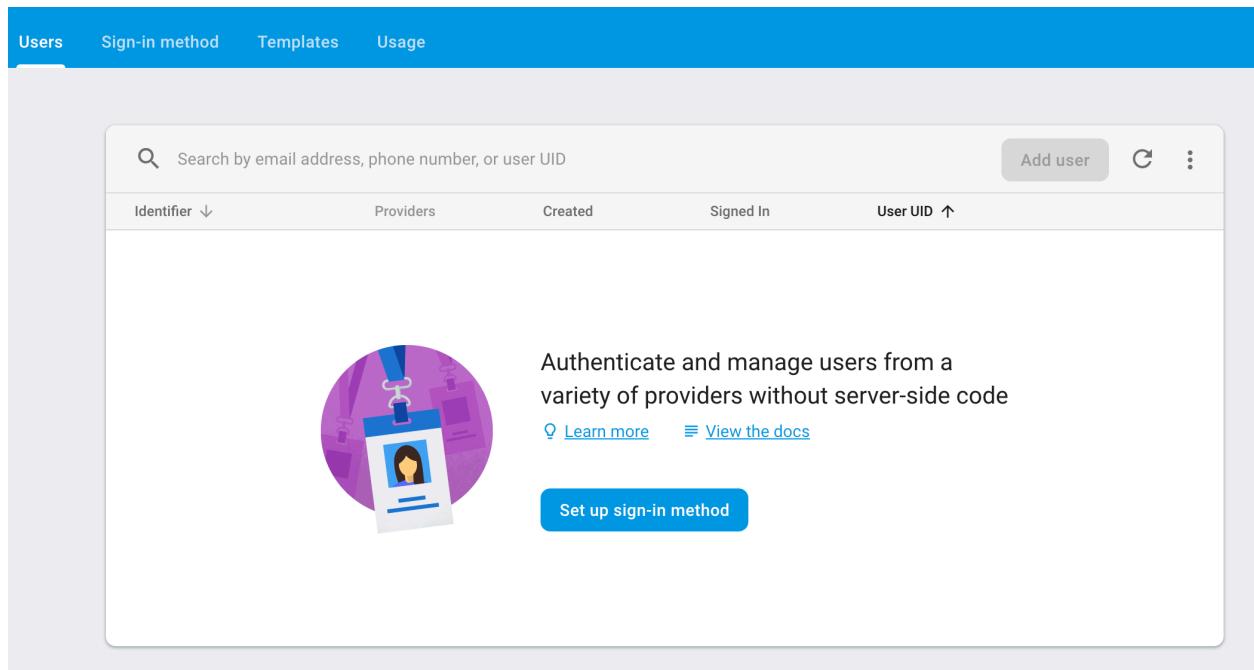


Figure 58: k039

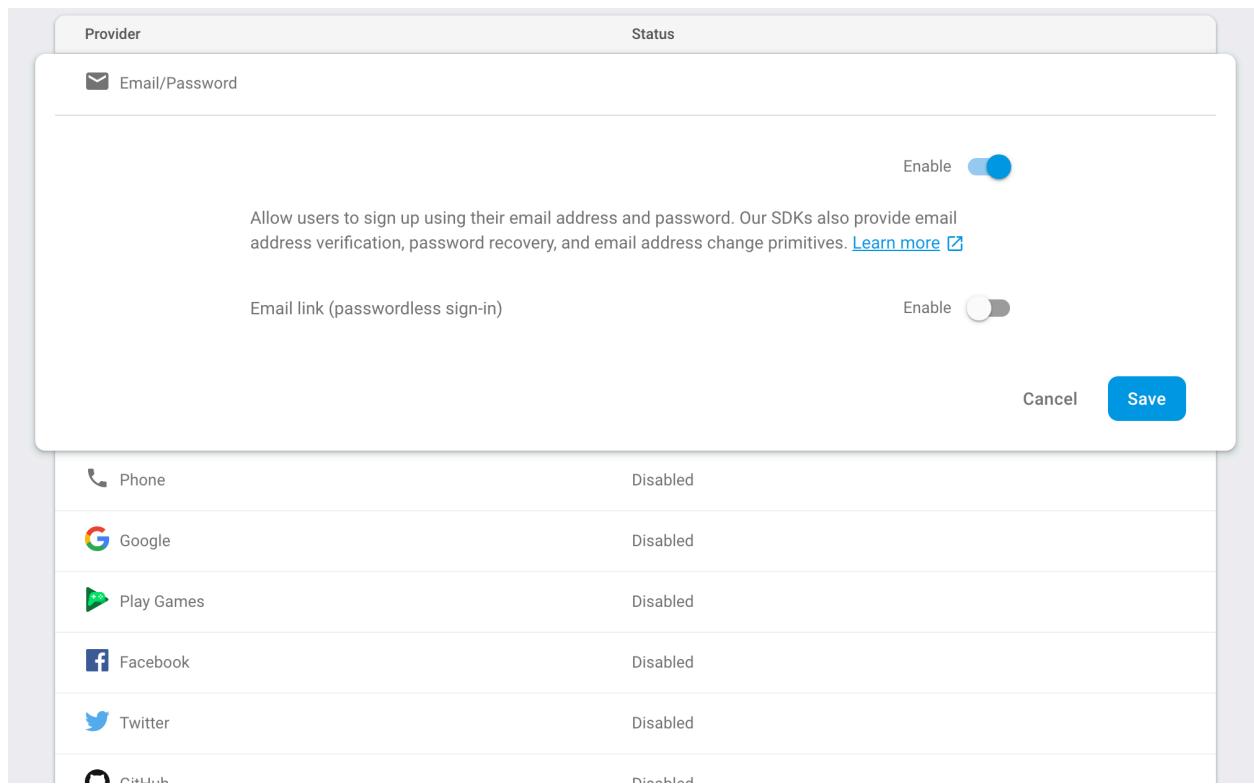


Figure 59: k040

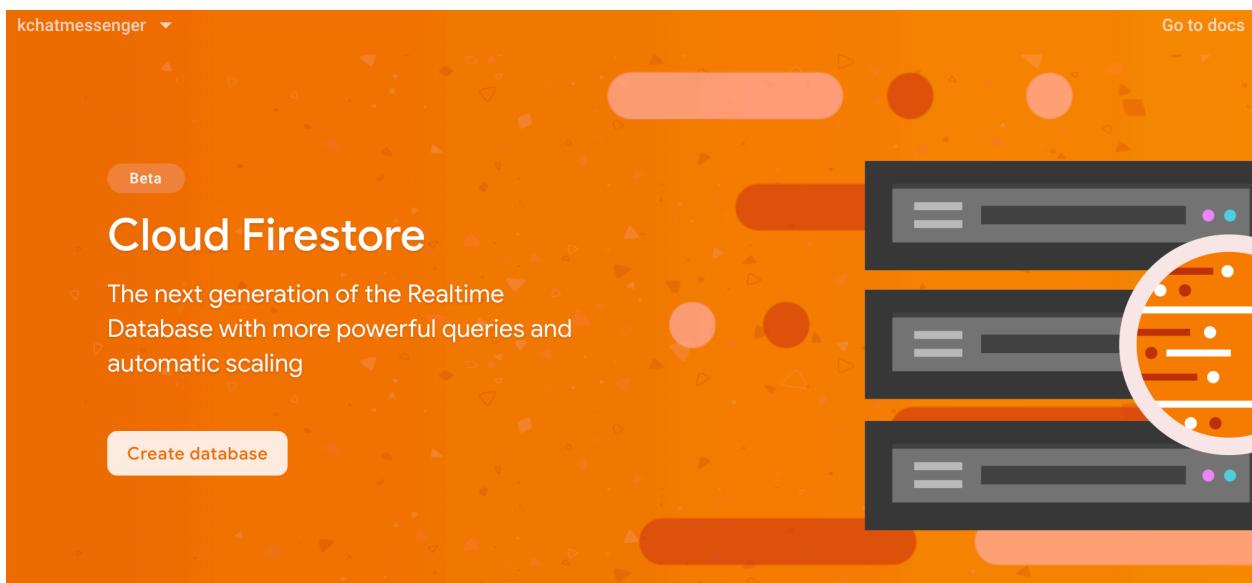


Figure 60: k041

Como nosso aplicativo fará a autenticação por email, escolhemos essa opção. Mas há vários aplicativos onde você pode fazer o login pelo Facebook ou pelo Google por exemplo.

Para o banco de dados vamos utilizar o novo banco chamado de **Cloud Firestore**. Clique em *Create database* para começar a criar um novo banco de dados.

Escolha a opção de **test mode**.

E por fim, vamos criar um **storage** para armazenar nossos arquivos de mídias. Para o nosso projeto, seria as imagens dos usuários, os arquivos .jpg ou .png.

O processo de criação desta estrutura é igual as outras (com poucos cliques tudo está pronto)!

11.7.2 #1 - Firebase Authentication

Antes de criar qualquer registro na base de dados, seja conversa ou usuário, precisamos criar um usuário autenticado. Agora que temos a biblioteca do Firebase pronta para uso, vamos adicionar ao evento de click um método responsável por criar esse usuário.

Abra seu Android Studio e adicione a dependência do **Firebase Auth**. Esse passo é necessário porque a implementação que temos atualmente é somente do básico do Firebase que é o *core firebase* onde inclui ferramentas de Crash e Analytics.

O componente de autenticação fica em outro pacote:

```
implementation 'com.google.firebaseio:firebase-auth:16.0.1'
```

Então, acesse seu **app/build.gradle** e adicione essa implementação e sincronize o projeto.

```
dependencies {
    // outras definições

    // firebase
```

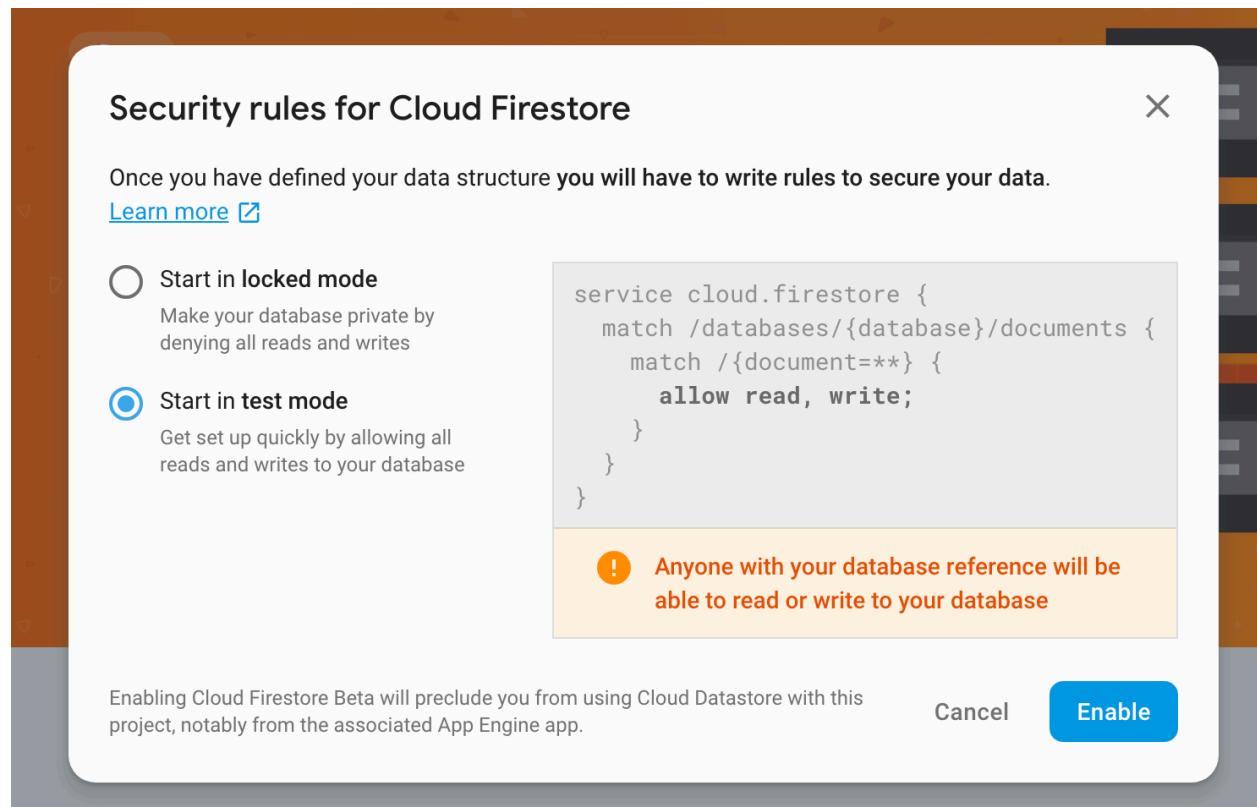


Figure 61: k042

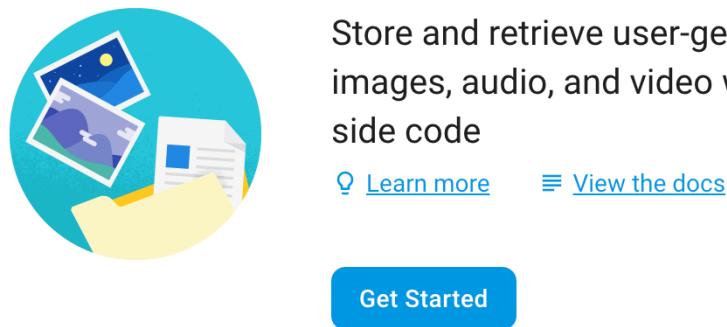


Figure 62: k043

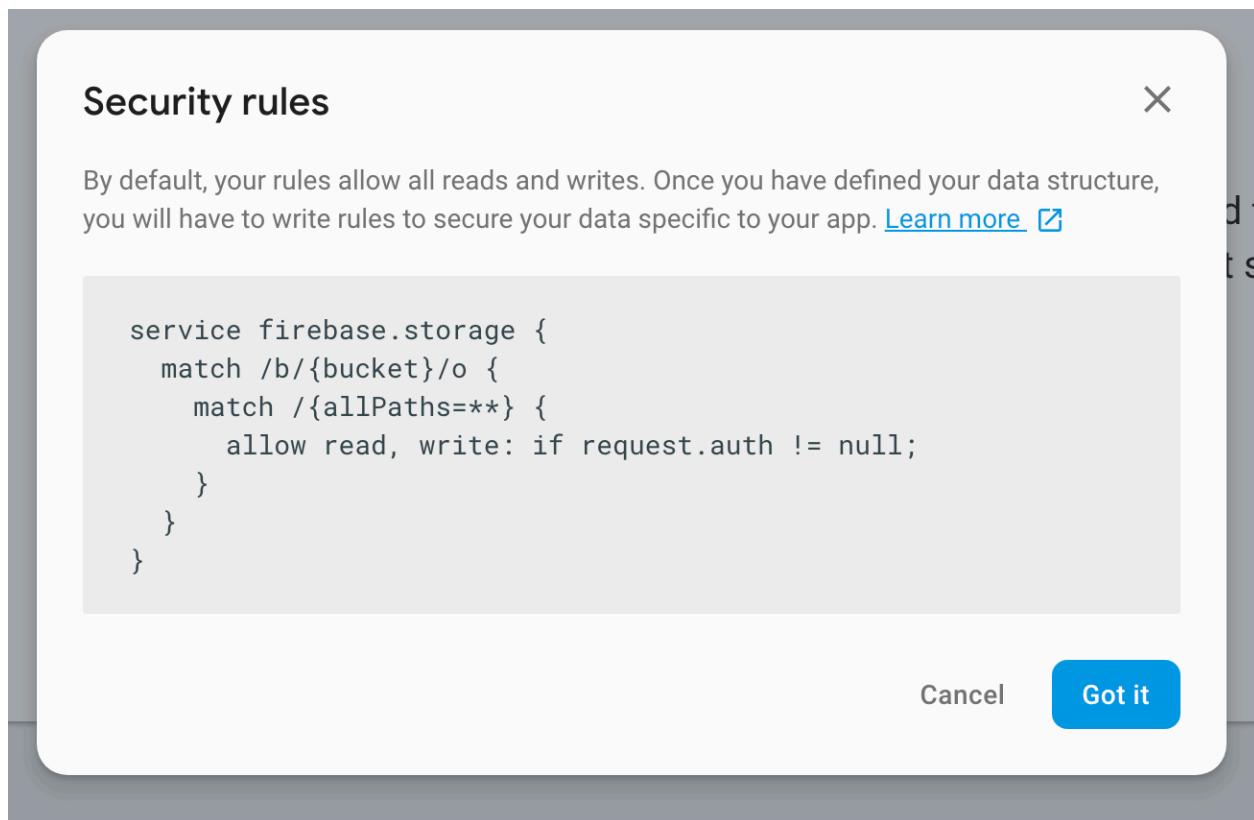


Figure 63: k044

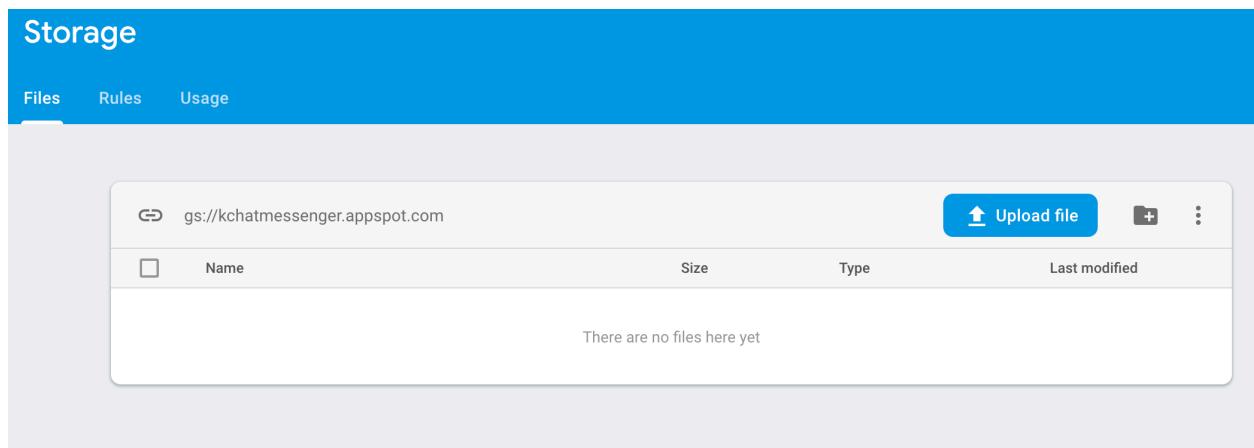


Figure 64: k045

```

implementation 'com.google.firebaseio:firebase-core:16.0.1'
implementation 'com.google.firebaseio:firebase-auth:16.0.1'
}

apply plugin: 'com.google.gms.google-services'

```

Agora sim, abra a classe **RegisterActivity** para criar um novo método de criação de usuário autenticado.

Para criar um método em Kotlin basta usar a sintaxe:

```

private fun createUser() {
}

```

- *private* para que o método seja visível somente no escopo da classe atual (**RegisterActivity**)
- *fun* para indicar um método/função
- *createUser()* é o nome do método sem parâmetros
- E por fim o bloco `{}` que é onde fica o corpo do método

Agora troque o evento de `onClick` para executar esse novo método. Dentro dele, vamos definir algumas chamadas para o Firebase.

Abaixo está o código completo. Vamos entender esse novo bloco de código `createUser()`.

```

package co.tiagoaguiar.kchatmessenger

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import com.google.firebase.auth.FirebaseAuth
import kotlinx.android.synthetic.main.act_register.*

class RegisterActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_register)

        btn_register.setOnClickListener {
            createUser()
        }
    }

    private fun createUser() {
        val email = edit_email.text.toString() // 1
        val password = edit_password.text.toString() // 1
    }
}

```

```

    if (email.isEmpty() || password.isEmpty()) { // 2
        Toast.makeText(this,
                      "email e senha devem ser informados",
                      Toast.LENGTH_LONG).show()
        return
    }

    FirebaseAuth.getInstance()
        .createUserWithEmailAndPassword(email, password) // 2
        .addOnCompleteListener { // 3
            if (it.isSuccessful) { // 4
                Log.i("Teste", "UserID é ${it.result.user.uid}") // 5
            }
        }.addOnFailureListener { // 6
            Log.e("Teste", it.message, it) // 7
        }
    }
}

```

Adicionei alguns comentários enumerados para explicar o que cada linha faz.

1. Esta linha guarda em uma variável o que o usuário digitou no campo de texto do nosso layout: neste caso o email e senha
2. Garante que somente email e senha digitados serão enviados ao Firebase.
3. Esta linha obtém uma instância do objeto FirebaseAuth através do método `getInstance()`. Com o objeto “vivo” em mãos, executamos o seu método chamado `createUserWithEmailAndPassword` que espera 2 parâmetros.
 1. Email
 2. Senha
4. Da mesma forma que criamos listeners para eventos de click, vamos adicionar listeners para eventos de falha e sucesso. O Método `addOnCompleteListener` espera um objeto anônimo que irá executar **depois** que o Firebase responder de volta para o aplicativo se o usuário foi criado ou não. Isso é feito através do bloco {}.
5. Nesta linha, com o objeto atual devolvido pelo Firebase (que representamos como `it`) podemos verificar se a ação que pedimos (criar usuário) foi realizada com sucesso. Se sim, vamos imprimir no Log o Identificador do usuário `uid`.
6. Imprími o Identificador do usuário `uid`.
7. Semelhante ao evento de `complete` este evento escuta a resposta do firebase caso ocorra algum erro como por exemplo, tentar cadastrar um email que não é válido (asdaksdjha.com)
8. Se ocorrer algum erro, imprimir no console para podermos corrigir ou, eventualmente, exibir ao usuário o por quê aconteceu aquele erro.

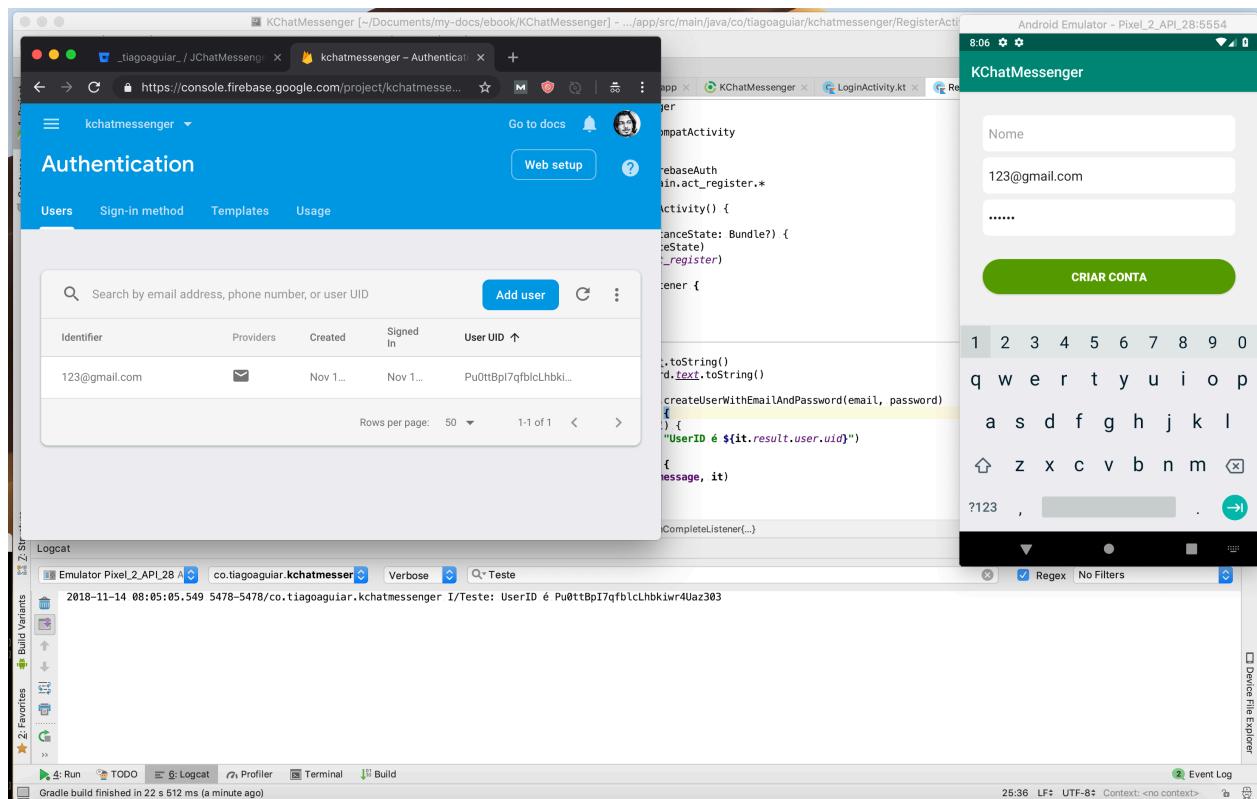


Figure 65: k046

Dando tudo certo você verá um novo usuário no painel do **FirebaseAuth**.

O nosso aplicativo também possui uma foto de perfil, mas por hora vamos apenas focar na autenticação ok!? Logo iremos adicionar imagens e trabalhar com o **Firebase Cloud Firestore** e **Storage**.

Com o usuário criado vamos realizar um Login deste usuário - lembrando que os dados de acesso são:

- Email: 123@gmail.com
- Senha: 123456

Para efetuar um Login, o processo é praticamente o mesmo de criar uma conta. Só que desta vez, vamos usar o método `signInWithEmailAndPassword`.

Tente desenvolver o método de Login por conta própria. Logo abaixo se encontra todo o código que fiz para realizar o login.

```
package co.tiagoaguiar.kchatmessenger

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
```

```

import android.widget.Toast
import com.google.firebase.auth.FirebaseAuth
import kotlinx.android.synthetic.main.act_login.*

class LoginActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_login)

        btn_enter.setOnClickListener {
            signIn()
        }

        txt_account.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java)
            startActivity(intent)
        }
    }

    private fun signIn() {
        val email = edit_email.text.toString()
        val password = edit_password.text.toString()

        if (email.isEmpty() || password.isEmpty()) {
            Toast.makeText(this,
                "email e senha devem ser informados",
                Toast.LENGTH_LONG).show()
            return
        }

        FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
            .addOnCompleteListener {
                if (it.isSuccessful)
                    Log.i("Teste", it.result.user.uid)
            }
            .addOnFailureListener {
                Log.e("Teste", it.message, it)
            }
    }

}

```

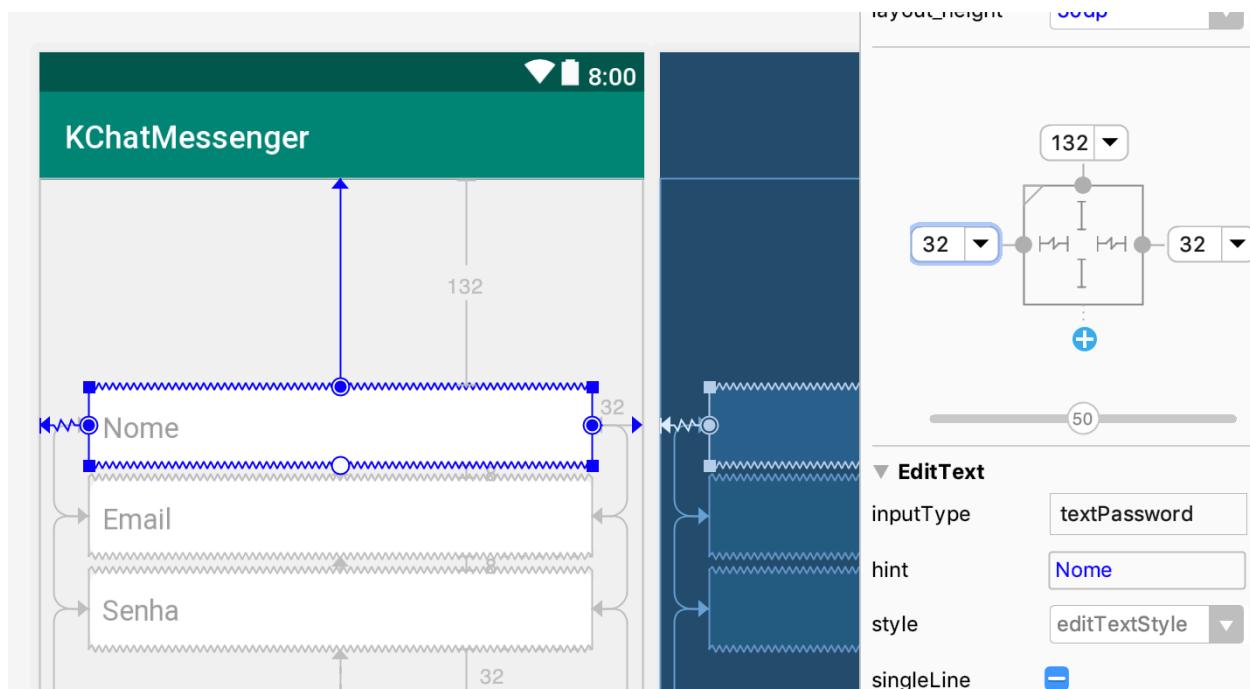


Figure 66: k047

11.7.3 #2 - Firebase Storage

Chegou a hora de trabalhar com mais um componente poderoso do Firebase - o **Armazenamento / Storage**. É aqui onde vamos gravar nossas fotos de perfis e vincular futuramente essas fotos com os usuários do Firestore.

Pois bem, vamos adicionar ao nosso layout um botão que terá eventos de click (novamente) para selecionarmos uma foto da galeria do smartphone.

Então abra o seu `act_register.xml` e adicione um botão ao topo do editor de textos usando constraints.

Desta vez, mantenha a largura e altura fixa para **150dp**.

Um truque para facilitar adicionar componentes em cima de outros é aumentar a constraint do primeiro editor para 130 por exemplo. Depois adicionar o botão, remover a constraint do editor e atribuir ao novo bottom do botão.

Crie um novo drawable com o **corners radius de 150dp** (que é a altura do botão) para garantir que ele fique circular.

Depois atribua as customizações necessárias ao botão.

<EditText

```
    android:id="@+id/edit_name"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginStart="32dp"
    android:layout_marginLeft="32dp"
    android:layout_marginEnd="32dp"
```

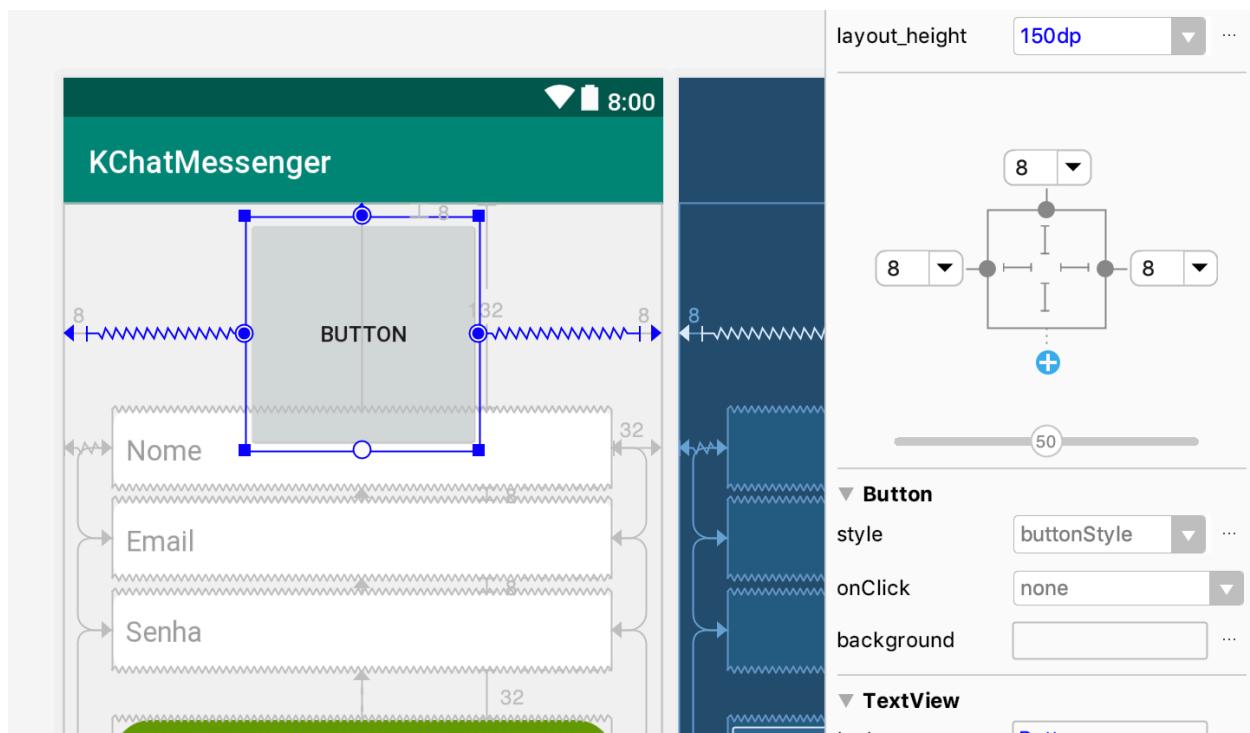


Figure 67: k048

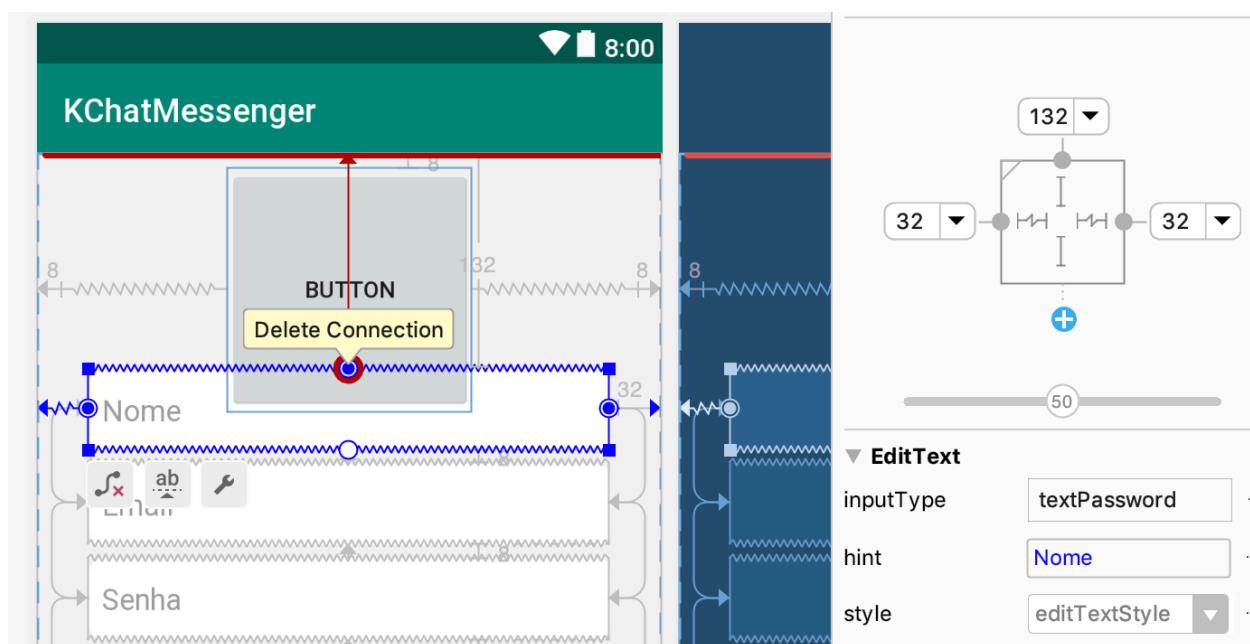


Figure 68: k049

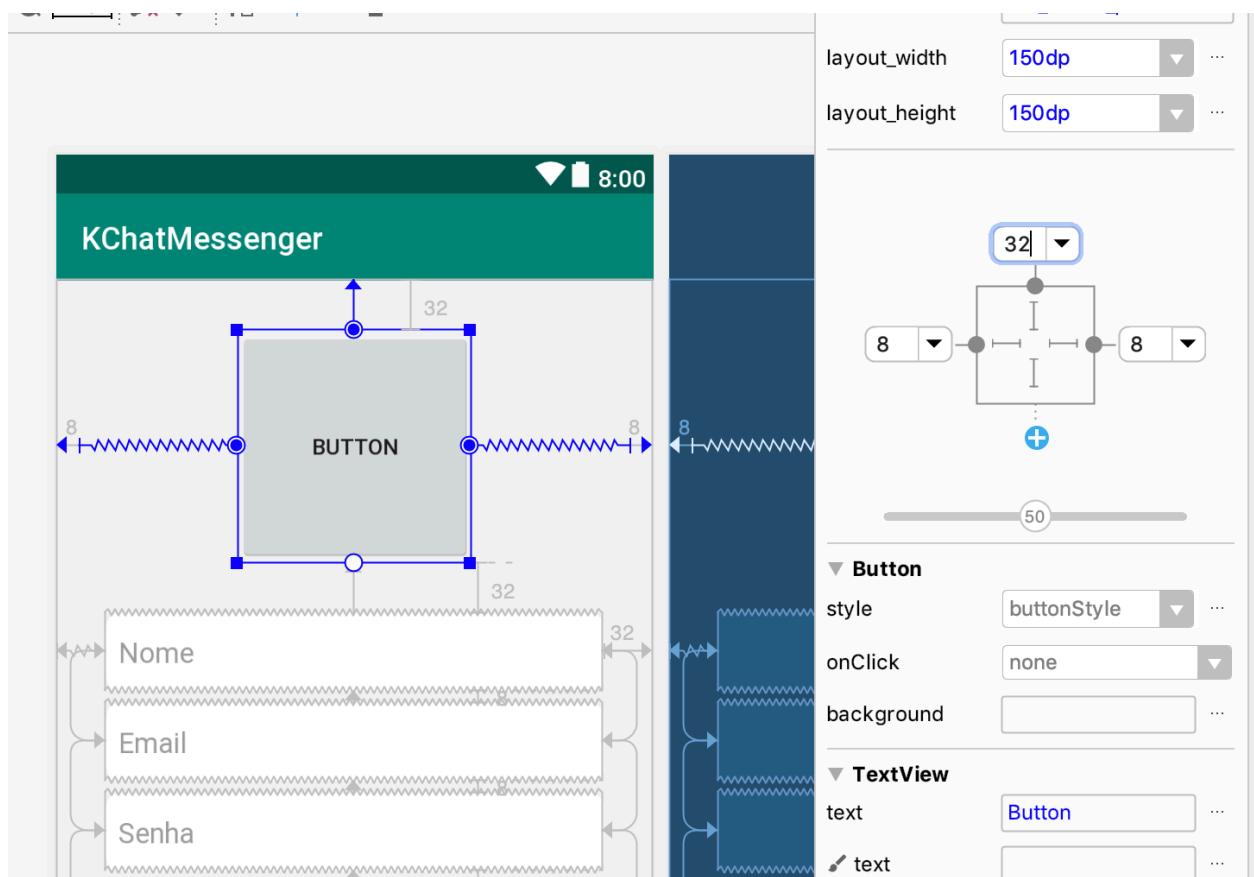


Figure 69: k050

```

        android:layout_marginRight="32dp"
        android:background="@drawable/bg_edittext_rounded"
        android:ems="10"
        android:hint="Nome"
        android:inputType="text"
        android:paddingLeft="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toBottomOf="@+id/btn_select_photo"/>

<!-- mais codigos -->

<Button
        android:text="@string/photo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:id="@+id/btn_select_photo"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:background="@drawable/bg_button_rounded_photo"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"/>

```

Corrija também o campo de inserção do nome do usuário que pode estar com `inputType="text"`.

Vamos trabalhar agora na galeria de fotos!

Abra o Chrome pelo celular a baixe alguma foto de perfil para ter na sua galeria. Neste exemplo, baixei a foto da Viúva Negra da Marvel.

De volta a classe **RegisterActivity**.. vamos criar um novo evento de click desse botão foto e abrir a galeria. O Seguinte código faz isso:

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_register)

    btn_register.setOnClickListener {
        createUser()
    }
}

```

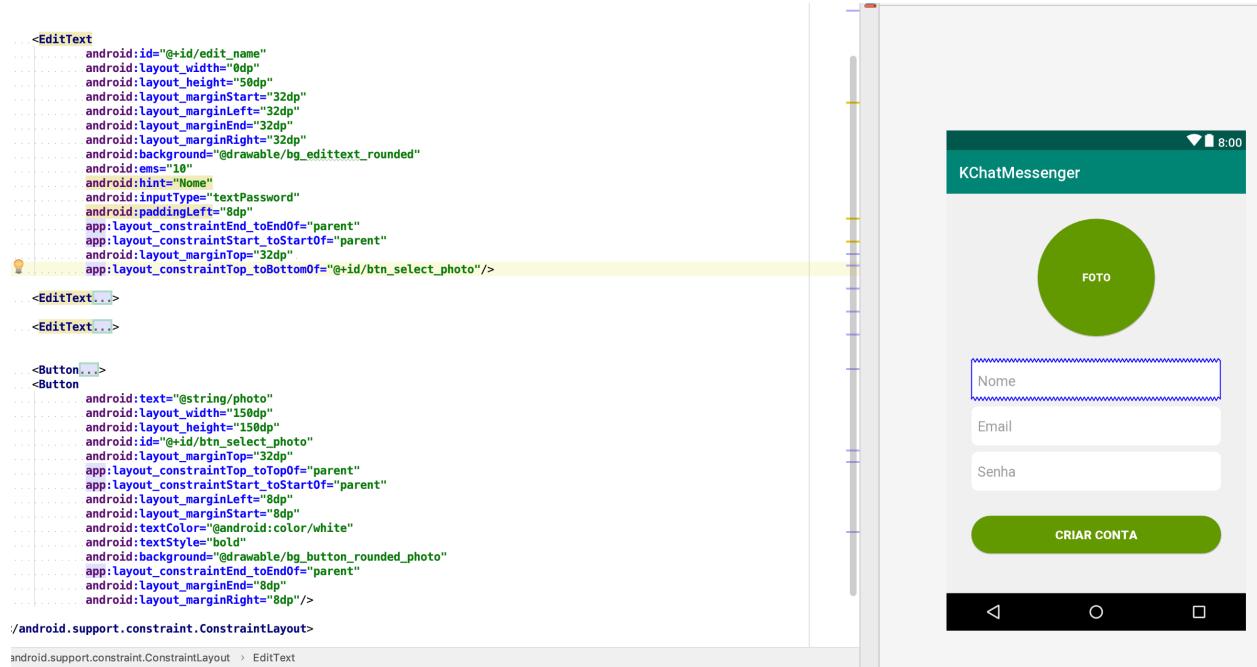


Figure 70: k051

```

        btn_select_photo.setOnClickListener {
            selectPhoto()
        }

    private fun selectPhoto() {
        val intent = Intent(Intent.ACTION_PICK) // 1
        intent.type = "image/*" // 2
        startActivityForResult(intent, 0) // 3
    }
}

```

1. Define o tipo de intenção. Desta vez não vamos abrir uma activity e sim uma ação de selecionar arquivos da galeria.

2. Define o tipo de arquivo que podemos selecionar - somente imagem.

3. Inicia um método que é disparado quando a activity volta ficar visível. Esse método é o:

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {}.

```

Vamos usar ele no próximo passo.

Sobrescreva o método da classe Activity `onActivityResult` para “escutar” os eventos de quando a galeria fechar e a Activity retornar.

Como passamos o código zero ao chamar `startActivityForResult(intent, 0)`, podemos verificar de volta se o código que o chamou está correto, veja:

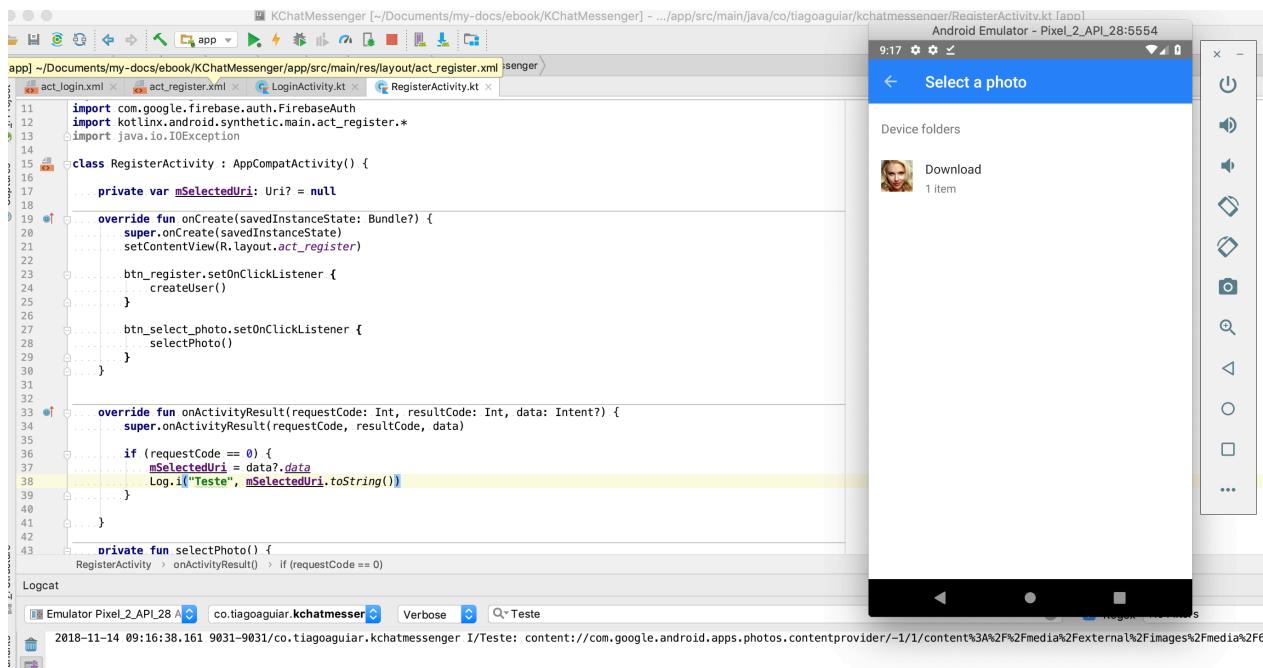


Figure 71: k052

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == 0) {
        mSelectedUri = data?.data
        Log.i("Teste", mSelectedUri.toString())
    }
}

```

Desta forma conseguimos buscar a URI de onde se encontra a foto dentro do nosso Smartphone. Agora vamos criar um novo componente - **ImageView** - para atribuir essa foto e esconder o botão.

Poderíamos usar o próprio ImageView do SDK Android, mas vamos deixar nosso aplicativo mais elegante adicionando uma imagem circular com a biblioteca

`de.hdodenhof.circleimageview.CircleImageView`.

Para adicionar essa biblioteca, acesse o **build.gradle** e adicione essa implementação:

```
implementation 'de.hdodenhof:circleimageview:2.2.0'.
```

Agora no arquivo de layout, posicione a ImageView exatamente no mesmo lugar que o Button e troque a implementação para o da biblioteca.

O novo layout de tela de cadastro ficará assim:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout

```

```

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:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@color/bgColor"
tools:context=".LoginActivity">

<EditText
    android:id="@+id/edit_name"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginStart="32dp"
    android:layout_marginLeft="32dp"
    android:layout_marginEnd="32dp"
    android:layout_marginRight="32dp"
    android:background="@drawable/bg_edittext_rounded"
    android:ems="10"
    android:hint="Nome"
    android:inputType="text"
    android:paddingLeft="8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="32dp"
    app:layout_constraintTop_toBottomOf="@+id/btn_select_photo"/>

<EditText
    android:id="@+id/edit_email"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    android:background="@drawable/bg_edittext_rounded"
    android:ems="10"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:paddingLeft="8dp"
    app:layout_constraintEnd_toEndOf="@+id/edit_name"
    app:layout_constraintStart_toStartOf="@+id/edit_name"
    app:layout_constraintTop_toBottomOf="@+id/edit_name"/>

<EditText
    android:id="@+id/edit_password"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    android:ems="10"

```

```

        android:background="@drawable/bg_edittext_rounded"
        android:hint="Senha"
        android:inputType="textPassword"
        android:paddingLeft="8dp"
        app:layout_constraintEnd_toEndOf="@+id/edit_email"
        app:layout_constraintStart_toStartOf="@+id/edit_email"
        app:layout_constraintTop_toBottomOf="@+id/edit_email"/>

<Button
        android:id="@+id/btn_register"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:background="@drawable/bg_button_rounded"
        android:text="@string/create_account"
        android:textAllCaps="true"
        android:textColor="@android:color/white"
        android:textSize="16sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="@+id/edit_password"
        app:layout_constraintStart_toStartOf="@+id/edit_password"
        app:layout_constraintTop_toBottomOf="@+id/edit_password"/>

<Button
        android:text="@string/photo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:id="@+id/btn_select_photo"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:background="@drawable/bg_button_rounded_photo"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"/>

<de.hdodenhof.circleimageview.CircleImageView
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:id="@+id/img_photo"
        app:civ_border_color="@android:color/holo_green_dark"
        app:civ_border_width="2dp"
        app:layout_constraintTop_toTopOf="@+id/btn_select_photo"

```

```

        app:layout_constraintBottom_toBottomOf="@+id/btn_select_photo"
        app:layout_constraintEnd_toEndOf="@+id/btn_select_photo"
        app:layout_constraintStart_toStartOf="@+id/btn_select_photo"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

E o código-fonte deverá transformar a URI em um objeto Bitmap para ser adicionado nesse componente `CircleImageView`.

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == 0) {
        mSelectedUri = data?.data
        Log.i("Teste", mSelectedUri.toString())

        val bitmap = MediaStore.Images.Media.getBitmap(
            contentResolver,
            mSelectedUri)
        img_photo.setImageBitmap(bitmap)
        btn_select_photo.alpha = 0f
    }
}

```

Por fim, escondemos (em tese) o botão ao diminuir suas cores a transparência com `alpha = 0f`.

Seu aplicativo deverá parecer com este:

Agora vamos realizar o upload dessa imagem para o **Firebase Storage**!

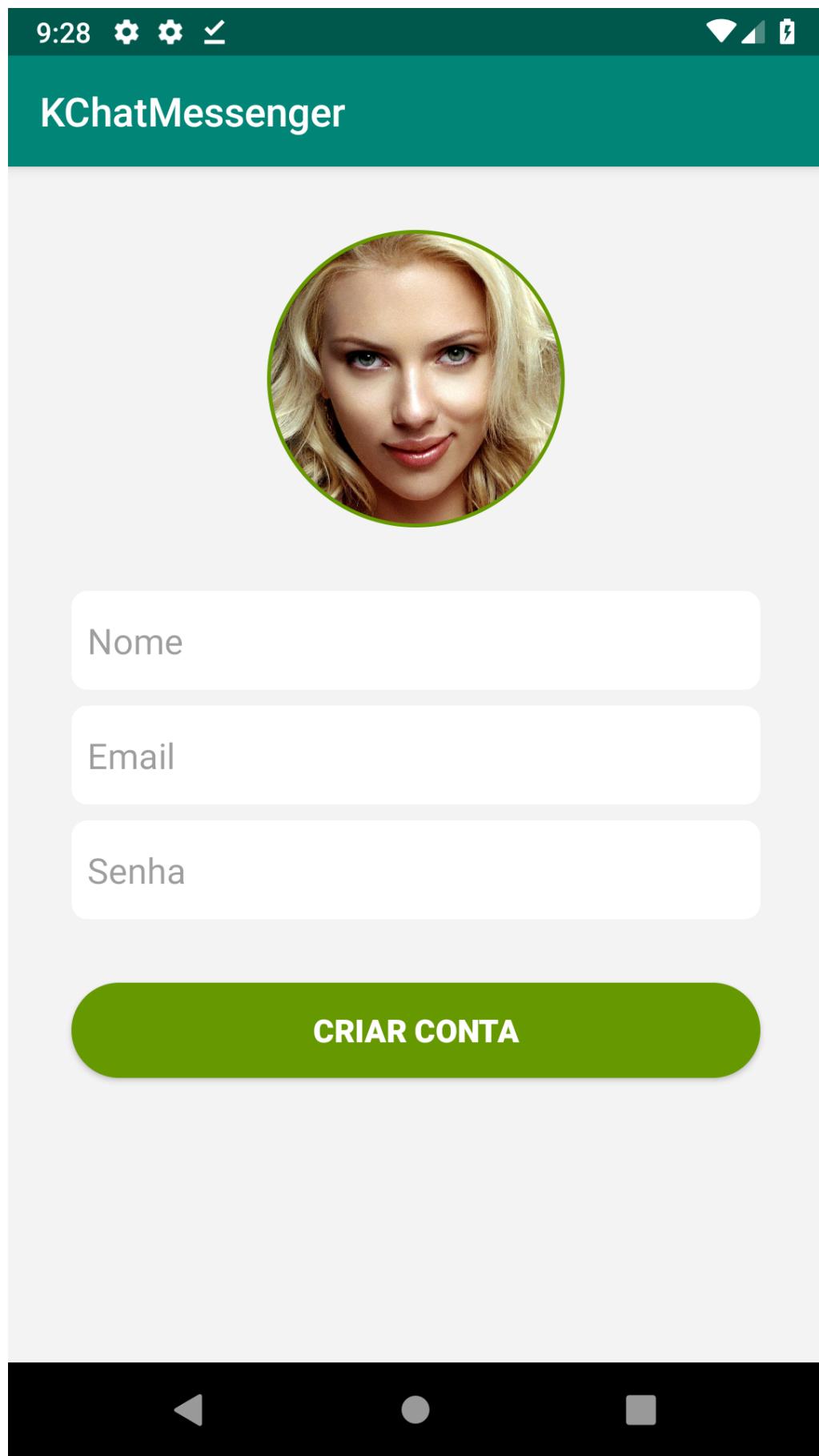
```

private fun createUser() {
    val email = edit_email.text.toString()
    val password = edit_password.text.toString()

    if (email.isEmpty() || password.isEmpty()) {
        Toast.makeText(this,
            "email e senha devem ser informados",
            Toast.LENGTH_LONG).show()
        return
    }

    FirebaseAuth.getInstance()
        .createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener {
            if (it.isSuccessful) {

```



```

        Log.i("Teste", "UserID é ${it.result.user.uid}")

        saveUserInFirebase() // 1
    }
}

.addOnFailureListener {
    Log.e("Teste", it.message, it)
}
}

private fun saveUserInFirebase() {
    val filename = UUID.randomUUID().toString() // 2
    val ref = FirebaseStorage.getInstance()
        .getReference("/images/${filename}") // 3

    mSelectedUri?.let { // 4
        ref.putFile(it) // 5
        .addOnSuccessListener { // 6
            ref.downloadUrl.addOnSuccessListener { // 7
                Log.i("Teste", it.toString())
            }
        }
    }
}
}

```

Com a referência da URI `mSelectedUri` podemos enviar esse arquivo para o Firebase. O código acima funciona da seguinte forma:

1. Adicionamos um evento de salvar a foto após o usuário ser autenticado
2. Criamos um nome de arquivo único através dos método `UUID.randomUUID().toString()`
3. Criamos um nó (diretório/caminho) dentro do banco de storage onde será salva na pasta `/images/` o nome do arquivo
4. Garantimos através do `let` que o bloco seguinte somente será executado SE a variável `mSelectedUri` realmente tiver uma referência de URI, ou seja, ela não estará nula.
5. Colocamos o objeto atual `it` que é nossa URI na referência para que ele seja feito o upload. É aqui que acontece o upload para o firebase. O Android irá verificar onde se encontra o arquivo no smartphone pela URI e irá subir o arquivo de imagem ao firebase.
6. Declaramos o evento de Listener de sucesso para conseguirmos buscar a foto lá no Storage do Firebase.
7. Executamos o download da imagem (URL) e imprimimos no console a URL oficial já fornecida pelo Firebase.

Acesse a url que é mostrada no console e você verá a foto que acabamos de subir no seu navegador.

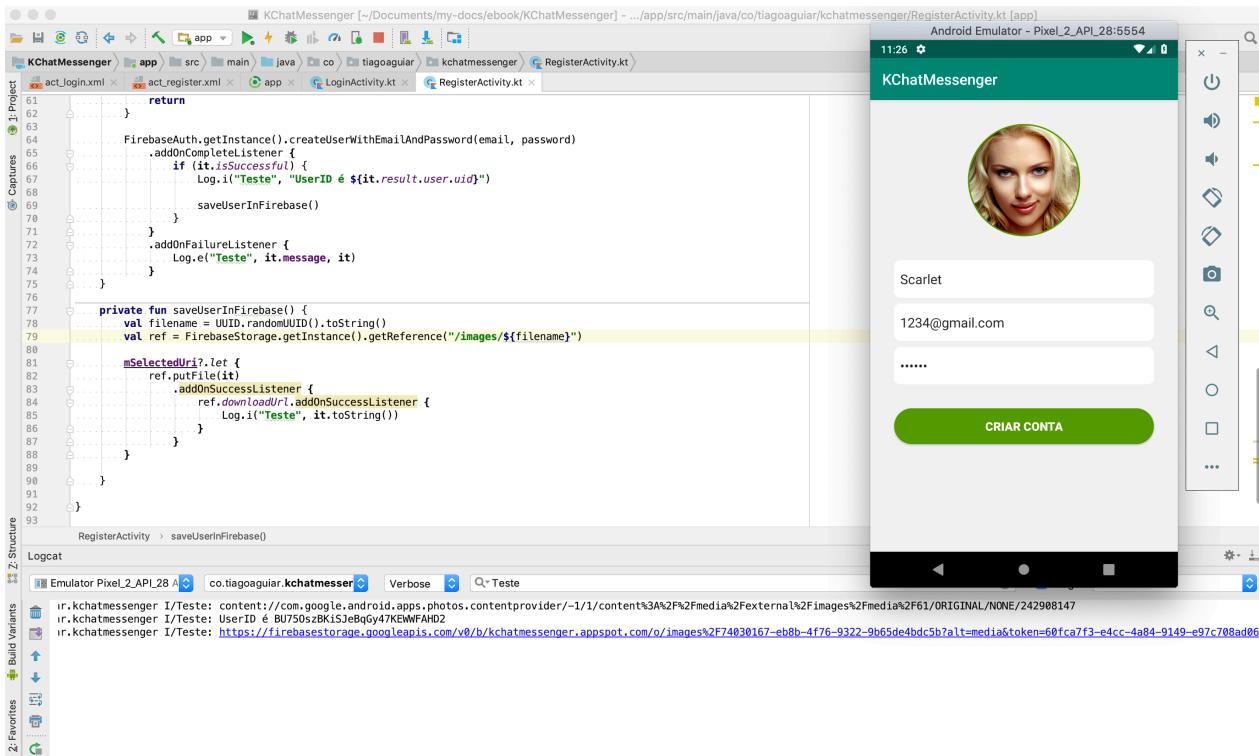


Figure 73: k054

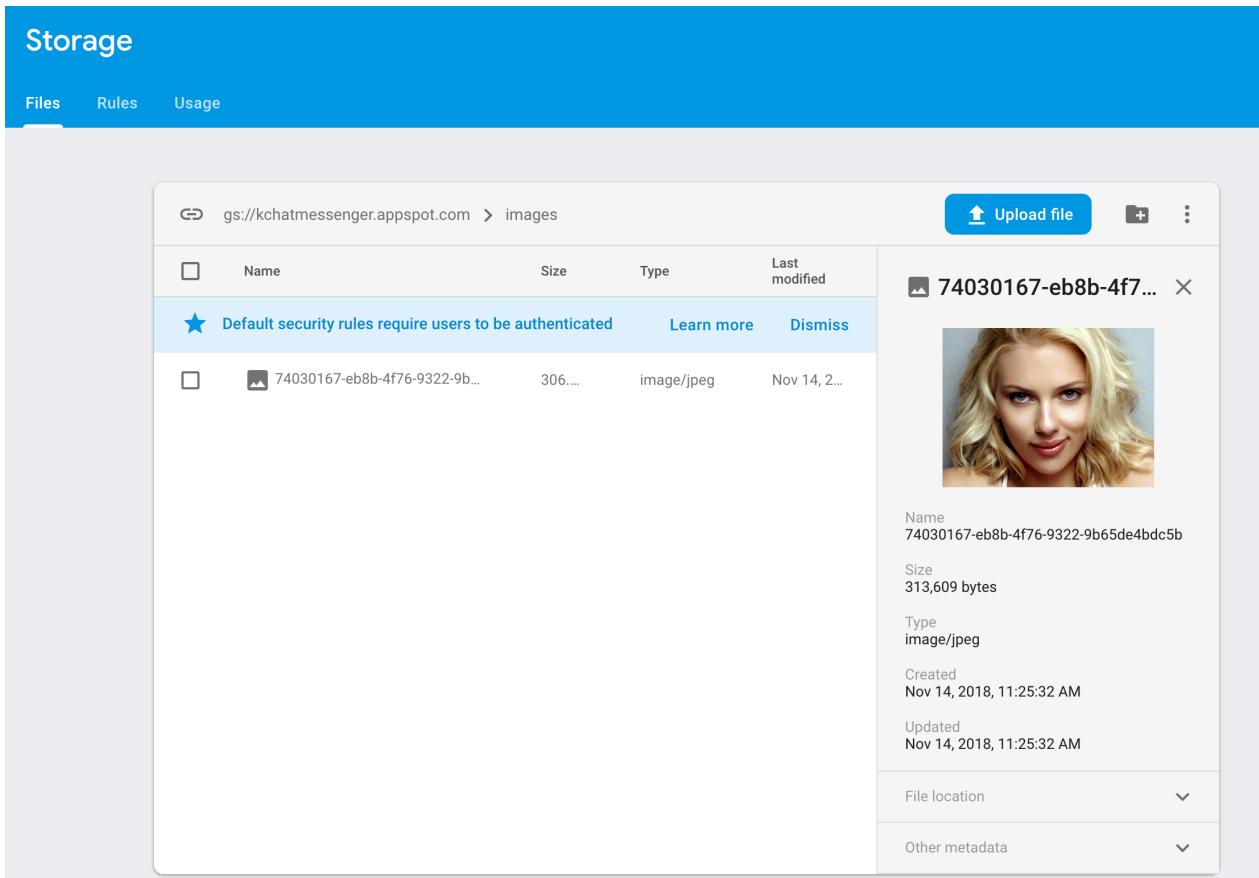


Figure 74: k055

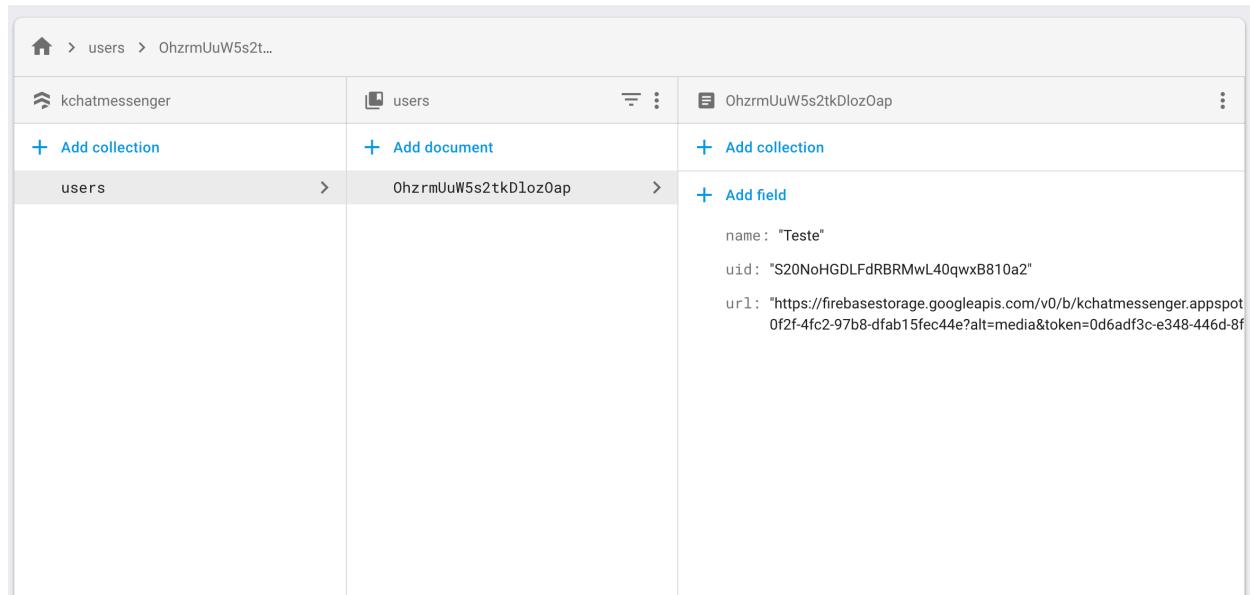


Figure 75: k056

Já no painel do Firebase você verá a foto do perfil do usuário:

Agora precisamos atrelar essa foto, o UID da Autenticação ao novo que ficará salvo no **Firebase Cloud Firestore**.

Vamos lá!

11.7.4 #3 - Firebase Database: Cloud Firestore

Com a autenticação pronta e uma foto de perfil vamos criar um usuário na estrutura de dados do firebase que trabalha com **coleções** e **documentos**.

Basicamente precisamos criar uma coleção de usuários e um documento (que é o usuário efetivamente).

Para executar essa tarefa, devemos primeiro ter em mãos a URL da foto e começar a usar o Firestore. Adicione a dependência

```
implementation 'com.google.firebaseio:firebase-firebase:15.0.0'  
no app/build.gradle.
```

Agora vamos criar uma classe que represente um Usuário com as propriedades:

- UID: user Id
- nome
- Url: Foto do perfil

Esta é a estrutura de um documento salvo dentro do FirebaseFirestore.

Veja que temos uma coleção com 0 ou N documentos, sendo que cada objeto possui campos - fields - e também outras coleções.

Crie um novo arquivo **User.kt** que servirá como uma classe de propriedades para o nosso usuário:

User.kt

```
data class User(val uid: String, val name: String, val url: String)
```

Essa é uma classe em Kotlin com um único objetivo principal - armazenar dados. Basicamente essas classes não possuem regras nem lógicas de programação, são somente espaços de armazenamento de variáveis. Por isso o nome **data class**.

Refatore o seu código de salvar usuário para que ele fique desta forma:

```
private fun saveUserInFirebase() {
    val filename = UUID.randomUUID().toString()
    val ref = FirebaseStorage.getInstance().getReference("/images/${filename}")

    mSelectedUri?.let {
        ref.putFile(it)
        .addOnSuccessListener {
            ref.downloadUrl.addOnSuccessListener {
                Log.i("Teste", it.toString()

                val url = it.toString() // 1
                val name = edit_name.text.toString() // 2
                val uid = FirebaseAuth.getInstance().uid!! // 3

                val user = User(uid, name, url) // 4

                FirebaseFirestore.getInstance().collection("users") // 5
                    .add(user) // 6
                    .addOnSuccessListener { // 7
                        Log.i("Teste", it.id)
                    }
                    .addOnFailureListener { // 8
                        Log.e("Teste", it.message, it)
                    }
                }
            }
        }
    }
}
```

No passo anterior, já tínhamos a URL da foto, agora vamos criar um novo usuário com o Objeto User e atribuir essa URL a ele. Depois, salvar no banco de dados do Firebase.

1. Obtem no formato String a url da foto

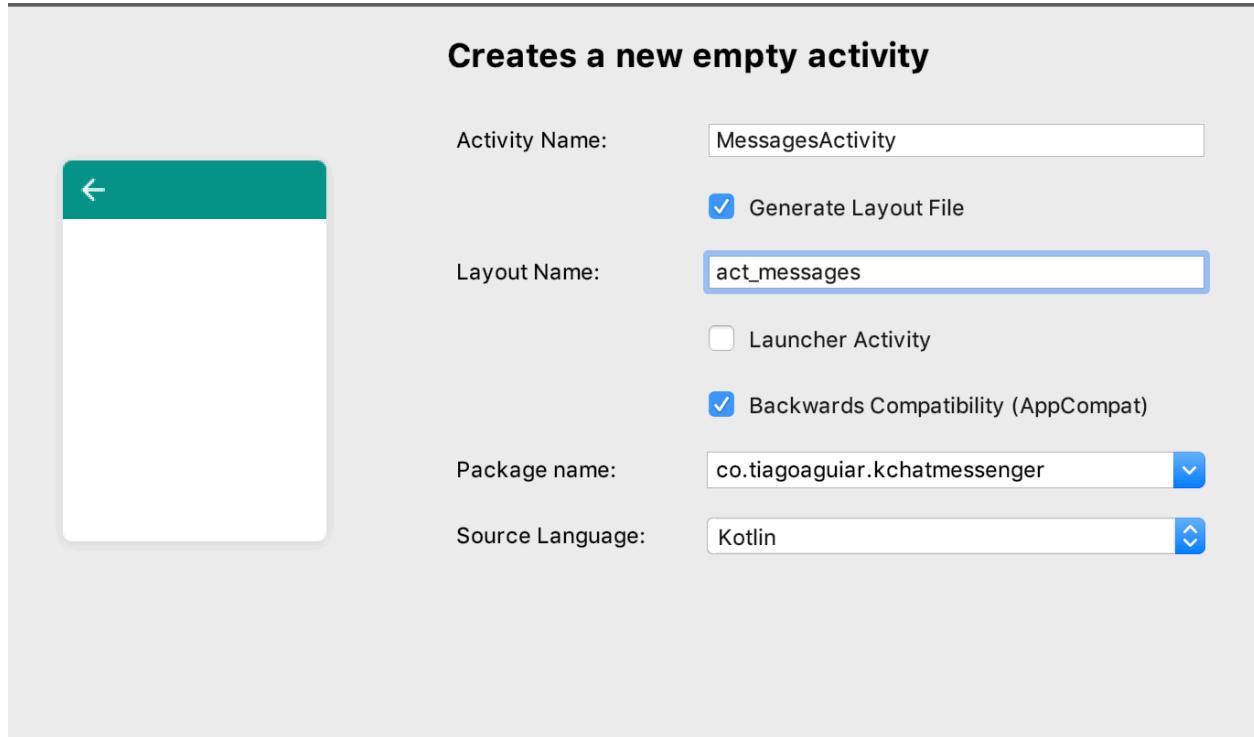


Figure 76: k057

2. Obtem o nome digitado pelo usuário
3. Obtem o UID do usuário autenticado
4. Cria um objeto User a a partir do modelo de classe `data class User`
5. Constrói uma referência a nossa primeira coleção: `users`
6. Efetiva a inserção de um novo usuário
7. Adiciona eventos de sucesso ao criar o novo documento
8. Adiciona eventos de falha ao criar o novo documento

Por hora terminamos a criação completa de um usuário com **Autenticação, Armazenamento de Mídia e Armazenamento de Documento (objeto user)**.

11.8 Lista de Mensagens e Contatos

Nosso próximo passo é exibir a lista com as últimas mensagens enviadas para cada contato e também uma lista de contatos - já cadastrados - para serem selecionados como uma nova conversa.

Após criar um usuário, já devemos redirecioná-lo para a tela de mensagens. Então, vamos criar essa tela de mensagens para realizar este teste.

Crie uma nova activity chamda **MessagesActivity**.

Adicione um texto (HelloWorld) no centro da nova activity - somente para teste.

Adicione a chamada para abrir uma nova activity pela intent. Repare que adicionamos algumas flags para indicar que esta nova activity está no topo de todas as outras. Ou seja, se

você apertar o botão de voltar do smartphone, ao invés de exibir novamente a tela de criar conta, iremos fechar o aplicativo - porque ele já está logado.

```

FirebaseFirestore.getInstance().collection("users")
    .add(user)
    .addOnSuccessListener {
        Log.i("Teste", it.id)

        // Abre uma nova activity e coloca no topo
        val intent = Intent(this@registerActivity,
            MessagesActivity::class.java)
        intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TASK or
        Intent.FLAG_ACTIVITY_NEW_TASK

        startActivity(intent)
    }
    .addOnFailureListener {
        Log.e("Teste", it.message, it)
    }
}

```

Porém, se fecharmos o aplicativo e abri-lo novamente, estaremos na tela de login.

Para entrar direto na tela de login quando estivermos logado, precisaremos fazer algumas alterações no código.

Primeiro, definir nossa **MessagesActivity** como sendo a principal pelo **AndroidManifest.xml**. E então, caso não estiver logado, exibir a Activity de login.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="co.tiagoaguiar.kchatmessenger">

    <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">
        <activity android:name=".LoginActivity" />
        <activity android:name=".RegisterActivity" />
        <activity android:name=".MessagesActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        
```

```

</activity>
</application>

</manifest>

```

Para fazer a checagem, vamos adicionar uma verificação no **onCreate** da `MessagesActivity`.

```

class MessagesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_messages)

        verifyAuthentication()
    }

    private fun verifyAuthentication() {
        if (FirebaseAuth.getInstance().uid == null) {
            val intent = Intent(this@MessagesActivity, LoginActivity::class.java)
            intent.flags =
                Intent.FLAG_ACTIVITY_CLEAR_TASK or Intent.FLAG_ACTIVITY_NEW_TASK
            startActivity(intent)
        }
    }
}

```

Desinstale o aplicativo do celular para remover o *uid* e testar essa nova funcionalidade. Quando você abrir o aplicativo, agora ele deve abrir direto na tela de Login.

Agora, se você criar um novo usuário, ele deverá te levar direto para o tela de Mensagens (que vamos construir mais a frente).

Por fim, vamos adicionar a funcionalidade de *logOut* no nosso aplicativo.

Vamos criar um **menu** na nossa activity.

Basta criar um arquivo `res/menu/message_menu.xml` e escrever o seguinte xml.

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/contacts"
          android:title="@string/contacts"
          app:showAsAction="ifRoom"
          />

    <item android:id="@+id/logout"

```

```

        android:title="@string/logout"
        app:showAsAction="ifRoom"
    />

</menu>

```

Não esqueça de adicionar ao **strings.xml** os títulos do menu.

Adicione o seguinte código para efetivar a criação deste menu xml no aplicativo.

```

override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.message_menu, menu)
    return true
}

```

Execute o aplicativo e veja os 2 novos menus.

```

override fun onOptionsItemSelected(item: MenuItem?): Boolean { // 1
    when(item?.itemId) { // 2
        R.id.logout -> { // 3
            FirebaseAuth.getInstance().signOut() // 4
            verifyAuthentication() // 5
        }
        R.id.contacts -> { // 6
            Log.i("Teste", "contacts clicked")
        }
    }
    return super.onOptionsItemSelected(item)
}

```

1. Método responsável por observar eventos de clicks nos menus
2. Quando o ID do item - menu - for selecionado, faça:
3. Identifica se o evento de click foi gerado pelo menu logout
4. Efetua o Logout no Firebase
5. Verifica novamente qual activity deve ser exibida - neste caso, a de Login.
6. Caso click no menu de contatos, por enquanto apenas faça o Log no console.

Para fechar o ciclo, vamos refatorar o código da tela de Login para que, após efetuar o login, exiba a tela de últimas mensagens (ainda com HelloWorld).

LoginActivity.kt

```

FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
    .addOnCompleteListener {
        if (it.isSuccessful)
            Log.i("Teste", it.result.user.uid)
    }
}

```

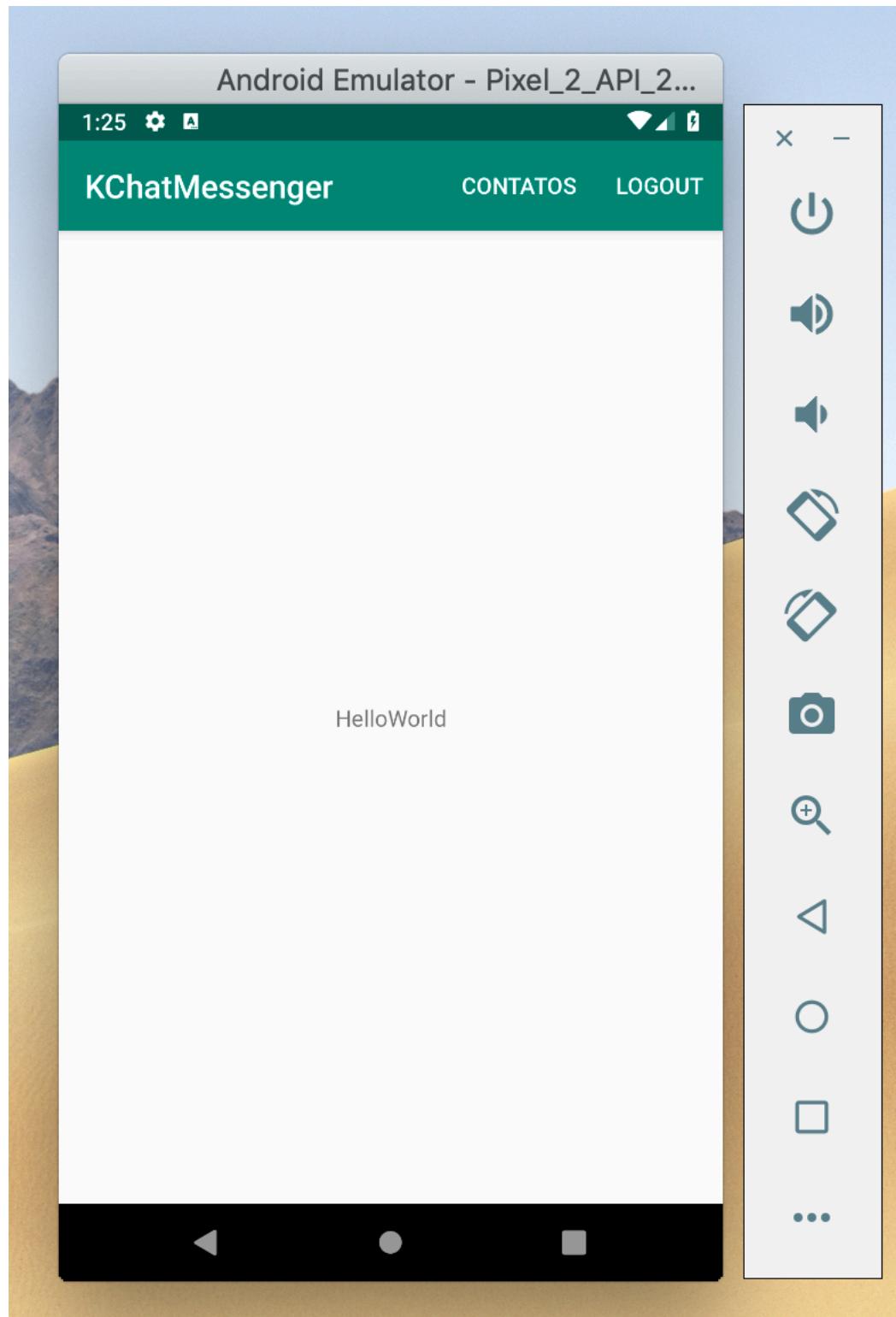


Figure 77: k058

```

// Lógica para abrir a tela MessagesActivity após logIn
    val intent = Intent(this@LoginActivity, MessagesActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK or
    Intent.FLAG_ACTIVITY_NEW_TASK)
    startActivity(intent)
}
.addOnFailureListener {
    Log.e("Teste", it.message, it)
}

```

11.8.1 Criando a Lista de Contatos

Crie mais uma activity chamada **ContactsActivity** usando o método anterior e adicione o evento de click no menu para abrir essa activity.

```

R.id.contacts -> {
    val intent = Intent(this@MessagesActivity, ContactsActivity::class.java)
    startActivity(intent)
}

```

E no **AndroidManifest.xml** vamos declarar essa nova activity e atribuir a activity pai. Isso fará a tela exibir uma seta indicando que há uma activity anterior que podemos voltar.

```

<activity android:name=".ContactsActivity"
          android:label="@string/select_contact">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MessagesActivity" />
</activity>

```

Essa activity terá apenas um componente - **RecyclerView**. A recycler nos ajuda a criar listas de outros componentes e layouts. Dessa forma, podemos criar um layout customizado para cada linha da lista - que será um usuário com foto do perfil e nome.

Para usar a recycler precisamos adicionar a dependência no gradle.

Além da recycler vamos usar outra biblioteca para facilitar a construção das linhas da lista e por fim, uma biblioteca que consegue baixar imagens e colocar em uma ImageView - que será nossas fotos de perfil.

```

implementation 'androidx.recyclerview:recyclerview:1.0.0'
implementation 'com.squareup.picasso:picasso:2.71828'
implementation 'com.xwray:groupie:2.1.0'

```

Adicione essas dependências embaixo das dependências do firebase.

O arquivo **act_contacts.xml** deverá ficar assim.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ContactsActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/list_contact"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Agora vamos “desenhar” a linha que representa um usuário na lista.

item_user.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:background="@android:color/white"
    android:layout_marginBottom="2dp"
    android:layout_height="wrap_content">

    <de.hdodenhof.circleimageview.CircleImageView
        android:id="@+id/img_photo"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:src="@drawable/common_google_signin_btn_icon_dark"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />

```

```

<TextView
    android:id="@+id/txt_username"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:text="TextView"
    app:layout_constraintBottom_toBottomOf="@+id/img_photo"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/img_photo"
    app:layout_constraintTop_toTopOf="@+id/img_photo" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

No lugar da ImageView, vamos adicionar a classe da biblioteca que fará com que as imagens fiquem redondas para deixar nosso app mais elegante.

Agora vamos buscar os usuários no Firebase e depois atribuir cada usuário em uma linha da RecyclerView.

Abra o **ContactsActivity** e crie o bloco de código que irá se conectar no Firestore e buscar todos os usuários cadastrados.

ContactsActivity.kt

```

class ContactsActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_contacts)

        fetchUsers()
    }

    private fun fetchUsers() {
        FirebaseFirestore.getInstance().collection("/users/") // 1
            .addSnapshotListener { snapshot, exception ->
                exception?.let { // 2
                    Log.d("Teste", it.message)
                    return@addSnapshotListener
                }
            }
    }
}

```

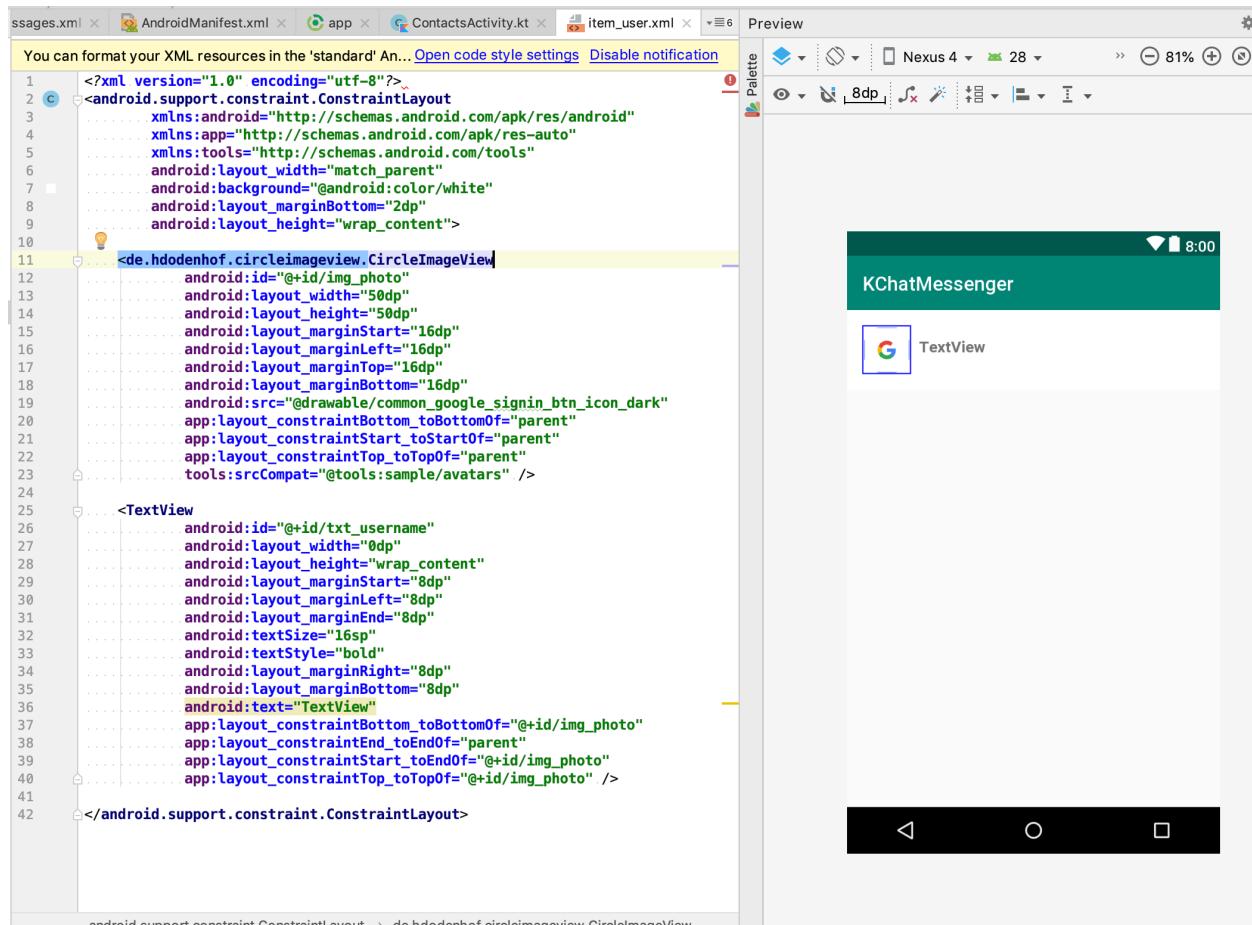


Figure 78: k059

```

        snapshot?.let { // 3
            for (doc in snapshot) { // 4
                val user = doc.toObject(User::class.java) // 5
                Log.i("Teste", "user ${user.uid}, ${user.name}")
            }
        }
    }
}

```

1. Criamos uma referência para o nó (coleção) chamada **users**.
2. Verificamos se ocorreu algum erro no Listener e na chamada do Firebase. Caso ocorra, exibimos no Log.
3. Verificamos se há um objeto com a lista de usuários (documentos).
4. Busca cada documento.
5. Transforma um documento em um objeto tipado para podermos manipular - no nosso caso, User.

O Firebase precisa de um objeto com construtor padrão (sem parâmetros). Para “criar” esse construtor pelo Kotlin, podemos atribuir valores padrão caso não passamos via parâmetro. Veja:

User.kt

```

data class User(val uid: String = "",
               val name: String = "",
               val url: String = "")

```

Desta forma, o Firebase consegue instanciar um novo objeto User usando um construtor padrão (sem parâmetros).

Execute o programa e veja a saída do Log no console com o uid e nome do usuário.

```

user ERB7TAcolPUQvpIgmcjNk8N3WLC2, 1234567
user S20NoHGDLFdRBRMwL40qwxB810a2, Teste
user CqK1b99gLRRAnhUdASsgndsxKbb2, 11
user VWFwlcl7NdU4uoHO4eRSqrBEKkG3, 123456

```

Nosso próximo passo é atribuir cada usuário em uma linha da lista.

Para fazer isso, precisamos criar um componente - **Item<ViewHolder>** - que será responsável por gerenciar o nosso layout **item_user.xml**.

Basicamente, o Recycler cria uma nova View sempre que há necessidade de exibi-la. E quando uma View já é criada, o Recycler toma a decisão de reutilizá-la para evitar recriar sempre um novo objeto.

Defina no onCreate uma instância para o **GroupAdapter** e adicionar esse adapter para a recycler.

```

private lateinit var mAdapter: GroupAdapter<ViewHolder>

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_contacts)

    mAdapter = GroupAdapter()
    list_contact.adapter = mAdapter

    fetchUsers()
}

```

Depois, vamos definir a classe que representa esse novo layout, ou seja, cada linha da lista. Faremos isso com uma classe interna chamada **UserItem**.

```

// onCreate método ...

private inner class UserItem(internal val mUser: User)
    : Item<ViewHolder>() { // 1

    override fun getLayout(): Int { // 2
        return R.layout.item_user
    }

    override fun bind(viewHolder: ViewHolder, position: Int) { // 3
        viewHolder.itemView.txt_username.text = mUser.name // 4
        Picasso.get()
            .load(mUser.url)
            .into(viewHolder.itemView.img_photo) // 5
    }
}

// fetchUsers() método ...

```

1. Declara a classe responsável por gerenciar uma linha da lista. Essa classe recebe em seu construtor um usuário e ela herda as propriedades da classe **Item<ViewHolder>**
2. Define qual será o layout de cada linha
3. Esse método “conecta” os componentes dessa nova view e atribui os dados sempre que ela for exibida
4. Atribui o nome do usuário no campo de texto
5. Atribui a imagem (que será baixada pelo Picasso) no componente ImageView (CircleImageview)

Execute o aplicativo e veja uma lista de usuários criados!

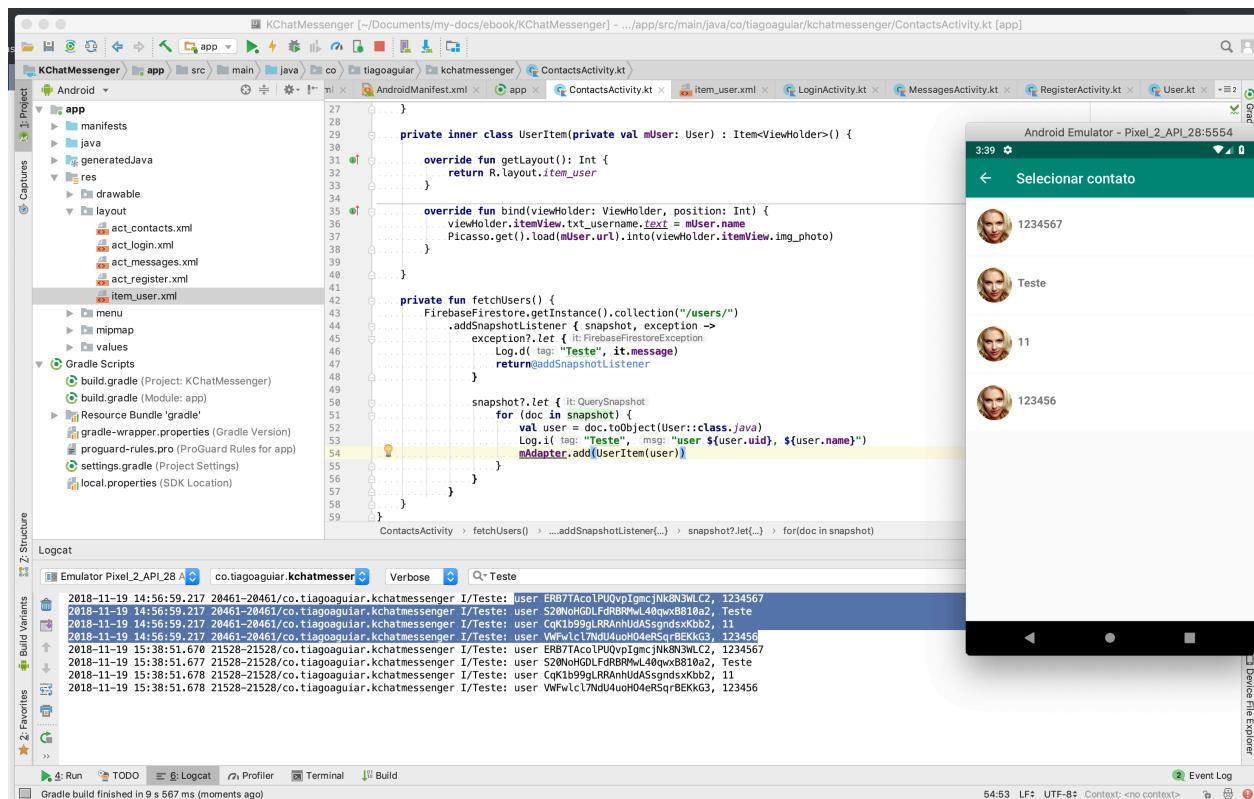


Figure 79: k060

11.9 O Chat!

Chegou a hora de criarmos a tela chat onde as mensagens serão enviadas e recebidas em tempo real.

Crie uma nova Activity chamada **ChatActivity** e adicione a Activity pai no **AndroidManifest.xml**

```
<activity android:name=".ChatActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MessagesActivity"/>
</activity>
```

Agora vamos editar o layout para que ele tenha um editor de texto, uma lista (RecyclerView) e um botão para enviar a mensagem.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
```

```

    android:layout_height="match_parent"
    android:background="@color/bgColor"
    tools:context=".ChatActivity">

    <EditText
        android:id="@+id/edit_msg"
        android:layout_width="0dp"
        android:background="@drawable/bg_edittext_rounded"
        android:layout_height="50dp"
        android:paddingLeft="10dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:ems="10"
        android:hint="@string/hello"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="@+id	btn_send"
        app:layout_constraintEnd_toStartOf="@+id	btn_send"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id	btn_send"
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:background="@drawable/bg_button_rounded"
        android:textColor="@android:color/white"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:text="@string/send"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/list_chat"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintBottom_toTopOf="@+id/edit_msg"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

O Layout deverá parecer como este:

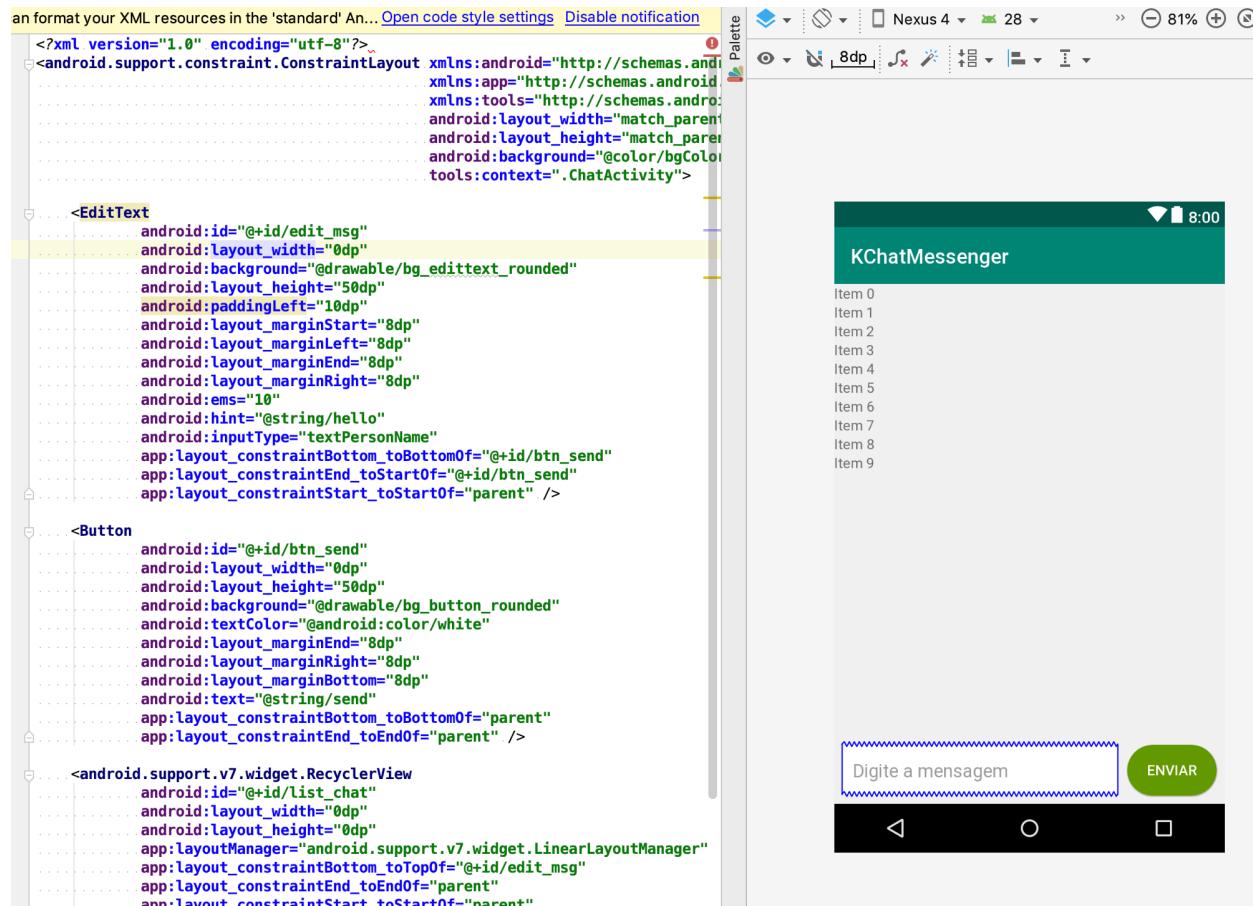


Figure 80: k061

Agora vamos criar os “balões” que representam uma mensagem do usuário que envia e do usuário que recebe.

Crie 2 layouts novos chamados de:

- item_from_message.xml
- item_to_message.xml

item_from_message.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_marginBottom="8dp"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/img_msg_from"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />

    <TextView
        android:id="@+id/txt_msg_from"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:padding="8dp"
        android:background="@drawable/bg_edittext_rounded"
        android:text="This is the new message so crazy, entao o que temos que fazer é"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/img_msg"
        app:layout_constraintTop_toTopOf="@+id/img_msg" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

item_to_message.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_marginBottom="8dp"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/img_msg"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />

    <TextView
        android:id="@+id/txt_msg"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:background="@drawable/bg_edittext_rounded"
        android:padding="8dp"
        android:text="This is the new message so crazy, entao o que temos que fazer é"
        app:layout_constraintEnd_toStartOf="@+id/img_msg"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/img_msg" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Como precisamos diferenciar a esquerda e a direita para indicar a mensagem que foi enviada e a mensagem que foi recebida, criamos 2 layouts.

Vamos inicializar essa nova activity quando clicarmos em um item da linha de contatos.

Abra o arquivo **ContactsActivity.kt** e crie um Listener para escutar eventos de click na linha e faça o teste de clicar no usuário na lista de contatos.

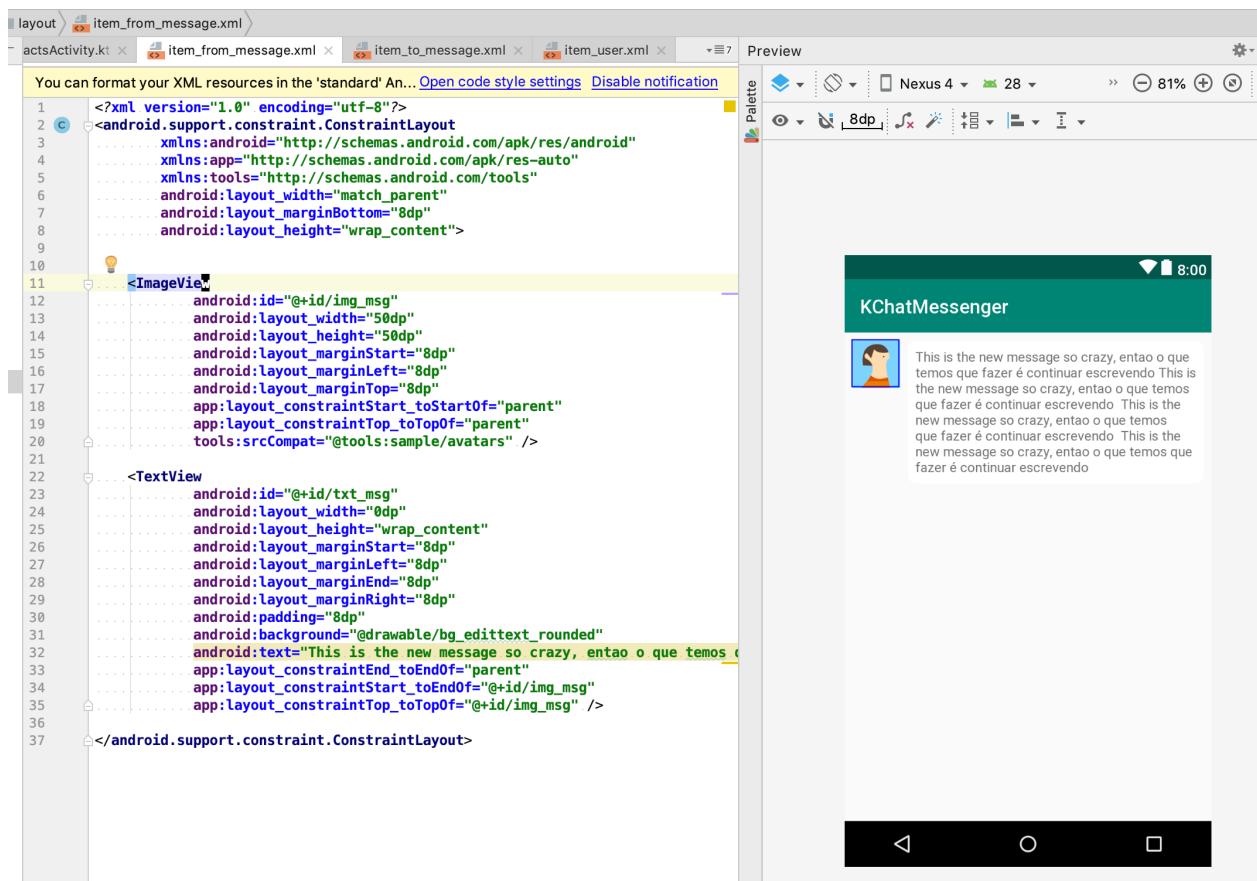


Figure 81: k062

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_contacts)

    mAdapter = GroupAdapter()
    list_contact.adapter = mAdapter

    // cria evento para abrir tela de chat (conversas)
    mAdapter.setOnItemClickListener { item, view ->
        val intent = Intent(this@ContactsActivity, ChatActivity::class.java)
        startActivity(intent)
    }

    fetchUsers()
}

```

Como ainda não temos conversas para exibir, vamos criar algumas *fakes* para visualizar nossos “balões” de conversas.

Abra o arquivo **ChatActivity.kt** e edite para que ele fique da seguinte forma:

```

class ChatActivity : AppCompatActivity() {

    private lateinit var mAdapter: GroupAdapter<ViewHolder>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_chat)

        mAdapter = GroupAdapter()
        list_chat.adapter = mAdapter

        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(false))
        mAdapter.add(MessageItem(false))
        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(false))
    }

    private inner class MessageItem(private val mLeft: Boolean)
        : Item<ViewHolder>() {

        override fun getLayout(): Int {
            return if (mLeft) R.layout.item_from_message
            else R.layout.item_to_message
        }
    }
}

```

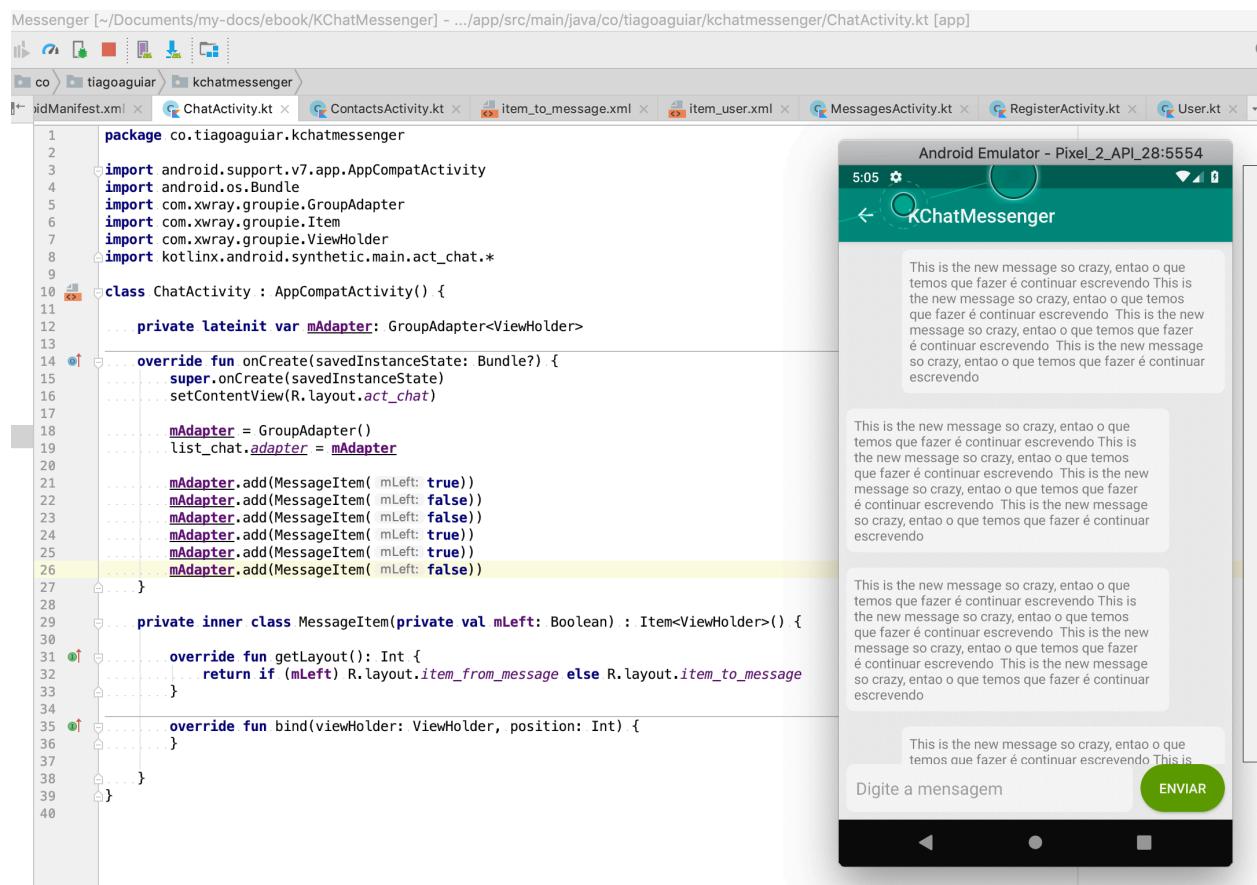


Figure 82: k063

```

    override fun bind(viewHolder: ViewHolder, position: Int) {
    }

}

```

11.9.0.1 Identificando Usuário Na Conversa

Após o evento de clique na lista, enviar o usuário que estamos conversando para a tela de Chat e, posteriormente, identificar os lados que os balões devem ser exibidos.

Para transportar objetos de uma activity para outra precisamos de um recurso conhecido como **Parcelable**.

Para tornar um objeto Parcelable em Kotlin precisamos de uma extensão que facilita essa implementação.

Adicione no seu **build.gradle** o seguinte bloco de código:

```

androidExtensions {
    experimental = true
}

```

```
    }
```

Agora podemos transformar o objeto User em um Parcelable com a Annotation e com a Interface Parcelable:

```
@Parcelize
data class User(val uid: String = "",
                val name: String = "",
                val url: String = "") : Parcelable
```

Feito isso, vamos alterar o evento `setOnItemClickListener` para enviar o usuário que queremos conversar para a outra Activity.

ContactsActivity

```
companion object {
    val USER_KEY = "user_key"
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_contacts)

    mAdapter = GroupAdapter()
    list_contact.adapter = mAdapter

    mAdapter.setOnItemClickListener { item, view ->
        val intent = Intent(this@ContactsActivity, ChatActivity::class.java)
        val userItem: UserItem = item as UserItem

        intent.putExtra(USER_KEY, userItem.mUser)
        startActivity(intent)
    }

    fetchUsers()
}
```

Adicionamos o usuário na intent e podemos buscar na outra Activity.

ChatActivity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_chat)

    val user = intent.extras.getParcelable<User>(ContactsActivity.USER_KEY)
    Log.i("Teste", "username ${user.name}")
```

```

        supportActionBar?.title = user.name

        mAdapter = GroupAdapter()
        list_chat.adapter = mAdapter

        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(false))
        mAdapter.add(MessageItem(false))
        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(true))
        mAdapter.add(MessageItem(false))
    }

```

Através da *intent* podemos buscar um objeto **Parcelable**.

Ao executar o aplicativo e clicar em um item da lista, veremos seu nome na Activity de Chat.

11.10 Refatorando Documentos e Coleções do Firestore

Vamos fazer alguns ajustes na estrutura do nosso documento para que o aplicativo funcione com as conversas novas.

Hoje, estamos gerando um documento com identificador aleatório, mas isso é problema porque precisaremos que o identificador seja atrelado a um usuário. Logo, vamos realizar as alterações para que o identificador tenha como chave o *uid* do usuário.

Desta forma, podemos buscar um usuário pelo seu uid direto da coleção de usuários (mais a frente você entenderá o por quê desta refatoração).

users	r8C1Y1LXwS6Aw1qZIBFI
+ Add document	+ Add collection
7KS3WZtXRDu2aRutFrEm	
OhzrmUuW5s2tkD1oz0ap	
WIU2AR6hoya9YS6KEYpS	
r8C1Y1LXwS6Aw1qZIBFI	+ Add field name: "123456" uid: "VWFwlcl7NdU4uoHO4eRSqrBEKkG3" url: "https://firebasestorage.googleapis.com/v0/b/kchatmessenger.appspot.com/c22d7-4f00-8434-326bf7ff0c45?alt=media&token=a4704f97-3bc9-4f1c-8d25-e9"
tEvNcnGCdVYzECeItTJfMeC4Mfg2	

Abra o RegisterActivity e refatore o método `saveUserInFirebase()` para que ele adicione o uid como chave do documento.

```
Firestore.getInstance().collection("users")
    .document(uid) // adicionando uid como chave
    .set(user) // atribuindo o usuário a esta chave (uid)
    .addOnSuccessListener {
        val intent = Intent(this@RegisterActivity, MessagesActivity::class.java)
        intent.flags =
            Intent.FLAG_ACTIVITY_CLEAR_TASK or Intent.FLAG_ACTIVITY_NEW_TASK

        startActivity(intent)
    }
    .addOnFailureListener {
        Log.e("Teste", it.message, it)
    }
}
```

Crie um novo usuário e acesse o painel do firebase para visualizar o novo documento com o uid sendo chave.

11.11 Enviando e Recebendo Mensagens do Chat

Para enviar uma documento de mensagem ao chat e começar a registrar as conversas precisaremos criar um objeto modelo de **Mensagem**.

Crie um arquivo chamado **Message.kt** e adicione as propriedades:

```
data class Message(val text: String = "",
                   val timestamp: Long = 0,
                   val fromId: String = "",
                   val toId: String = "")
```

Vamos alterar o **MessageItem** para que ele receba essa mensagem e identifique qual será o layout que exibiremos dependendo do usuário (remetente ou destinatário)

```
private inner class MessageItem(private val mMessage: Message)
    : Item<ViewHolder>() {

    override fun getLayout(): Int {
        return if (mMessage.fromId == FirebaseAuth.getInstance().uid)
            R.layout.item_from_message
        else
            R.layout.item_to_message
    }

    override fun bind(viewHolder: ViewHolder, position: Int) {
```

```

        txt_msg.text = mMessage.text
        Picasso.get().load(mUser.url).into(img_photo)
    }

}

```

E ao clicar no botão **Enviar** vamos enviar a mensagem para o Firebase.

```

lateinit var mUser: User // 1

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.act_chat)

    mUser = intent.extras.getParcelable<User>(ContactsActivity.USER_KEY)
    Log.i("Teste", "username ${mUser.name}")
    supportActionBar?.title = mUser.name

    mAdapter = GroupAdapter()
    list_chat.adapter = mAdapter

    btn_send.setOnClickListener {
        sendMessage()
    }
}

private fun sendMessage() {
    val text = edit_msg.text.toString()

    edit_msg.text = null // 2

    val fromId = FirebaseAuth.getInstance().uid.toString()
    val toId = mUser.uid
    val timestamp = System.currentTimeMillis()

    val message = Message(text = text,
        timestamp = timestamp, toId = toId, fromId = fromId) // 3

    if (!message.text.isEmpty()) {
        // 4
        FirebaseFirestore.getInstance().collection("/conversations")
            .document(fromId)
            .collection(toId)
            .add(message)
            .addOnSuccessListener {
                Log.i("Teste", it.id)
            }
    }
}

```

```

        .addOnFailureListener {
            Log.e("Teste", it.message)
        }

    // 5
    FirebaseFirestore.getInstance().collection("/conversations")
        .document(toId)
        .collection(fromId)
        .add(message)
        .addOnSuccessListener {
            Log.i("Teste", it.id)
        }
        .addOnFailureListener {
            Log.e("Teste", it.message)
        }
    }
}
}

```

1. Guarda uma referência ao objeto User
2. Apaga o EditText após clicar no botão enviar. Permitindo escrever uma nova mensagem
3. Cria o objeto de Mensagem
4. Cria uma nova coleção chamada **conversations** que terá basicamente documentos dos remetentes. Esses documentos (remetentes) podem ter 1 ou vários destinatários (outra coleção) e, para cada remetente, teremos 1 ou várias mensagens
5. Fazemos a mesma coisa só que invertida, para que quando o outro usuário abrir o aplicativo e poder visualizar os balões de quem recebeu e de quem enviou. Quando terminarmos a implementação da lista de contatos vamos ver essa funcionalidade.

Abra o seu painel do Firebase e veja as coleções e documentos que foram criados.

Agora vamos buscar essas mensagens e exibir os balões de conversas.

Boa parte da classe será refatorada. Dê uma olhada de como ficou nosso código final

```

class ChatActivity : AppCompatActivity() {

    private lateinit var mAdapter: GroupAdapter<ViewHolder>

    lateinit var mUser: User

    private var mMe: User? = null // 1

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_chat)

        mUser = intent.extras.getParcelable<User>(ContactsActivity.USER_KEY)
    }
}

```

```

Log.i("Teste", "username ${mUser.name}")
supportActionBar?.title = mUser.name

mAdapter = GroupAdapter()
list_chat.adapter = mAdapter

btn_send.setOnClickListener {
    sendMessage()
}

FirebaseFirestore.getInstance().collection("/users") // 2
    .document(FirebaseAuth.getInstance().uid.toString())
    .get()
    .addOnSuccessListener {
        mMe = it.toObject(User::class.java) // 3
        fetchMessages()
    }
}

private fun fetchMessages() {
    mMe?.let {
        val fromId = it.uid.toString()
        val toId = mUser.uid.toString()

        FirebaseFirestore.getInstance().collection("/conversations") // 4
            .document(fromId)
            .collection(toId)
            .orderBy("timestamp", Query.Direction.ASCENDING) // 5
            .addSnapshotListener { querySnapshot, firebaseFirestoreException ->
                querySnapshot?.documentChanges?.let {
                    for (doc in it) {
                        when(doc.type) {
                            DocumentChange.Type.ADDED -> { // 6
                                val message =
                                    doc.document.toObject(Message::class.java)
                                mAdapter.add(MessageItem(message))
                            }
                        }
                    }
                }
            }
    }
}

private fun sendMessage() {
    val text = edit_msg.text.toString()

```

```

edit_msg.text = null

val fromId = FirebaseAuth.getInstance().uid.toString()
val toId = mUser.uid
val timestamp = System.currentTimeMillis()

val message = Message(text = text,
    timestamp = timestamp, toId = toId, fromId = fromId)

if (!message.text.isEmpty()) {
    FirebaseFirestore.getInstance().collection("/conversations")
        .document(fromId)
        .collection(toId)
        .add(message)
        .addOnSuccessListener {
            Log.i("Teste", it.id)
        }
        .addOnFailureListener {
            Log.e("Teste", it.message)
        }
}

FirebaseFirestore.getInstance().collection("/conversations")
    .document(toId)
    .collection(fromId)
    .add(message)
    .addOnSuccessListener {
        Log.i("Teste", it.id)
    }
    .addOnFailureListener {
        Log.e("Teste", it.message)
    }
}

private inner class MessageItem(private val mMessage: Message)
    : Item<ViewHolder>() {

    override fun getLayout(): Int {
        return if (mMessage.fromId == FirebaseAuth.getInstance().uid)
            R.layout.item_from_message
        else
            R.layout.item_to_message
    }

    override fun bind(viewHolder: ViewHolder, position: Int) {
        if (mMessage.fromId == FirebaseAuth.getInstance().uid) {

```

```
        viewHolder.itemView.txt_msg_from.text = mMessege.text
        Picasso.get().load(mMe?.url).into(viewHolder.itemView.img_msg_from) // 7
    } else {
        viewHolder.itemView.txt_msg.text = mMessege.text
        Picasso.get().load(mUser.url).into(viewHolder.itemView.img_msg) // 8
    }
}
}
```

1. Declare uma referência para o usuário logado
 2. Busca os dados do usuário logado
 3. Transforma um documento em um objeto User
 4. Busca as conversas dos usuários
 5. Ordena as conversas pela data de inserção (timestamp)
 6. Escuta eventos para que quando uma nova mensagem for adicionada, em tempo real, o outro usuário possa visualizar essa mensagem
 7. Carrega a foto do perfil do usuário logado
 8. Carrega a foto do perfil do usuário (destinatário)

Para corrigir o layout vamos alterar alguns aspectos do design como imagens redondas e bordas arredondadas da mensagem.

item_from_message.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_marginBottom="8dp"
    android:layout_height="wrap_content">

    <de.hdodenhof.circleimageview.CircleImageView
        android:id="@+id/img_msg_from"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />
```

```

<TextView
    android:id="@+id/txt_msg_from"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:background="@drawable/bg_edittext_rounded"
    android:maxLength="240dp"
    android:padding="8dp"
    android:text="Oi"
    app:layout_constraintStart_toEndOf="@+id/img_msg_from"
    app:layout_constraintTop_toTopOf="@+id/img_msg_from" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

item_to_message.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_marginBottom="8dp"
    android:layout_height="wrap_content">

```

```

<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/img_msg"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@tools:sample/avatars" />

```

```

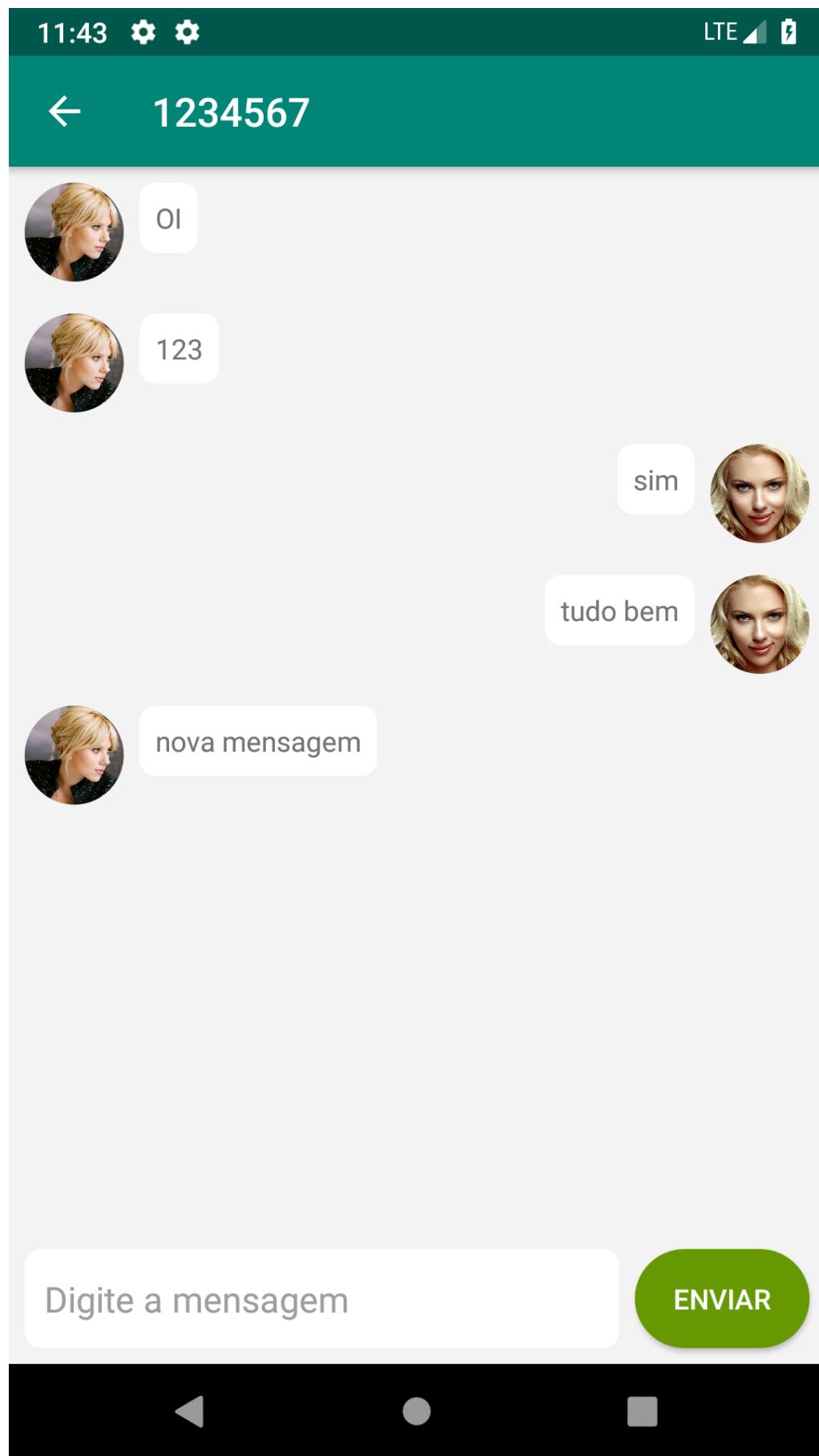
<TextView
    android:id="@+id/txt_msg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:background="@drawable/bg_edittext_rounded"
    android:maxLength="240dp"
    android:padding="8dp"

```

```
        android:text="This is the new message so crazy, entao o que temos que fazer é
        app:layout_constraintEnd_toStartOf="@+id/img_msg"
        app:layout_constraintTop_toTopOf="@+id/img_msg" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```

Após todos esses ajustes, execute o aplicativo e veja o resultado final das conversas do chat.



11.12 Listando as Últimas Mensagens de Cada Contato

Para finalizar nosso aplicativo vamos listar as últimas mensagens de cada contato na tela **MessagesActivity**.

Para listar essas mensagens, primeiramente precisamos adicionar a mensagem em um novo nó chamado de **/last-messages**.

Crie um novo modelo de classe chamado **Contact.kt**.

```
data class Contact(
    val uuid: String = "",
    val username: String = "",
    val lastMessage: String = "",
    val photoUrl: String = "",
    val timestamp: Long = 0
)
```

Agora crie uma coleção para inserir o último contato de cada usuário.

```
FirebaseFirestore.getInstance().collection("/conversations")
    .document(fromId)
    .collection(toId)
    .add(message)
    .addOnSuccessListener {
        Log.i("Teste", it.id)

        val contact = Contact(toId,
            mUser.name, text, mUser.url, message.timestamp)

        FirebaseFirestore.getInstance().collection("/last-messages")
            .document(fromId)
            .collection("contacts")
            .document(toId)
            .set(contact)
    }
    .addOnFailureListener {
        Log.e("Teste", it.message)
    }

FirebaseFirestore.getInstance().collection("/conversations")
    .document(toId)
    .collection(fromId)
    .add(message)
    .addOnSuccessListener {
        Log.i("Teste", it.id)
```

```

    val contact = Contact(toId,
        mUser.name, text, mUser.url, message.timestamp)

    FirebaseFirestore.getInstance().collection("/last-messages")
        .document(toId)
        .collection("contacts")
        .document(fromId)
        .set(contact)
    }
    .addOnFailureListener {
        Log.e("Teste", it.message)
    }
}

```

Com as últimas mensagens criadas podemos criar uma RecyclerView com os itens para poder exibir.

Altere o layout da MessagesActivity com uma RecyclerView.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MessagesActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/list_messages"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

E também adicione o layout de cada linha dessa RecyclerView - a linha da mensagem.

item_user_message.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginBottom="2dp"
android:background="@android:color/white">

<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/img_photo"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginBottom="16dp"
    android:src="@drawable/common_google_signin_btn_icon_dark"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@tools:sample/avatars" />

<TextView
    android:id="@+id/txt_username"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginBottom="8dp"
    android:text="TextView"
    android:textSize="14sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@+id/txt_last_message"
    app:layout_constraintStart_toEndOf="@+id/img_photo"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_chainStyle="packed" />

<TextView
    android:id="@+id/txt_last_message"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:textSize="16sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"

```

```

        app:layout_constraintStart_toEndOf="@+id/img_photo"
        app:layout_constraintTop_toBottomOf="@+id/txt_username" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

O próximo passo é praticamente o mesmo do que já fizemos - criar uma classe interna para representar um item da linha e buscar no Firebase a coleção de últimas mensagens.

Segue abaixo a classe interna - Adapter.

```

private inner class ContactItem(private val mContact: Contact)
    : Item<ViewHolder>() {

    override fun getLayout(): Int {
        return R.layout.item_user_message
    }

    override fun bind(viewHolder: ViewHolder, position: Int) {
        viewHolder.itemView.txt_last_message.text = mContact.lastMessage
        viewHolder.itemView.txt_username.text = mContact.username
        Picasso.get()
            .load(mContact.photoUrl)
            .into(viewHolder.itemView.img_photo)
    }

}

```

E agora o método responsável por buscar as mensagens no Firebase. Veja abaixo a classe completa já buscando as últimas mensagens.

```

class MessagesActivity : AppCompatActivity() {

    private lateinit var mAdapter: GroupAdapter<ViewHolder>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.act_messages)

        mAdapter = GroupAdapter()
        list_messages.adapter = mAdapter

        verifyAuthentication()

        fetchLastMessage()
    }

    private fun fetchLastMessage() {

```

```

    val uid = FirebaseAuth.getInstance().uid.toString()

    FirebaseFirestore.getInstance().collection("/last-messages")
        .document(uid)
        .collection("contacts")
        .addSnapshotListener { querySnapshot, firebaseFirestoreException ->
            val changes = querySnapshot?.documentChanges
            changes?.let {
                for (doc in it) {
                    when (doc.type) {
                        DocumentChange.Type.ADDED -> {
                            val contact =
                                doc.document.toObject(Contact::class.java)
                            mAdapter.add(ContactItem(contact))
                        }
                    }
                }
            }
        }
    }

    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        when(item?.itemId) {
            R.id.logout -> {
                FirebaseAuth.getInstance().signOut()
                verifyAuthentication()
            }
            R.id.contacts -> {
                val intent =
                    Intent(this@MessagesActivity, ContactsActivity::class.java)
                startActivity(intent)
            }
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.message_menu, menu)
        return true
    }

    private fun verifyAuthentication() {
        if (FirebaseAuth.getInstance().uid == null) {
            val intent =
                Intent(this@MessagesActivity, LoginActivity::class.java)
            intent.flags =
                Intent.FLAG_ACTIVITY_CLEAR_TASK or Intent.FLAG_ACTIVITY_NEW_TASK
        }
    }
}

```

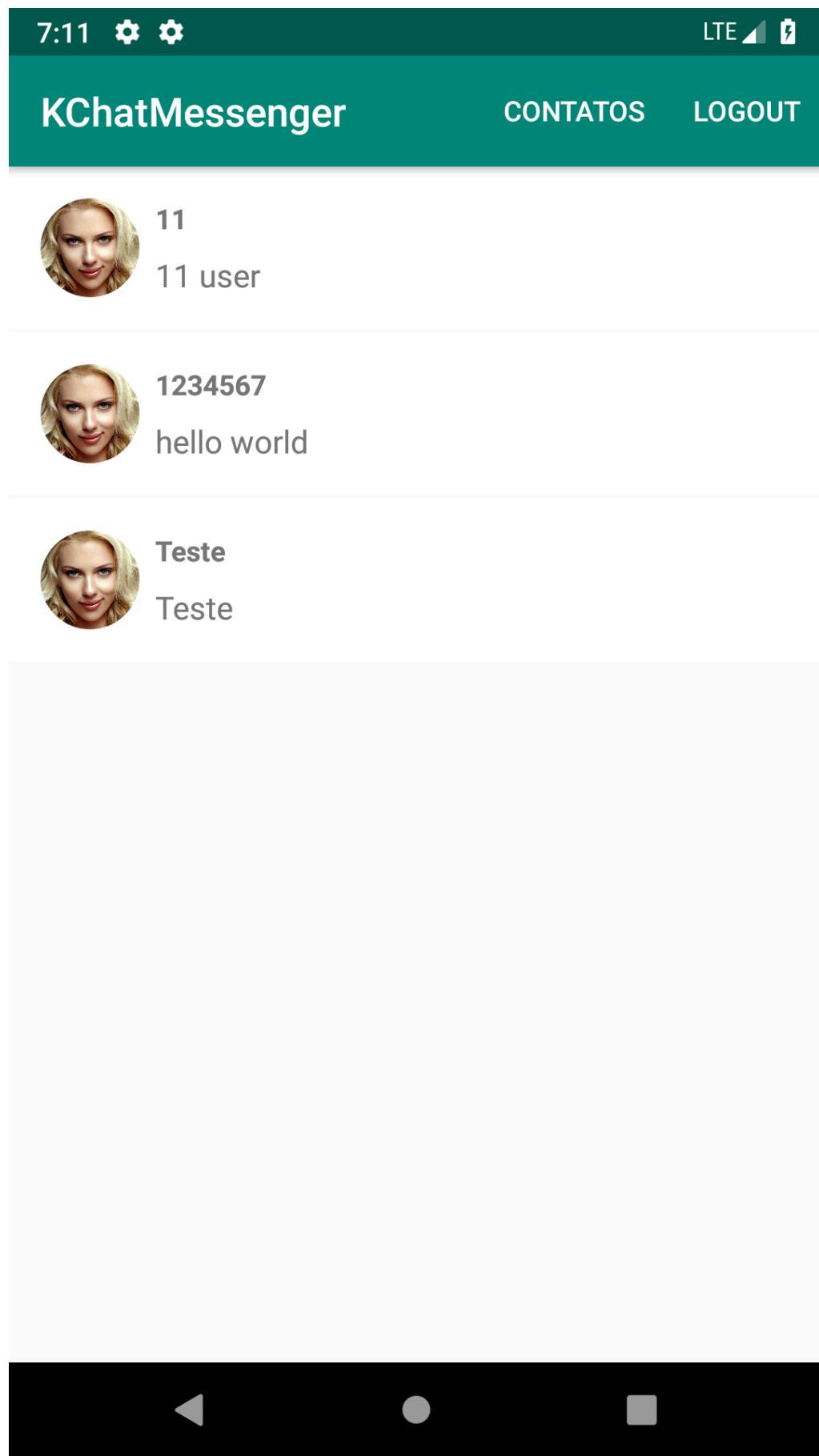
```
        startActivity(intent)
    }
}

private inner class ContactItem(private val mContact: Contact)
    : Item<ViewHolder>() {

    override fun getLayout(): Int {
        return R.layout.item_user_message
    }

    override fun bind(viewHolder: ViewHolder, position: Int) {
        viewHolder.itemView.txt_last_message.text = mContact.lastMessage
        viewHolder.itemView.txt_username.text = mContact.username
        Picasso.get()
            .load(mContact.photoUrl)
            .into(viewHolder.itemView.img_photo)
    }
}
```

Execute o aplicativo e você verá a última mensagem de cada conversa feita pelos usuários.



Para garantir que todo o fluxo do Chat está funcionando corretamente, apague todos os registros do Firebase - Storage, Database e Authentication - e comece a criação de 2 ou 3 usuários novamente.

Desta forma, os nós (coleções e documentos) serão refeitos com a versão final do aplicativo.

12 Capítulo 8: Publicando o Aplicativo no Google Play

Parabéns! Você conseguiu construir um aplicativo de Chat do zero usando a linguagem de programação Kotlin. Agora o “céu é o limite” ;)

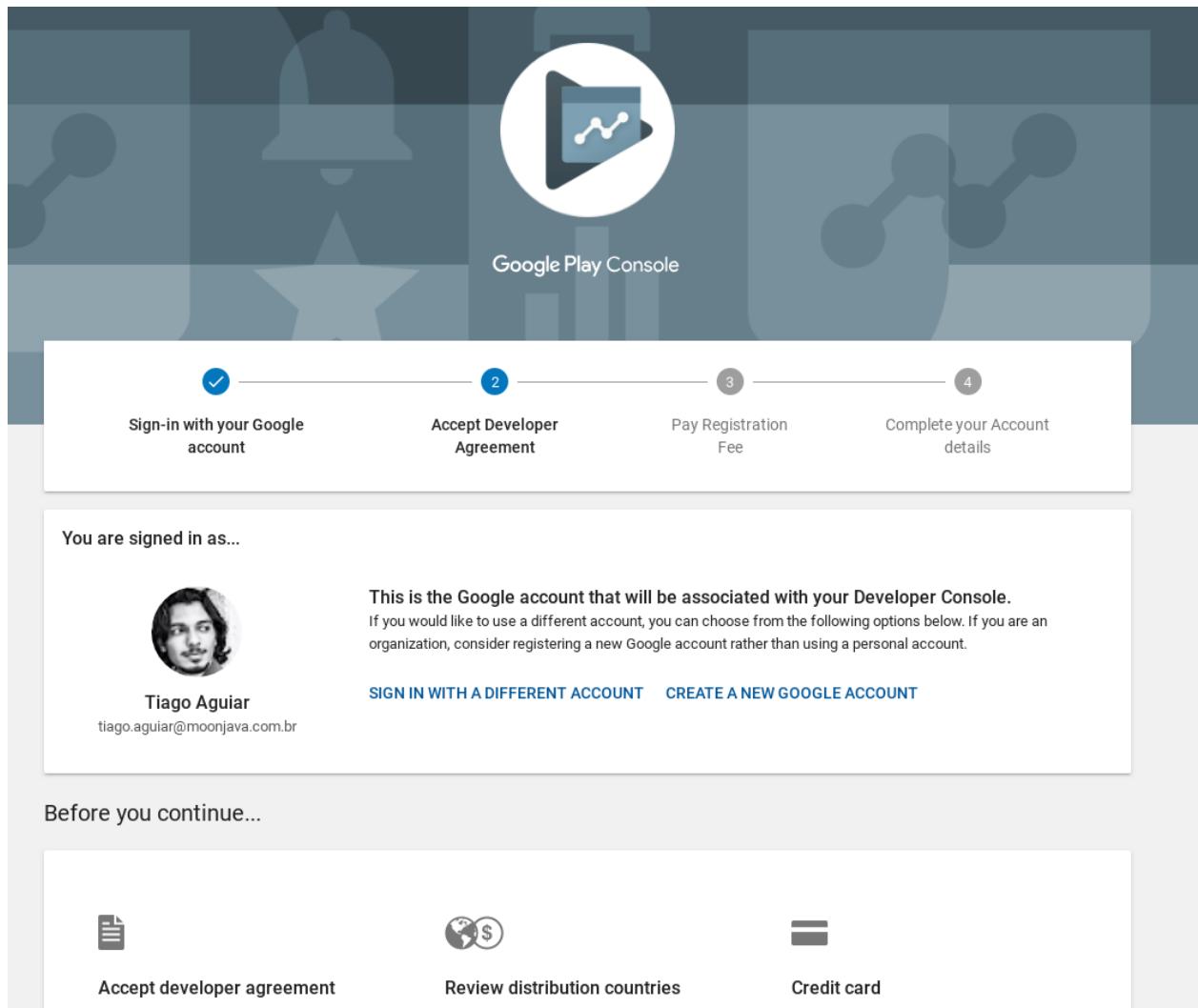
Claro que ainda temos muito a estudar. Na verdade, o conhecimento nunca para por completo. Ainda temos conceitos de arquitetura de software, padrões de Design e muito mais.

Pois bem, para fechar com chave de ouro, vamos subir nosso primeiro aplicativo Android no Google Play.

Para subir um aplicativo precisaremos de uma conta Google (pode ser um gmail) e se cadastrar no programa de desenvolvedores do Google conhecido como Google Developers.

Esse programa tem um custo único de apenas \$25 dólares para você subir quantos aplicativos quiser.

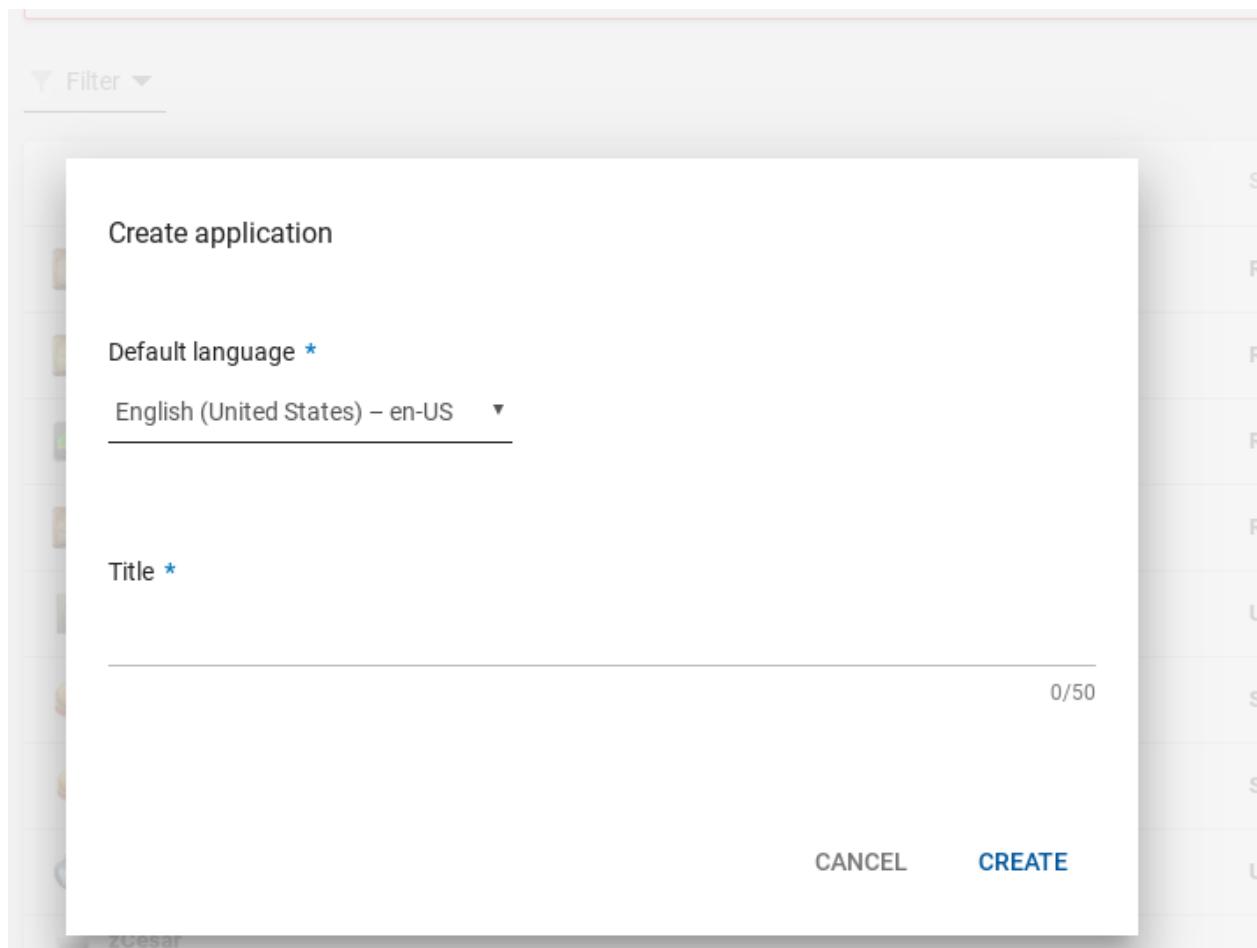
Acesse o link [google publish](#) e aceite os termos do Google, efetue o pagamento e siga com detalhes da sua conta.



Uma vez com a conta ativa, vamos criar o modelo do nosso aplicativo.

12.1 Criando um Aplicativo no Google Play

O primeiro passo é criar uma nova aplicação clicando no botão **CREATE APPLICATION** ou **criar aplicativo**.



Escolha a lingua principal do aplicativo e o título que irá aparecer na Google Play.

Agora uma tela irá abrir com vários campos a serem preenchidos que dizem muito sobre o seu aplicativo. No nosso caso, vamos preencher e mostrar os campos que precisamos:

- Título: KChatMessenger
- Curta descrição: Aplicativo de Chat usando o Firebase para Teste
- Longa descrição: Aplicativo de Chat usando o Firebase para Teste
- Screenshots: Com o emulador aberto, aperte no ícone da Camera para capturar as fotos do aplicativo e subir no Google Play
- Application Type: Tipo de aplicativo (Jogos ou Apps). No nosso caso será aplicativo
- Categoria: A categoria onde o aplicativo se encaixa. No nosso caso poderia ser Social.
- Email: Seu e-mail de contato obrigatório
- Política de privacidade: Se você não tiver um website com política de privacidade poderá selecionar a opção **Não enviar URL de política de privacidade**

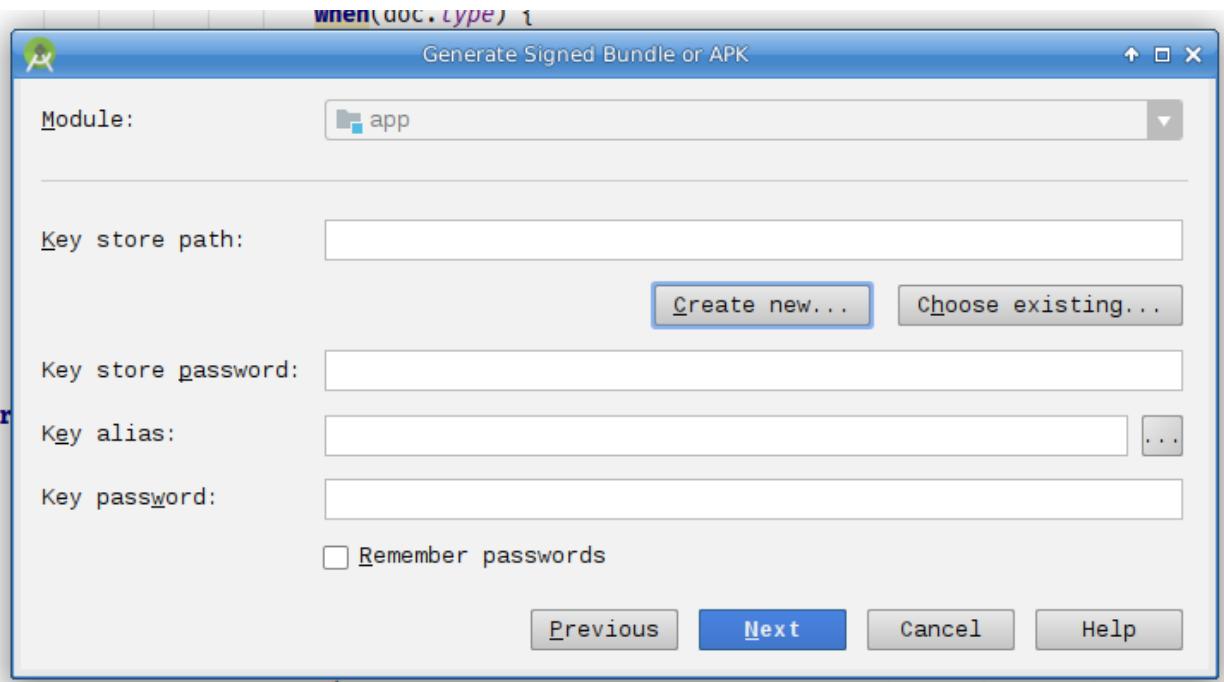
Clique para salvar o rascunho e continuar a preencher outros campos.

De volta no Android Studio, vamos gerar o arquivo final do projeto conhecido como **.apk**.

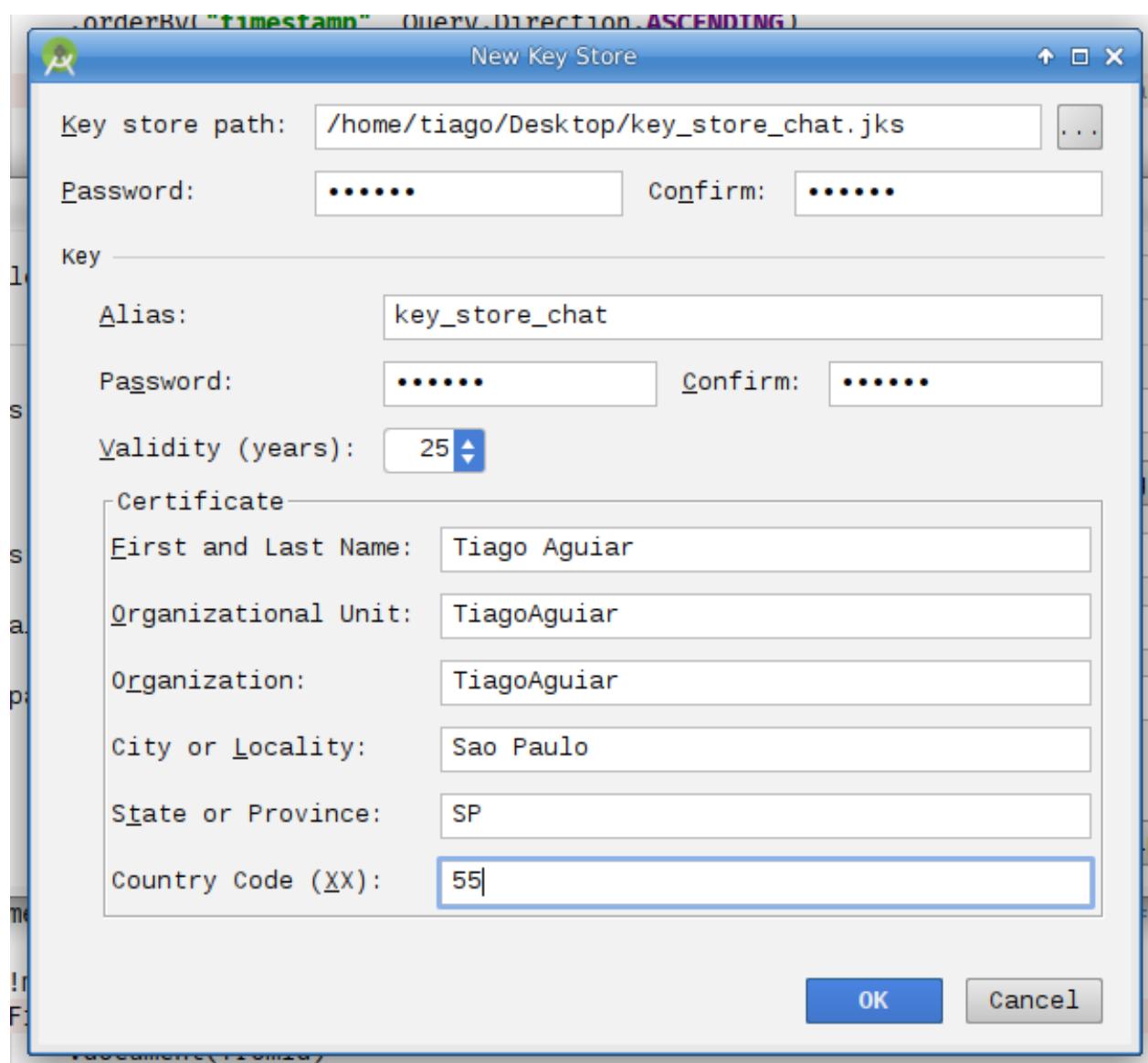
Este é o pacote final que deve ser enviado ao Google Play. É ali que contém todos os arquivos compilados e prontos para serem executados em outros smartphones e tablets.

Acesse o menu **Build > Generate Signed Bundle / Apk...**

Selecione a opção **Apk** e clique em **Next**.



Se você ainda não tiver uma key_store (que é a sua chave única para subir os apps), clique em **create new...**



Preencha todos os campos corretamente e clique em Ok.

Depois clique em Next com os dados preenchidos.

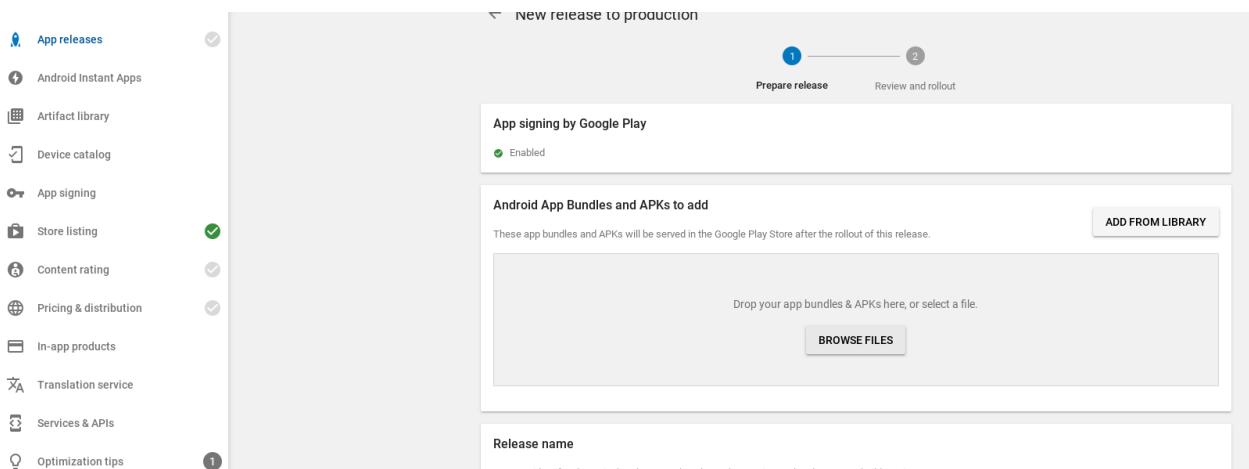
Nesta próxima tela o Android Studio mostrará onde o apk será gerado. No meu caso, será no diretório KChatMessenger/app.

Marque a opções: - V1 (Jar Signature) - V2 (Full Apk Signature)

Clique em **Finish**.

De volta ao Google Play, selecione no menu ao lado a opção **App Releases** (ou lançamentos dependendo do idioma do seu google play).

Na guia **Produção** clique em **Gerenciar**. Faça o upload do Apk que deve estar no diretório app/release/app-release.apk:



No bloco onde pede uma breve descrição dessa versão, coloque:

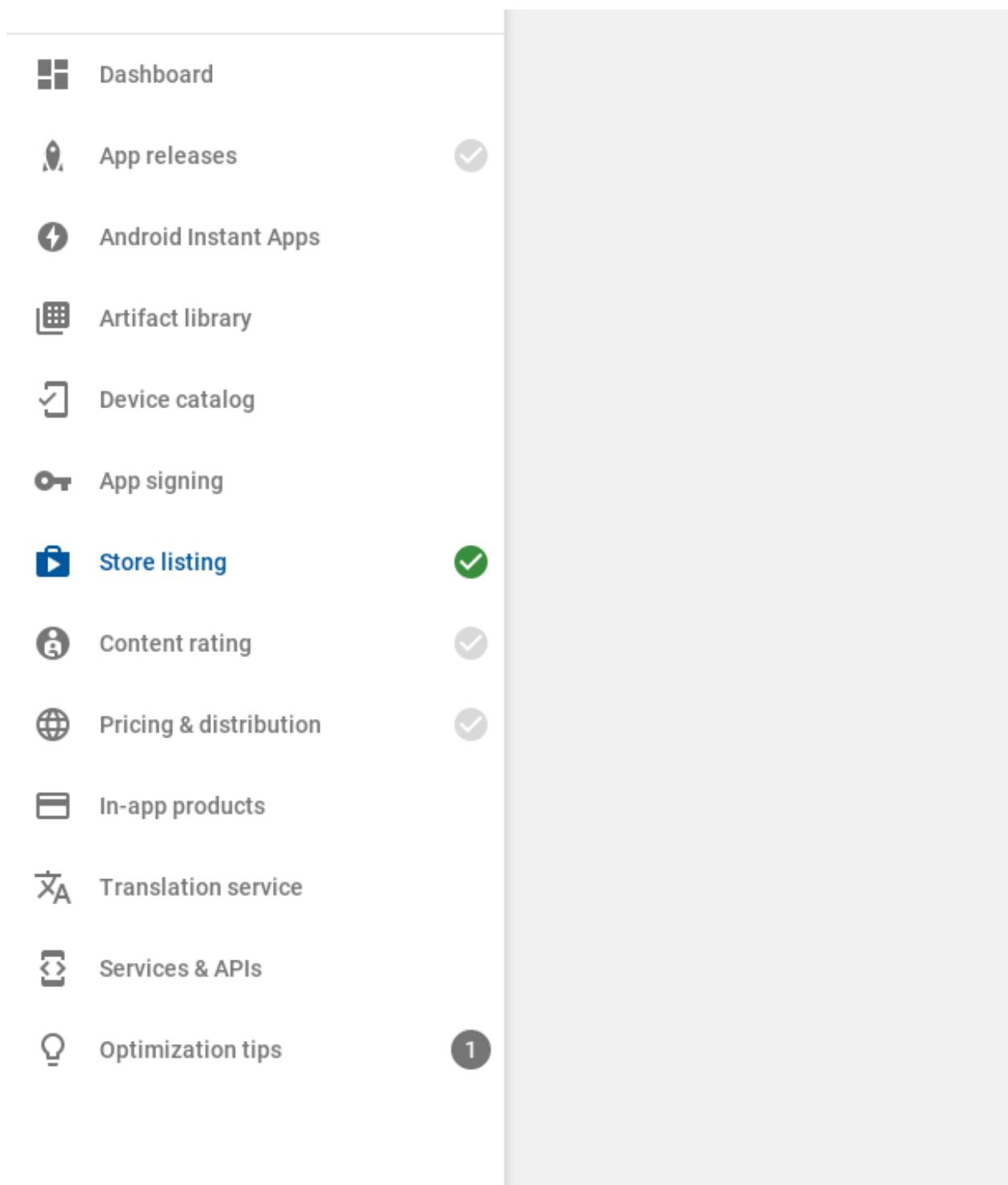
```
<pt-BR>
Primeira versão,
</pt-BR>
```

Clique em Salvar e depois Revisar.

Agora acesse o painel de **preço e distribuição** e defina como será “comercializado” o seu aplicativo

- Vamos definir como gratuito
- Escolher os países onde ele ficará disponível: Brasil por enquanto
- Selecione a opção onde indica que o aplicativo não é para o público infantil
- Selecione a opção onde indica que o aplicativo não contém anúncios patrocinados
- Selecione a opção sobre a guideline do Google e sobre as leis (export laws)

Agora no menu ao lado, selecione a opção **Content rating** e clique em **Continue**



Nesta parte você deve digitar o seu email e selecionar a categoria do aplicativo. Vamos selecionar Social e responder as perguntas conforme a imagem abaixo:

Please complete the questionnaire so that we can calculate your app rating.

 **SOCIAL NETWORKING, FORUMS, BLOGS, AND UGC SHARING**
App is a social networking app. [Edit Category](#)

APP TYPE CLOSE 

Is the primary focus of the app to connect people for the purposes of dating or sexual relationships or endeavors? * [Learn more](#)

Yes No

Does the app permit the public sharing of nudity? *

Yes No

Does the app permit the public sharing of real-world, graphic violence outside of a newsworthy context? *

Yes No

Does the app share the user's current physical location with other users? * [Learn more](#)

Yes No

Does the app allow users to purchase digital goods? * [Learn more](#)

Yes No

[CALCULATE RATING](#) [SAVE QUESTIONNAIRE](#) **IARC**

Salve o questionário e clique para calcular a **classificação indicativa** do seu aplicativo. Depois, clique em aplicar.

Volte ao menu do aplicativo para lançamento. Clique em editar > revisar > iniciar lançamento

← Confirm rollout to production: 1.0

1 2

Prepare release Review and rollout

Review summary

This release is ready to be rolled out.

APKs in this release

Expand all

Type	Version code	Uploaded	APK download size	Installs on active devices
1 APK added	1	6 minutes ago	3.10 MB	No data
APK	1			

What's new in this release?

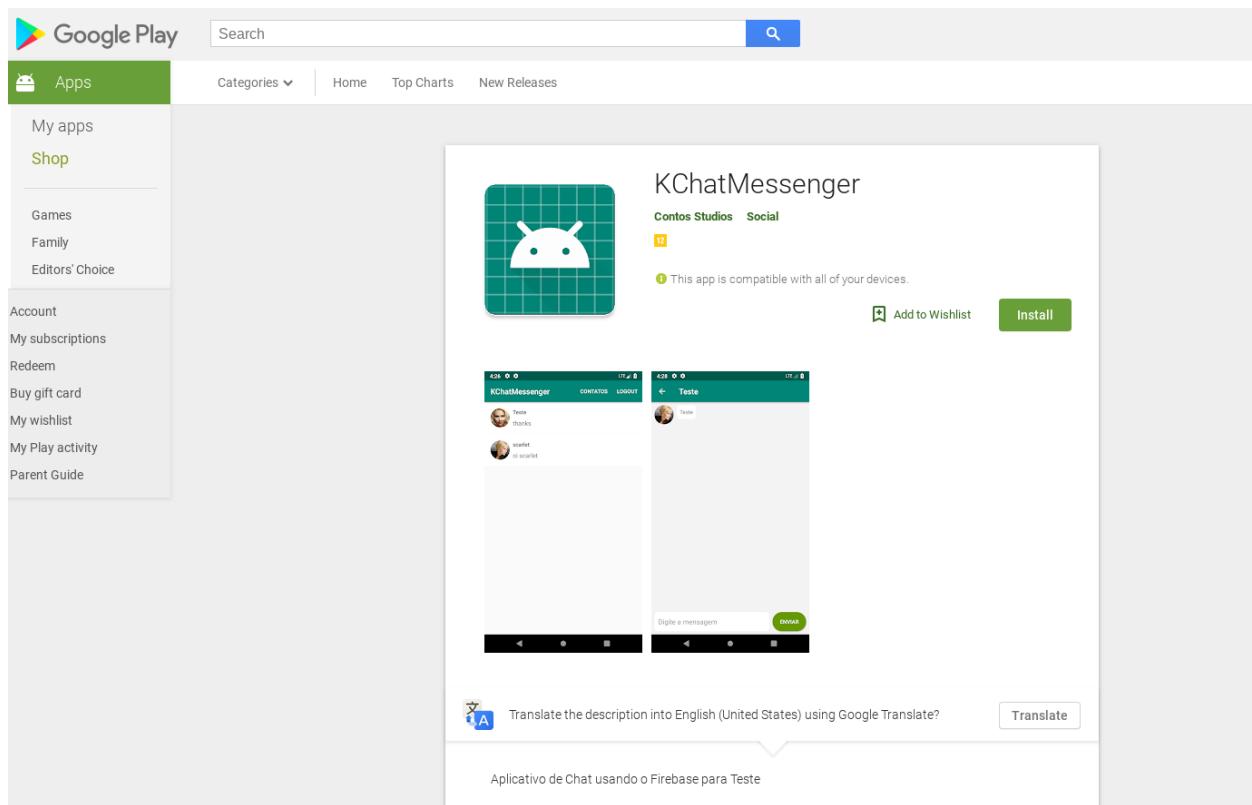
Default – Portuguese (Brazil) – pt-BR
Primeira versão,
1 language translation

PREVIOUS DISCARD START ROLLOUT TO PRODUCTION

Dentro de algumas horas o seu aplicativo será publicado no Google Play para todos baixarem.

KchatMessenger
co.tiagoaguiar.kchatmessenger

– ★ – Nov 21, 2018 Pending publication



13 Conclusão

Você acaba de dar um grande passo à frente rumo a sua carreira de desenvolvedor Android profissional. De fato ainda há muito a se estudar, aprender e por em prática, mas por hora você acaba de fazer um ótimo trabalho. Parabéns ;)

Muitos não chegam até o final de um processo de estudo e aprendizado e se você leu todo esse material e chegou até aqui é um vitorioso.

Agora continue sua jornada com outros tutoriais um pouco mais avançados que distribuo no Youtube e no meu site e acesse os links úteis logo abaixo.

No mais, fique ligado no seu e-mail que eventualmente disponibilizarei conteúdos exclusivos por lá. Este será um dos principais canais de comunicação.

Se tiver dúvidas, sugestões ou críticas sobre o material e/ou outro conteúdo que disponibilizo, acesse o site: <http://tiagoaguiar.co/contato>

Eu vou ficando por aqui.

Um forte abraço ;)

13.1 Links Úteis

- [Primeiro App](#)
- [Chat Messenger Kotlin](#)
- [Playlist em Java](#)
- [Curso Iniciante Android](#)
- [Curso Completo e Avançado](#)
- [Website](#)
- [Youtube](#)
- [Facebook](#)
- [Instagram](#)