



# Desenvolvendo Aplicações Mobile com Android Studio

---



fundaçāo bradesco | escola virtual

# Sumário

Apresentação	4
Módulo 1	6
Aplicativos <i>mobile</i> e plataformas de desenvolvimento	6
Aplicativos <i>mobile</i> e plataformas de desenvolvimento	7
Conceito e mercado de aplicações mobile	7
As principais plataformas de desenvolvimento mobile	9
Instalação e conimagemção do Android Studio	14
Módulo 2	28
Criando uma aplicação <i>mobile</i> e <i>activity</i>	28
Criando uma aplicação <i>mobile</i> e <i>activity</i>	29
Ferramentas de interface do Android Studio	29
Criando um projeto	32
Teste da aplicação	37
Interface gráfica ou do usuário e tipos de atividades	39
Conceitos de <i>View</i> e <i>ViewGroup</i>	41
Widgets	47
Os principais tipos de layouts para a aplicação no Android	56

# Sumário

Módulo 3	74
Requisições HTTP	74
Requisições HTTP	76
Carregando arquivos externos	76
Banco de dados <i>SQLite</i>	89
Publicação do aplicativo no Google Play Store	94
Distribuição via Google Play	102
Upload do APK	105
Fechamento	107
Referências	108

# Apresentação

Olá!

Bem-vindo(a) ao curso **Desenvolvendo Aplicações *Mobile* com Android Studio**.

O objetivo do curso é auxiliar no processo de criação de aplicativos utilizando a plataforma Android Studio, incluindo as boas práticas de programação na criação de layouts interativos (a partir de banco de dados *SQLite*) e protocolos de comunicação web, direcionados para uma melhor experiência do usuário.

Desejamos a você um bom curso!



## Vídeo

Confira o [vídeo](#) de apresentação do curso.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

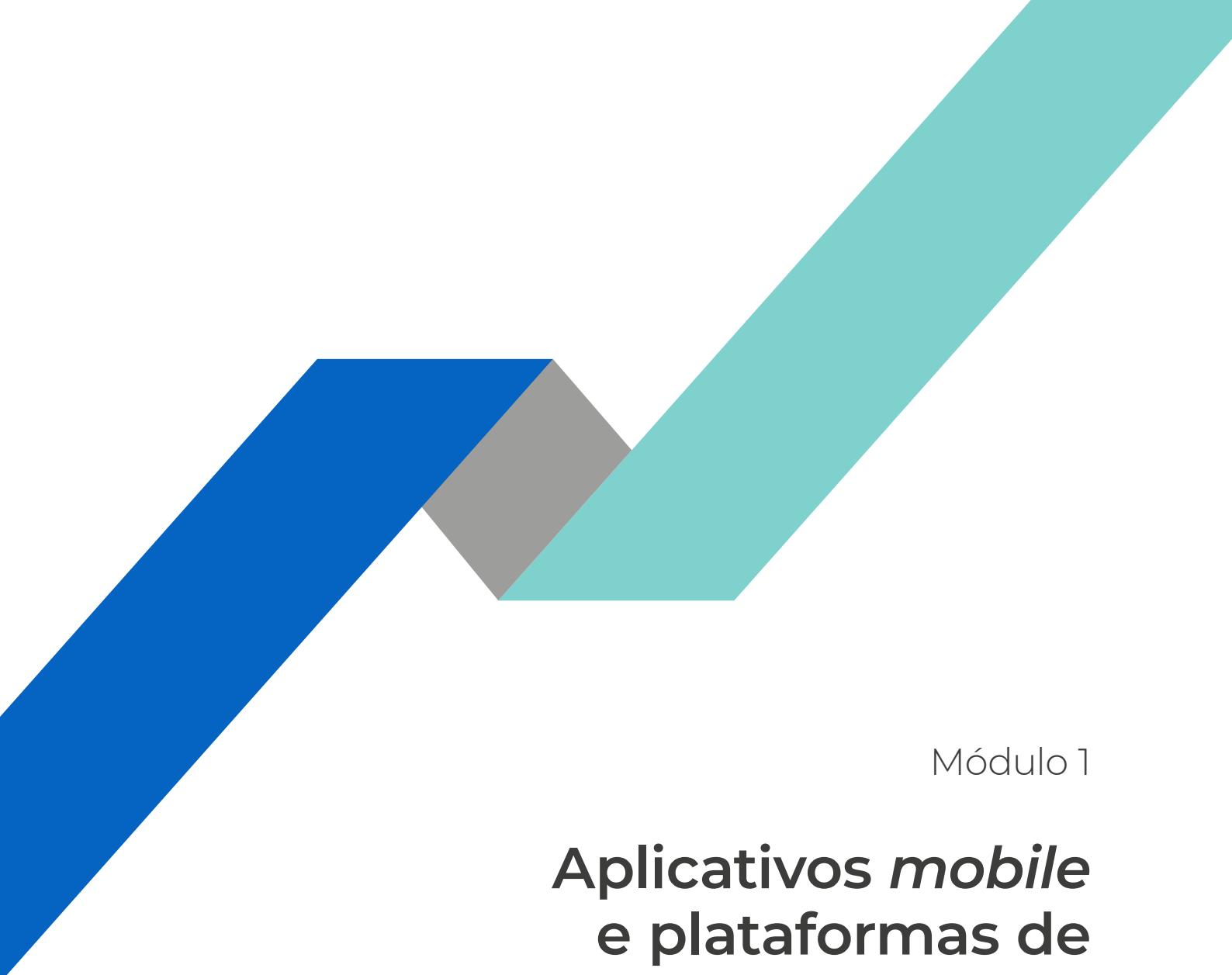
Olá!

Bem-vindo(a) ao curso **Desenvolvendo Aplicações *Mobile* com Android Studio**.

Aqui você entenderá um pouco mais a respeito das aplicações *mobile* e plataformas disponíveis atualmente para seu desenvolvimento. Você aprenderá também sobre o conceito de desenvolvimento *mobile* de modo mais aprofundado, bem como o mercado do segmento e o Android Studio, o qual será apresentado e utilizado na prática, para sua melhor compreensão.

Além disso, você identificará as ferramentas do Android Studio, como ocorre a criação de um projeto no ambiente, quais são os principais layouts para aplicações Android e o que são as requisições HTTP, incluindo carregamento de arquivos externos e publicação do aplicativo criado.

Vamos começar essa jornada?



Módulo 1

# Aplicativos *mobile* e plataformas de desenvolvimento

---

# Aplicativos *mobile* e plataformas de desenvolvimento

Neste primeiro módulo, você estudará sobre os conceitos de mercado *mobile* e as principais plataformas de desenvolvimento. Além disso, você conhecerá o Android Studio, um ambiente de criação do sistema operacional Android. Assim, você realizará a instalação e conimgagem do programa, bem como a criação dos emuladores para testar os aplicativos. Vamos lá!

## Conceito e mercado de aplicações *mobile*

O universo de aplicações *mobile* (softwares desenvolvidos e desenhados para o contexto de smartphones, tablets, smartvs e similares) está em franca expansão. Só que, para dar os primeiros passos nesse universo, é importante você entender o conceito e o mercado de aplicações *mobile*.

Desta forma, para dar início aos seus estudos, que tal começar com um *podcast*? Nele, você poderá acompanhar com mais detalhes o conceito *mobile* e seu mercado!



### **Podcast**

Confira o [podcast](#) sobre conceito e mercado de aplicações *mobile*.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

O mercado *mobile* se encontra em crescimento, buscando um amadurecimento mais próximo às necessidades de cada usuário ou cliente. A quantidade de aplicativos criados diariamente tem aumentado de forma exponencial, o que demonstra ser algo sem retorno.

Com isso, perceba que cresce também a necessidade de profissionais qualificados em uma demanda menos significativa, pois a busca por esses indivíduos ganha certo destaque, e o investimento na carreira acaba se tornando necessário.

Além disso, as tecnologias estão cada vez mais presentes em tarefas simples do dia a dia. Isso faz com que as organizações busquem por profissionais qualificados, que saibam resolver problemas de forma criativa, fazendo uso das ferramentas tecnológicas apresentadas a todo instante.

Dante desse cenário, com toda a expansão do mercado *mobile*, percebe-se como é necessário conhecer as plataformas de desenvolvimento relacionadas a esse contexto, uma vez que elas são utilizadas para a criação de aplicativos.

Vamos conferir as principais plataformas existentes?

Agora que você já tem uma base do que é o mercado *mobile* e como ele vem crescendo nos últimos tempos, veja na sequência as principais plataformas de desenvolvimento da área, sobretudo, as que envolvem o ambiente híbrido e o ambiente nativo, bem como a relação delas com as demandas exigidas pelos seus futuros clientes. Continue a leitura!

## As principais plataformas de desenvolvimento mobile

Conforme você viu anteriormente, o mercado de aplicações *mobile* está crescendo vertiginosamente, necessitando a cada dia mais de novos profissionais para esse fim. Para isso, é relevante que conheça e domine as plataformas de desenvolvimento, principalmente, do ambiente híbrido e nativo, pois cada uma será mais adequada, dependendo do projeto solicitado pelos futuros clientes. Sabendo disso, vamos lá.

Os aplicativos são desenvolvidos com o uso de ambientes multiplataformas. Tais ambientes são divididos em híbrido e nativo. Aliás, você saberia explicar o que cada ambiente traz e o que os diferencia? Ouça o *podcast* a seguir para compreender e saber como responder ao questionamento!



### ***Podcast***

Confira o [podcast](#) sobre as principais plataformas de desenvolvimento *mobile*.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Até em seus estudos, você viu o conceito *mobile* e como ele está atuante no mercado, e identificou como esse mercado está associado às plataformas de desenvolvimento. Com isso, você verificou como é importante conhecer essas plataformas e os ambientes que elas oferecem para o desenvolvimento de aplicativos. Agora, vamos conhecer com mais detalhes sobre o ambiente híbrido e nativo. Acompanhe!

No **ambiente híbrido**, o desenvolvedor não cria sua codificação diretamente na linguagem específica do sistema operacional do aparelho, independentemente de ser um smartphone, um tablet, um relógio ou qualquer outro aparelho. Ele utiliza software de terceiros, como Flutter com Dart, React Native com JX, Ionic com HTML, CSS e JS, os quais são frameworks capazes de criar aplicativos para os dois ambientes tanto Android como o iOS, considerando as linguagens de seus respectivos compiladores.

Por outro lado, no **ambiente nativo**, o desenvolvedor cria sua codificação diretamente no sistema operacional do aparelho, como Android e iOS. Nesse caso, utiliza-se o software nativo do ambiente, como o Android Studio com Java ou Kotlin para devices que fazem uso do sistema operacional Android, e Swift para o sistema operacional iOS, da Apple.

Viu só como cada ambiente se diferencia e possui suas especificidades? Por isso, é necessário conhecer as necessidades de cada cliente para a criar o aplicativo.

Para criação de aplicativos, seja qual for a ideia de projeto, é preciso considerar as necessidades dos clientes, bem como analisar se o que desejamos criar será útil para o usuário. Esse diferencial é extremamente importante no processo de desenvolvimento de qualquer tipo de aplicativo.

## Saiba mais



Em relação às plataformas híbrida ou nativa, deve-se considerar o projeto que será realizado, a equipe para o seu desenvolvimento e o produto. Dessa forma, será fundamental ter uma reunião com a equipe de projetos e organizar todos os processos, analisando quais serão as ferramentas necessárias e o que se deseja obter como produto final para o mercado. Para saber mais sobre essas plataformas, sugerimos que assista ao vídeo [Aplicativo Híbrido ou Nativo?](#), pois complementará seus estudos e o ajudará em seu dia a dia.

Neste tópico, você entendeu o conceito de plataformas de desenvolvimento *mobile*, as quais são divididas em ambiente híbrido e ambiente nativo, além de que, independente de qual for usar, deve-se conhecer o projeto do cliente, para que o processo de planejamento e execução do que foi solicitado seja mais eficiente.

Neste curso, vamos nos aprofundar no ambiente nativo. Dessa forma, utilizaremos a plataforma de desenvolvimento Android Studio, que disponibiliza ao desenvolvedor uma gama de ferramentas e processos incríveis para a criação de aplicativos para smartphones, tablets, relógios, TV e internet das coisas.

Sabendo disso, vamos ver mais detalhadamente sobre o Android Studio.

## Android Studio

De acordo com o que você estudou até aqui, existem dois tipos de ambientes nas plataformas de desenvolvimento: o híbrido e o nativo. Vamos focar no segundo tipo, principalmente, na plataforma Android Studio. Isso porque além da sua versatilidade, o sistema operacional Android, no cenário mobile, é o mais utilizado no mundo. Portanto, muitos projetos e demandas são direcionados a ele. Em função disso, vamos começar pela sua definição.



## Entenda o Conceito

O Android Studio é um sistema operacional livre para a criação de aplicativos móveis, o qual foi desenvolvido pelo Google e seus associados.



O Android Studio tem uma interface simples, robusta e intuitiva, desenvolvida para auxiliar no processo de construção de aplicativos móveis voltados às boas práticas de design e programação.

#PraCegoVer

Na imagem, com fundo claro e desfocado, estão as mãos de uma pessoa. Uma mão segura o celular e a outra está digitando.

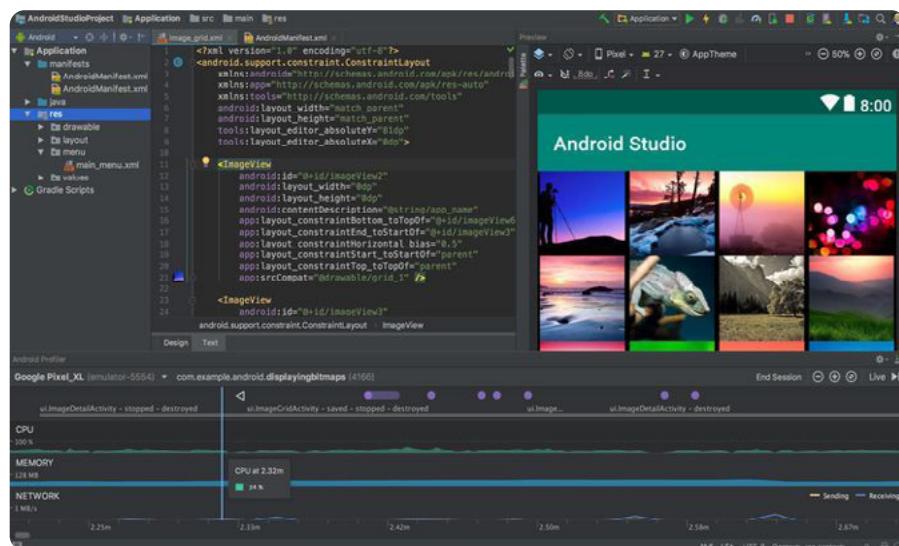
Seu sistema é utilizado como base para a construção dos layouts, materiais de design e práticas de UX (*user experience* ou experiência do usuário).



#PraCegoVer

Na imagem, um homem com camisa branca está sentado em uma mesa de escritório. Não é possível ver seu rosto, mas ele está em frente ao notebook aberto, com um braço apoiado na mesa e o outro segura um celular.

De fato, um bom aplicativo será aquele que apresenta uma boa interface, relacionada ao fluxo adequado de janelas e informações para o usuário. Na imagem a seguir, podemos observar esses pontos no ambiente de desenvolvimento do Android Studio.



#PraCegoVer: na imagem, temos um *print* do ambiente de desenvolvimento Android Studio. Trata-se de uma tela escura dividida em quatro partes: na coluna da esquerda, existe o *dashboard*, ao lado dela mais ao centro, o painel com as linhas de código, à direita, um painel com imagens. Na parte inferior, existe um painel visor com informações de desempenho do aplicativo a ser criado, incluindo código, visualização e outros menus.

A criação de layouts no Android Studio demanda planejamento, prototipação e análise de fontes, cores, imagens, entre outras particularidades. Para isso, foram criadas especificações e métodos que auxiliam no posicionamento e nos estilos com diversas possibilidades.

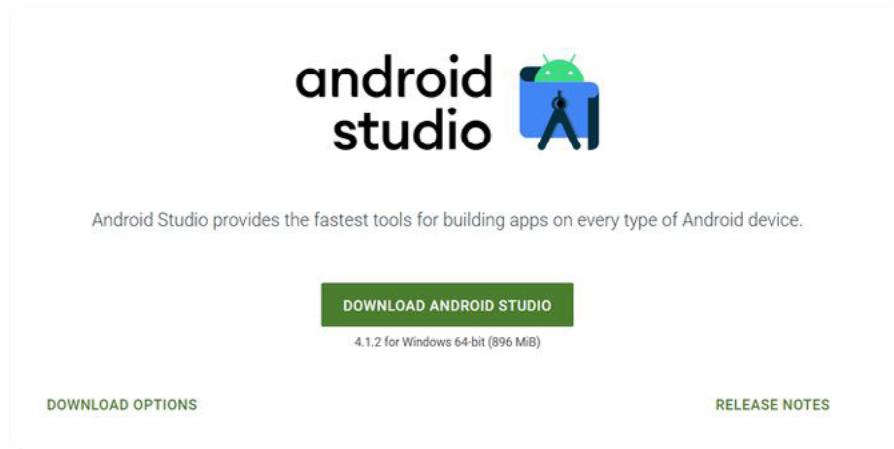
Desta forma, você pôde conhecer as principais características do Android Studio, bem como ver um *print* do ambiente de desenvolvimento dele.

No entanto, para que você entenda melhor sobre a temática e estar mais preparado para os próximos assuntos, é necessário realizar a instalação e conimagemão do Android Studio. Veja a seguir!

## Instalação e conimagemção do Android Studio

Após você conhecer mais as características do Android Studio, além de já ter uma ideia do ambiente de desenvolvimento dele, chegou a hora de iniciar a transição da teoria para prática. O primeiro estágio desse processo é aprender sobre a instalação e conimagemção do Android Studio.

Para começar a desenvolver os aplicativos no Android Studio, é necessário instalar o *Software Development Kit* (SDK), também conhecido no português como kit de desenvolvimento de software. O Android SDK traz um conjunto de ferramentas-base e emuladores para realizar a codificação e os testes da aplicação.



#PraCegoVer: na imagem, temos um *print* da janela de download do Android Studio para Windows. Nela, aparece a mascote do ambiente e um botão em verde para download mais abaixo.

Para baixá-lo, acesse o [Android SDK](#) e, após realizar a leitura dos termos e as condições apresentadas, selecione o “aceite”. Na sequência, a versão mais recente do programa será disponibilizada, seguindo o sistema operacional. Caso o processador ou sistema seja diferente do Android, há outras versões e plataformas, as quais estão disponíveis no [Android Studio Downloads](#).

A seguir, você pode conferir todos os requisitos mínimos para cada plataforma.

Windows	Mac	Linux	Chrome OS
- 64-bit Microsoft® Windows® 8/10 - x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor - 8 GB RAM or more - 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) - 1280 x 800 minimum screen resolution	- MacOS® 10.14 (Mojave) or higher - ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework - 8 GB RAM or more - 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) - 1280 x 800 minimum screen resolution	- Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later. - x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3 - 8 GB RAM or more - 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator) - 1280 x 800 minimum screen resolution	For information on recommended devices and specifications, as well as Android Emulator support, visit <a href="https://chromeos.dev">chromeos.dev</a> .

Após realizar o download do Android SDK no sistema operacional, inicie a instalação. O processo completo está disponível no item “[Instalar o Android Studio](#)”, basta seguir os passos oferecidos para o sistema operacional desejado.

Com a instalação do ambiente, podemos seguir para a sua conimagemção.

Assista ao vídeo que preparamos para que possa entender os próximos passos!



## Vídeo

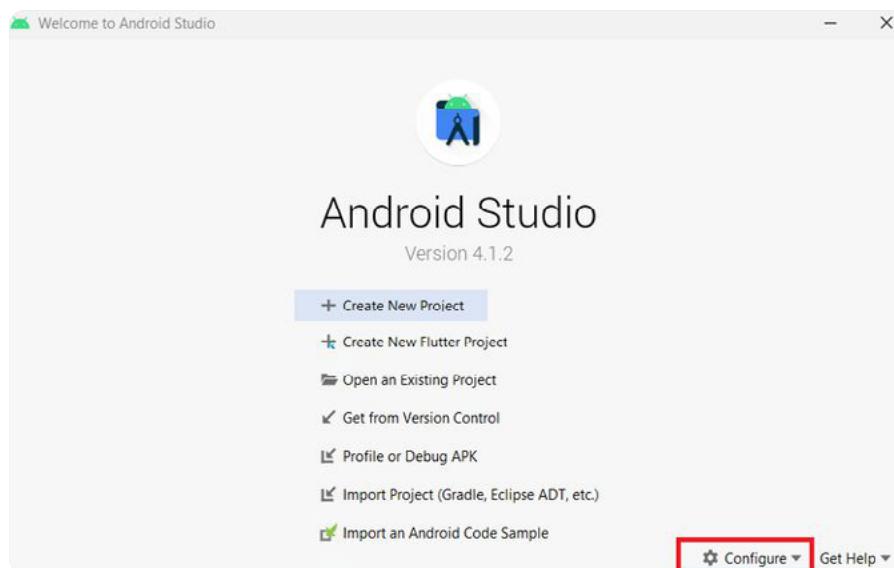
Confira o [vídeo](#) sobre instalação e conimagemção do Android Studio

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Agora que você já instalou o Android Studio, é preciso configurá-lo, antes de iniciar o projeto. Para isso, inicie o Android Studio.

Lembre-se! Independentemente da versão baixada, a base da janela será a mesma.

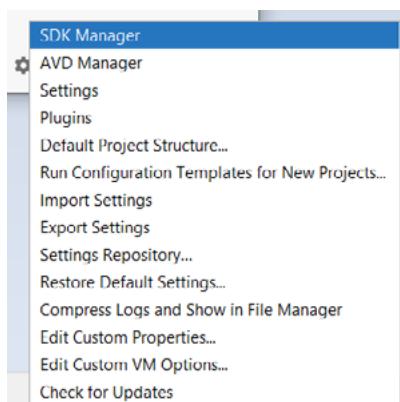
Observe que o botão de configuração está logo abaixo, no canto inferior à direita da tela.



#PraCegoVer: na imagem, temos um print da janela principal do Android Studio para Windows. No topo, há o logo do ambiente. Abaixo dele, existem as opções de interação do app. No canto inferior direito da tela, existe um ícone em que há um trecho destacado em vermelho escrito "configure"..

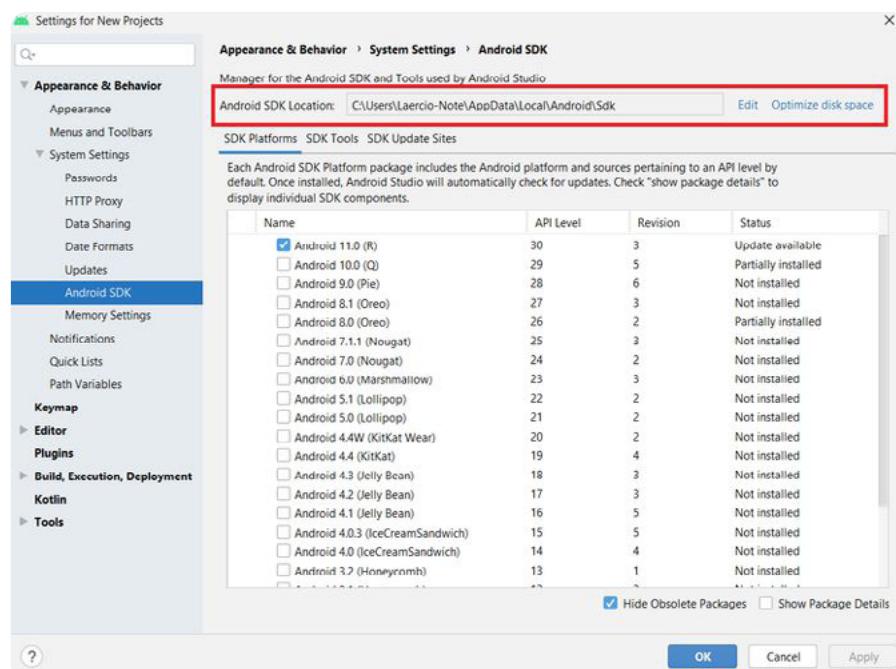
Ao selecioná-lo, prossiga com as seguintes configurações principais:

1. Configure o SDK clicando em “Configure”.
2. Selecione a opção “SDK Manager”.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existem as opções disponíveis de conimagemção do Android Studio, dentre elas, a opção "SDK Manager".

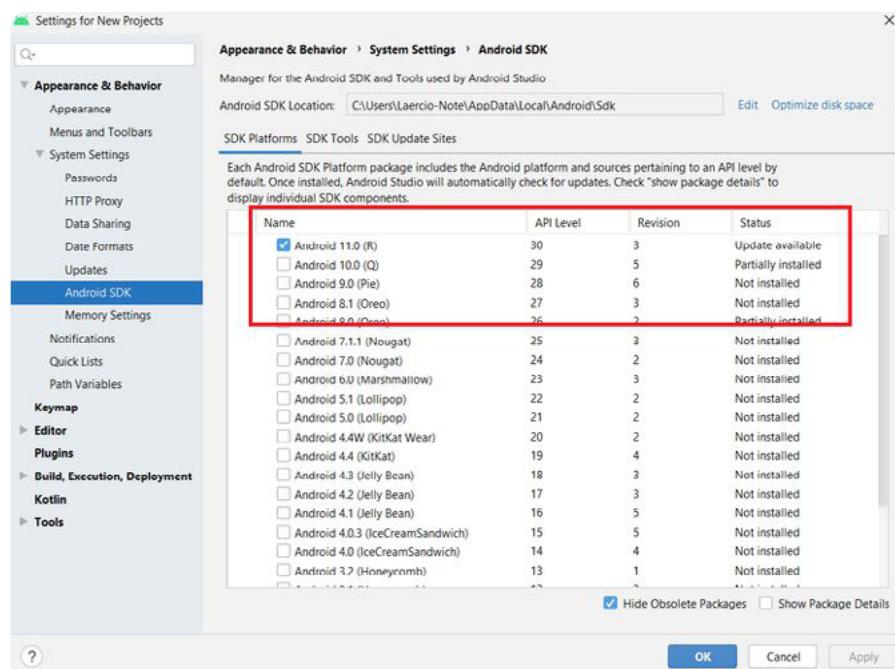
Com esses passos seguidos corretamente, será aberto o Android SDK Location, em que aparecerá a instalação do kit de desenvolvimento Android, necessário para criar os aplicativos.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um dashboard à esquerda e, no meio, existem as opções de conimagemção do SDK Location. Nesse local, há um trecho destacado em vermelho escrito "Android SDK Location".

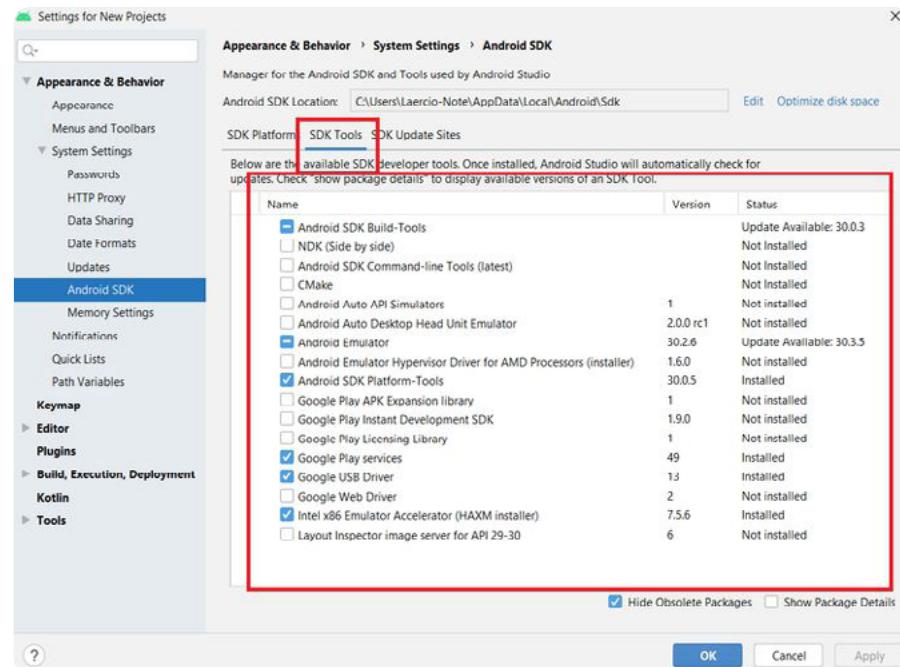
Um ponto importante é que as APIs já estão disponíveis nos aparelhos que serão testados, visto que, por padrão, o Android Studio sempre instala a última versão. No entanto, esse processo é um pouco demorado, dependendo do hardware e da rede de internet.

Assim, é aconselhável manter a versão que foi instalada e, em último recurso, instalar outras.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um dashboard à esquerda e, no meio, existem as opções de configuração do SDK Location. Nesse local, há um trecho destacado em vermelho onde estão as últimas versões do Android.

Outro item importante na conimagemção consiste no SDK Tools, que diz respeito às ferramentas utilizadas pelo Android Studio para emular, os recursos de virtualização de processador, ao Google Play e outras. Diante disso, manter esses itens selecionados é de suma importância para o desenvolvimento do trabalho.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um dashboard à esquerda e, no meio, existem as opções de conimagemção do SDK Location. Nesse local, há dois trechos destacados em vermelho. O maior ocupa o trecho central da tela onde estão os SDK Tools selecionados. O menor está acima dele onde está escrito "SDK Tools".

Em relação ao suporte de aceleração para processadores Intel, verifique se o computador tem o suporte para virtualização. Nas atualizações de versões do SDK do Android Studio, os desenvolvedores são informados que devem ter o sistema sempre atualizado.

## Saiba mais



No artigo [Como Posso Habilitar a Virtualização \(VT\) no meu PC?](#), há um passo a passo para que você consiga habilitar o suporte referente à virtualização em seu computador ou notebook, a fim de aplicar os exercícios práticos que vamos realizar ao longo de todo o curso. Caso ainda não saiba como fazer, vale a pena conferir!

Finalizada a etapa de conimagemção do SDK do Android Studio, é necessário conimagemr o emulador, que é a ferramenta indispensável para testar os aplicativos em tempo de execução. Ou seja, os testes serão realizados no ambiente que o usuário final utilizará. Para tanto, deve-se conimagemr o *AVD Manager (Android Virtual Device)* nesse ambiente, com atenção especial ao que depende exclusivamente de hardware.

Assista a mais um vídeo para entender o assunto e colocar as ideias em prática!



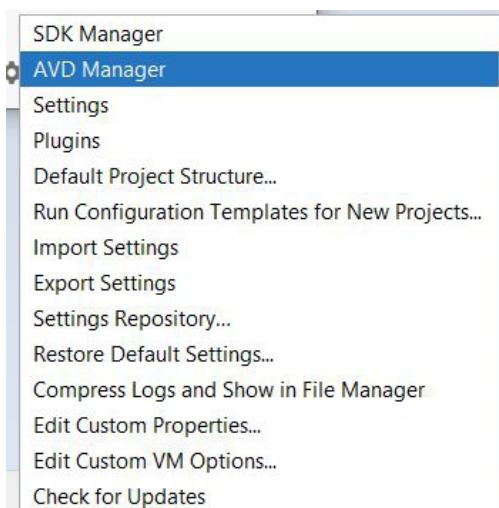
## Vídeo

Confira o [vídeo](#) sobre instalação e conimagemção do *Android Virtual Device*.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Agora que já configurou o Android Studio, chegou o momento de testar o aplicativo, mas antes é necessário conimagemr o emulador. Confira como fazer isso agora.

Na tela inicial do Android, vá em “Configure” e selecione a opção disponível “AVD Manager”.



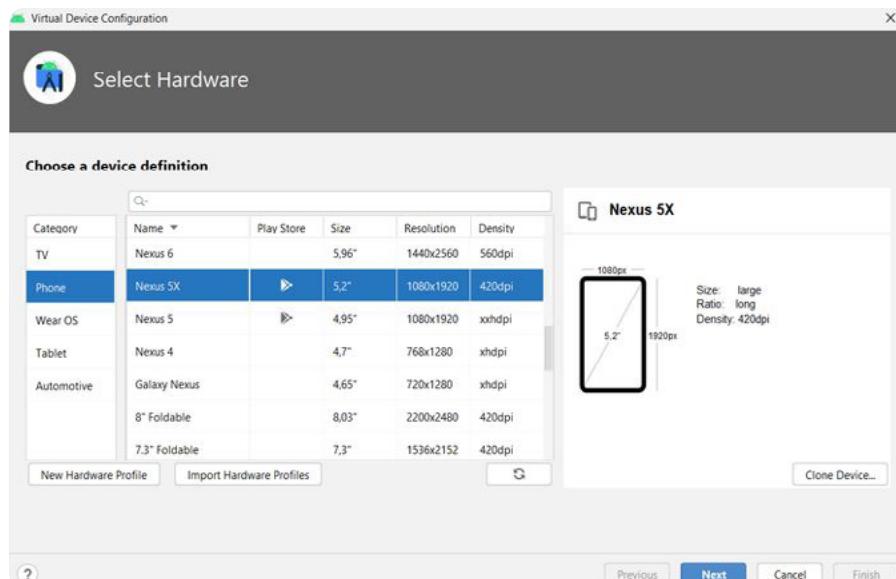
#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existem as opções disponíveis de conimagemção do Android Studio, dentre elas, a opção “AVD Manager”.

Como é a primeira vez que vai conimagemr o device, aparece a seguinte tela. Nela, selecione “Create Virtual Device”. Que traz a seleção do device a ser utilizado.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior em que está escrito "Your Virtual Devices". Abaixo dele, existe um trecho explicativo sobre a escolha do device, na parte intermediária, existem cinco ícones que representam, da esquerda para a direita, um celular e tablet, um relógio, uma tela de computador, um carro e um microchip. Abaixo, há um trecho destacado em vermelho com a opção escrita "Create Virtual Device".

Na tela de seleção, é possível definir a categoria do device, como optar pelo smartphone, por exemplo, e selecionar o tipo a ser escolhido: Phone do tipo Nexus 5X com Play Store de 5.2 polegadas, resolução de 1080x1920 e densidade de 420 pontos por polegada. Em seguida, clique em "Next".

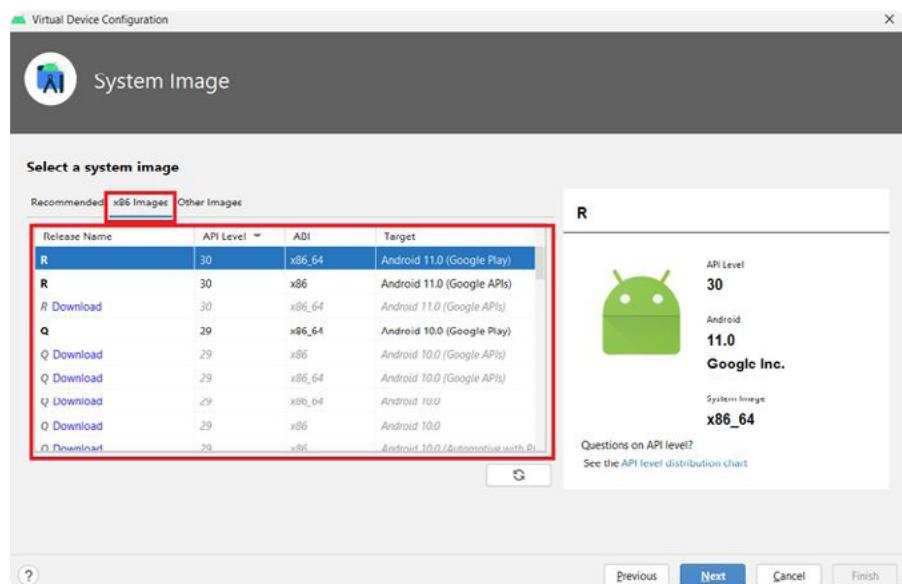


#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior no qual está escrito "select hardware". Abaixo dele, existem duas telas de conimagemção do device.

Agora, escolha a versão do sistema operacional Android a ser instalada no device. Nesse caso, cabe uma observação importante: para conimagemr corretamente o sistema no device, é preciso ignorar a aba de recomendado e clicar na aba de “x86 Images”.

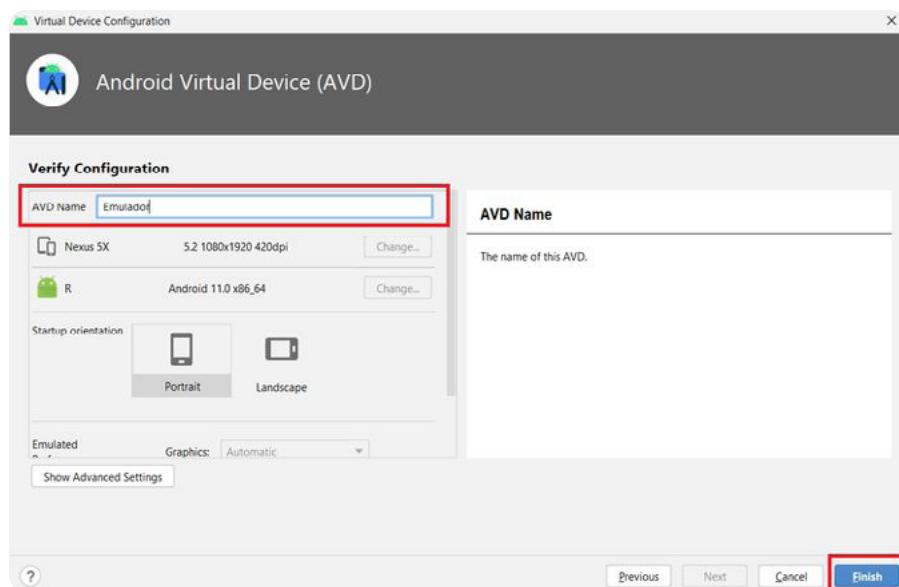
Em caso de não ser um processador Intel, é preciso ir por outro caminho, clicando em “Other Images”. Neste momento selecione o sistema que deseja utilizar para teste no device, que pode ser qualquer device.

Perceba que, alguns itens não estão escritos para download, enquanto outros estão. Como sugestão, opte pela primeira versão, em que o sistema será o Android R, com API Level de 30 e ABI de x86\_64. O sistema que irá rodar será o Android 11.0 (Google Play).



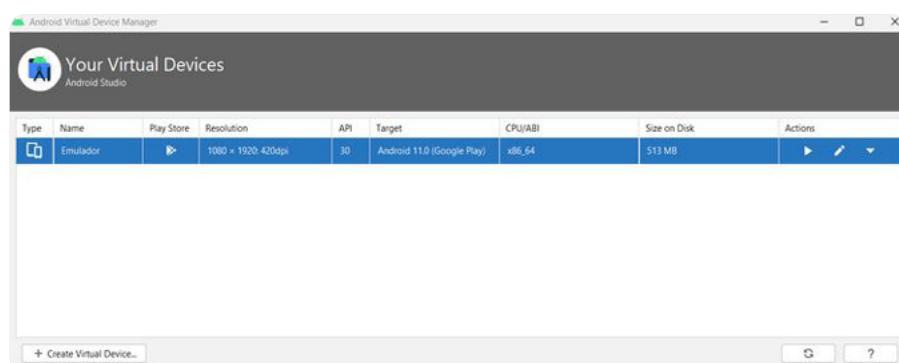
#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior no qual está escrito “System Image”. Abaixo dele, existem duas telas de conimagemção da imagem do sistema relacionado ao device.

Essa parte demora um pouco e dependerá do hardware e sua internet. Após o término, clique em “Next”. Agora, configure o nome do emulador. Depois, selecione “Finish”.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior no qual está escrito “Android Virtual Device (AVD)”. Abaixo dele, existem duas telas de conimagemção associadas à verificação da conimagemção.

Dessa maneira, será criado o emulador. A lista indicada apresenta o nome e as conimagemções realizadas anteriormente.



#PraCegoVer: na imagem, temos um print da janela do android Studio para Windows. Nela, existe um banner superior no qual está escrito “Your Virtual Devices”. Abaixo dele, há uma tela de conimagemção do device.

Pronto, agora é só executar.

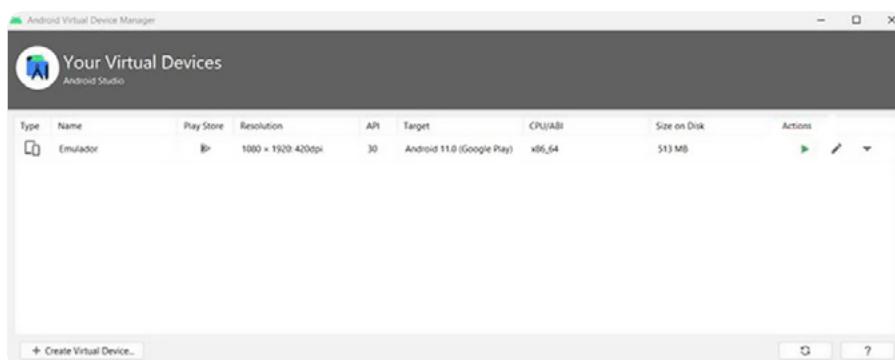
Vale ressaltar que, em “Actions”, na última tela que apresentamos no vídeo, é possível apagar, alterar, copiar e criar outros emuladores, conforme as necessidades de teste que surgirão. Todavia, a execução de mais de um emulador dependerá exclusivamente da capacidade de hardware.

## Saiba mais



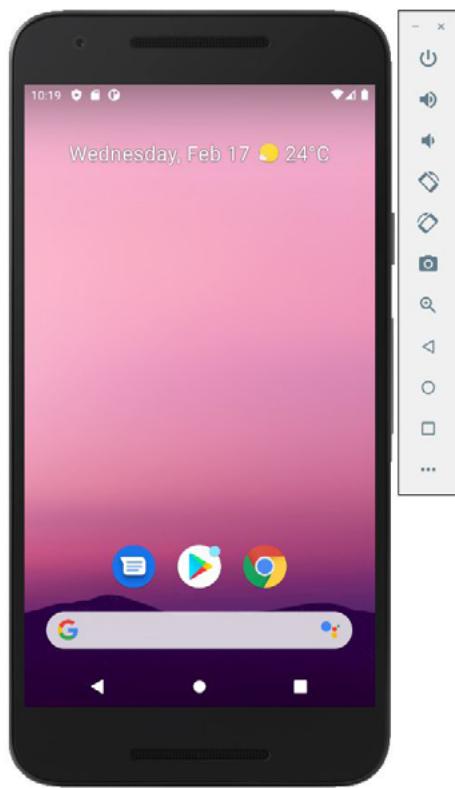
Para obter mais informações referentes ao suporte de virtualização, você pode ler o artigo [Conimagemr a Aceleração de Hardware para o Android Emulator](#), para conferir instruções mais avançadas a respeito do assunto. Vale conferir!

Aliás, vale mencionar que o emulador poderá ser executado até antes de criar um projeto, selecionando em “executar”, conforme você pode observar a seguir.



#PraCegoVer: na imagem, temos um print da execução do emulador, estando a opção “executar” em destaque à direita, na seção de “Actions”.

O emulador demora um pouco para ser carregado, mas logo disponibilizará as conimagemções de um smartphone. Depois desse procedimento, já estará pronto para testar no aplicativo que estamos elaborando juntos no curso.



#PraCegoVer: na imagem, temos a ilustração de um celular com o emulador em execução. Aparece uma tela inicial no aparelho, junto de seus botões e outras particularidades, como horário, nível de bateria e rede. Do lado direito, ainda há um menu vertical com opções.

Parabéns! Você chegou ao final do módulo 1.

Aqui você conheceu o conceito de aplicação *mobile* e viu sobre o contexto do mercado atual desse segmento, o qual está em grande expansão. Além disso, aprendeu um pouco sobre as plataformas de desenvolvimento (do ambiente híbrido e nativo) e a sua importância, principalmente, quando estiver produzindo para seus futuros clientes.

Em seguida, você foi apresentado ao Android Studio, plataforma de desenvolvimento do sistema operacional do Android, um dos mais populares e requisitados atualmente. Assim, você pôde entender as suas características e conhecer seu ambiente de desenvolvimento.

Você descobriu a maneira de instalá-lo e configurá-lo, lembrando que o primeiro estágio desse processo consistiu em fazer o download e instalar o *Software Development Kit* (ou kit de desenvolvimento de software em português).

Por fim, somado a isso, você observou o tutorial de como conimagemr o emulador, o recurso de testagem do *app*, enquanto você o desenvolve ou quando estiver finalizado.

No próximo módulo você terá uma proposta voltada para a prática, sobretudo, envolvendo a geração, a execução e a testagem de um aplicativo, fases essenciais para que você tenha capacidade de atuar no ramo da aplicação *mobile*. Vamos lá?



Módulo 2

## Criando uma aplicação *mobile e activity*

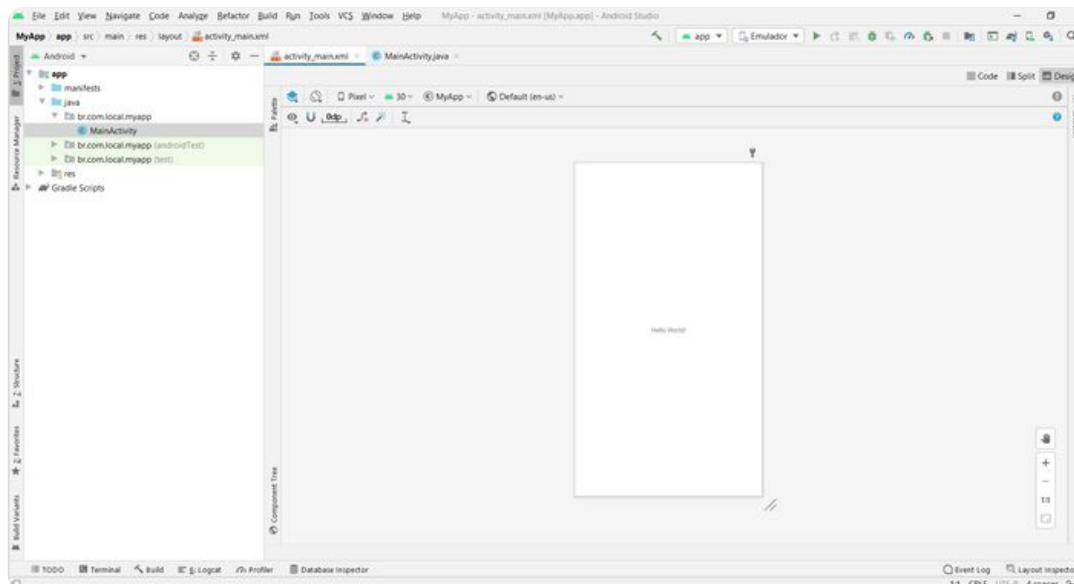
---

# Criando uma aplicação mobile e activity

Neste módulo, você estudará as principais ferramentas para criar, executar e testar o aplicativo. Além disso, conhecerá como é a interface de um aplicativo, quais são seus principais *widgets* ou componentes, assim como seus grupos e layouts. Veja a seguir!

## Ferramentas de interface do Android Studio

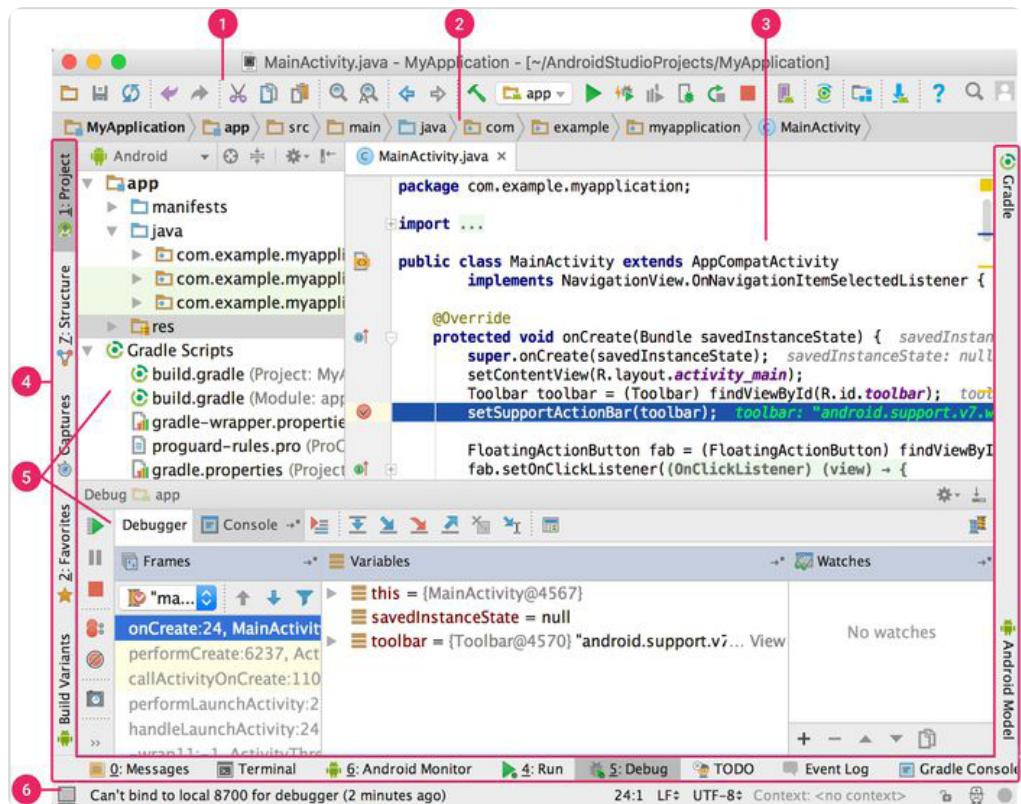
Após a instalação, conimgemção do ambiente e do emulador do Android Studio, que vimos no primeiro módulo, você já pode criar seu primeiro aplicativo. No entanto, antes de iniciar, é importante tomar nota quanto às suas principais janelas. Desse modo, na imagem abaixo apresentamos a interface do Android Studio, confira:



#PraCegoVer: na imagem, temos um print do ambiente de desenvolvimento ou interface do Android Studio. À esquerda, encontramos alguns repositórios. À direita, temos uma tela em branco e opções para personalização e criação.

Esse ambiente de desenvolvimento é separado para garantir a visualização da estrutura do projeto, das abas de codificação, do design e do teste do aplicativo.

Observe na imagem e observe cada parte dessa interface.



#PraCegoVer: na imagem, temos um *print* da interface do usuário com barra de ferramentas, barra de navegação, janela do editor, barra de janela de ferramentas, janelas de ferramentas e barra de status.

## 1- Barra de ferramentas

Permite que ações sejam realizadas, incluindo a execução de aplicativos e inicialização de ferramentas do Android.

## 2- Barra de navegação

Auxilia na navegação do projeto e na abertura de arquivos para edição. Destaca-se que essa barra oferece uma visualização mais compacta da estrutura na janela do projeto.

### 3- Janela do editor

É o espaço em que se cria e modifica o código, a depender do tipo de arquivo que está sendo utilizado. Por exemplo, em um arquivo de layout, o editor consegue abrir o editor de layout.

### 4- Barra de janela de ferramentas

Barra externa à janela do ambiente de desenvolvimento integrado. Traz opções que permitem expandir ou recolher as janelas de cada ferramenta.

### 5- Janelas de ferramentas

Possibilitam o acesso a tarefas específicas, como pesquisa e controle de versões ou gerenciamento de projetos, podendo ser recolhidas ou expandidas.

### 6- Barra de status

Exibe o status do projeto e ambiente de desenvolvimento integrado, incluindo avisos e mensagens relacionadas.

## Saiba mais



A interface do Android Studio pode ser organizada conforme as necessidades do desenvolvedor, mas é bom se familiarizar com as janelas para não haver equívocos ou trocas que podem afetar seu projeto. No artigo [Conheça o Android Studio](#), você encontra informações valiosas a respeito da temática!

Depois de se familiarizar com a interface primária dessa plataforma, você aprenderá como criar seu primeiro projeto com tranquilidade!

## Criando um projeto

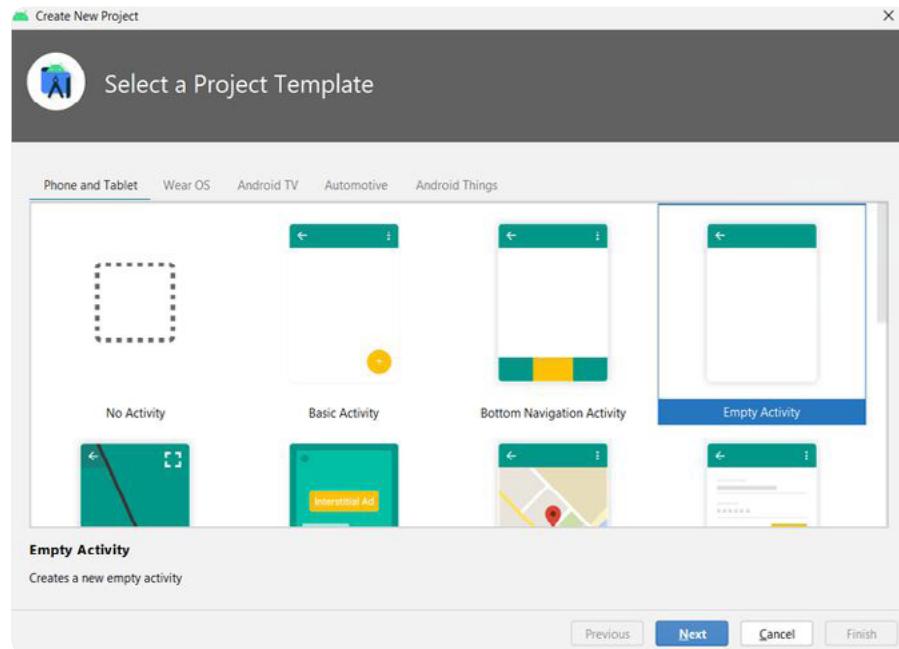
Para criar um projeto, o primeiro passo é acessar a janela principal do Android Studio. Selecione “*Create New Project*”, conforme demonstra a imagem abaixo.



#PraCegoVer: na imagem, temos um *print* da janela para seleção de “Create New Project”. Há o mascote do Android Studio em cima e algumas opções listadas abaixo, uma embaixo da outra.

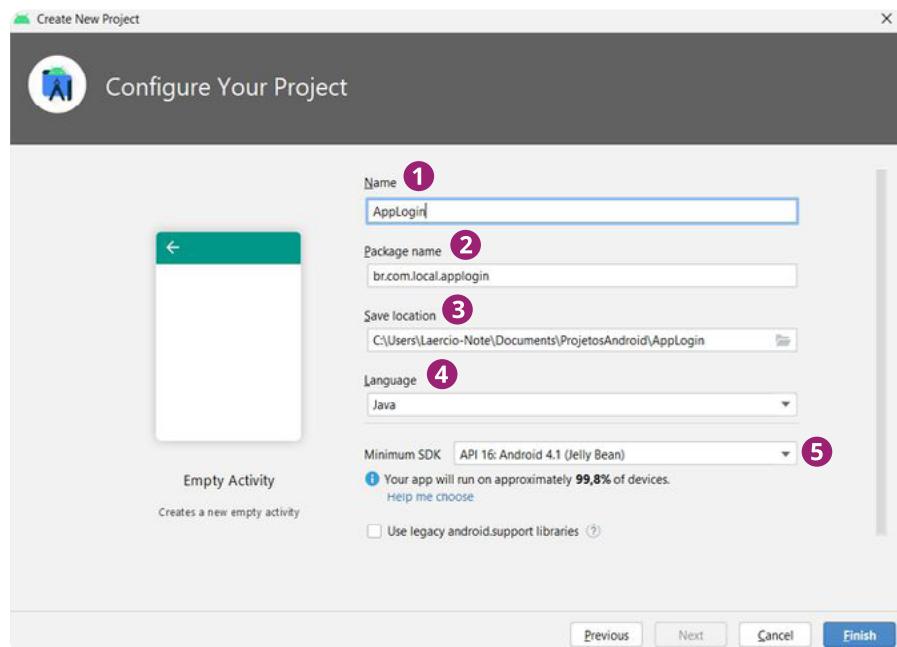
O Android Studio disponibiliza uma série de *templates*, os quais são janelas pré-construídas para auxiliar o desenvolvedor no processo de criação de layouts, com diversos componentes definidos. Dessa forma, utilize um *template* vazio com as configurações básicas para o projeto.

Selecione “*Empty Activity*” e em seguida “*Next*”.



#PraCegoVer: na imagem, temos um *print* do Android Studio na área de "Create New Project". Há alguns modelos de templates para escolha, estando selecionado o "Empty Activity".

A seguir, devem ser conimagemdos os nomes de projeto e pacote, o local de salvamento, a linguagem utilizada e o SDK mínimo para o projeto. Em resumo, observe na imagem abaixo para entender melhor cada conimagemção:



#PraCegoVer: na imagem, temos um print do *Android Studio* na área de conimagemção do projeto. O *template* escolhido está selecionado à esquerda, com campos para preenchimento à direita, como nome do projeto, local de salvamento etc.

## 1- Nome do projeto

Identificará o aplicativo no *device* que estamos rodando, por isso, é importante nomear conforme aquilo que você vai realizar, por exemplo, iFood, Uber, entre outros.

## 2- Nome do pacote

É uma definição um pouco mais elevada, como se o projeto pertencesse a um site, por exemplo: www.ifood.com.br. Para o nome do pacote, coloque, nesse caso, "br.com.ifood.LojaApp".

### 3- Local de salvamento

Essa parte é de extrema importância, pois é necessário inserir o projeto em um local que saibamos como recuperá-lo posteriormente.

### 4- Linguagem

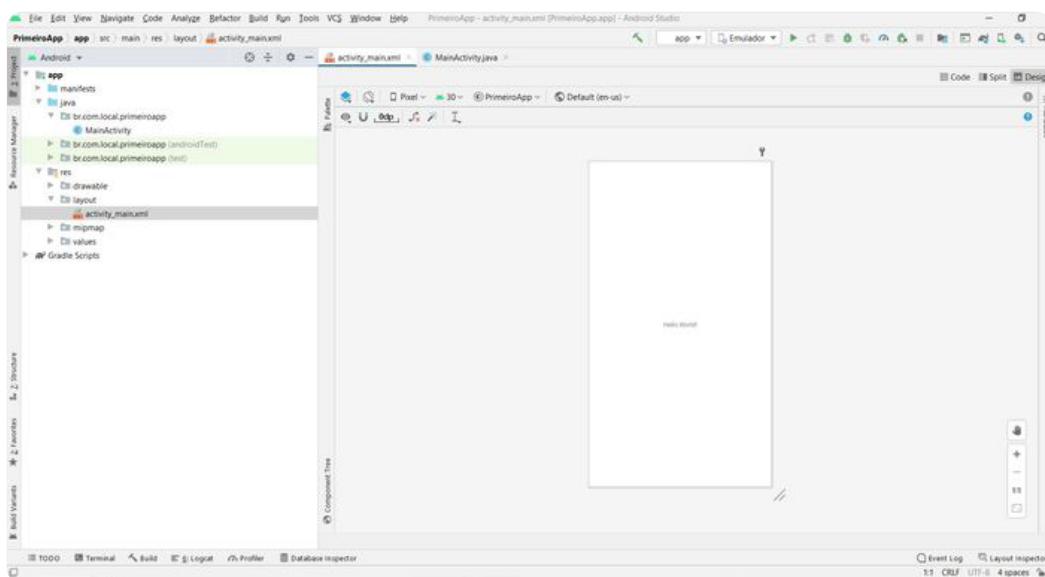
Define qual tipo de programação será utilizado. No caso, contamos com Java e Kotlin. Para o curso, utilizaremos a linguagem Java.

### 5- SDK mínimo

Define qual kit de desenvolvimento será utilizado, caso necessite de um *device* que não suporte a API conimagemda quando instalou o ambiente. Se tiver um aparelho muito antigo, o Android utilizará a API mínima escolhida. Uma dica é manter a API informada pelo próprio sistema.

Uma observação importante é que essas conimagemções podem ser alteradas no projeto, caso seja necessário.

Realizadas as conimagemções do projeto, selecione “*Finish*”. Na sequência, o Android Studio construirá o projeto, esse procedimento pode demorar um pouco. A dica é aguardar a construção total, para depois iniciar a programação.



#PraCegoVer: na imagem, temos um print do Android Studio com a tela do aplicativo pronta. À esquerda, encontramos alguns repositórios. À direita, temos uma tela em branco e opções para personalização e criação.

Com o projeto pronto, este já pode ser executado, sem qualquer codificação.

## Saiba mais



No artigo [Criar um Projeto para Android](#), há um compilado de informações interessantes para você se aprofundar na temática. Dessa maneira, será muito mais fácil criar o aplicativo. Não deixe de ler as dicas!

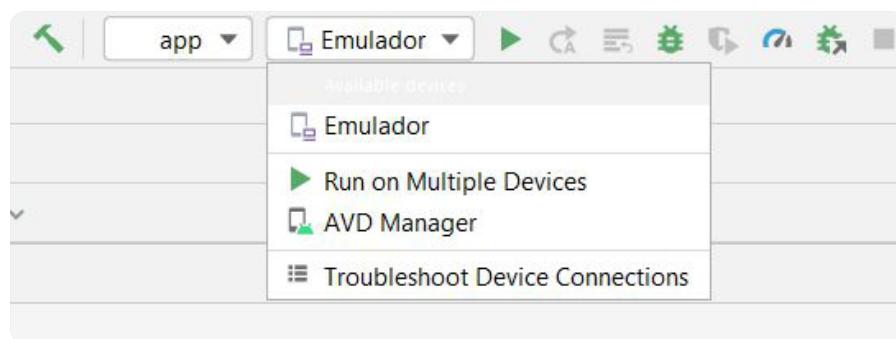
Até este ponto, você pôde entender como iniciar o projeto de criação de um aplicativo. Notou, também, que a plataforma disponibiliza *templates* que você poderá usar, dependendo do estilo e proposta do seu projeto.

Em seguida, viu como realizar as configurações mínimas para o seu projeto.

Portanto, como o projeto foi criado e o ambiente já está configurado, chegou a hora de testar a aplicação. Faremos isso juntos no próximo tópico!

## Teste da aplicação

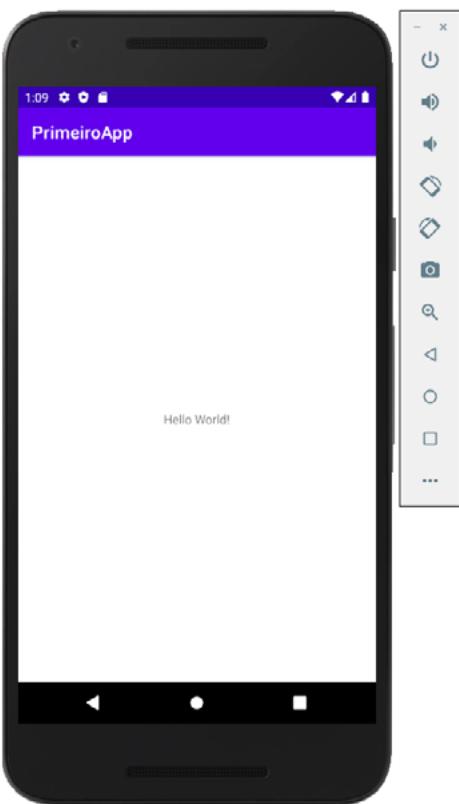
Após começar a criar o projeto do aplicativo, vamos para a etapa da testagem. O processo de teste da aplicação demanda saber em qual tipo de *device* será testado, que poderá ser em smartphone, um relógio, uma televisão ou a internet das coisas. No curso, utilizaremos o emulador criado no primeiro módulo.



#PraCegoVer: na imagem, temos um *print* com destaque para um menu horizontal e a opção de emulador, juntamente com suas alternativas.

Para selecionar o emulador, clique no nome ou na seta verde e, depois, aguardar a execução, conforme indicado.

Desta forma, ao acionar o emulador, você estará diante da seguinte tela:



#PraCegoVer: na imagem, temos a ilustração de um celular com o primeiro App em execução no emulador. Aparece uma tela inicial no aparelho, junto de seus botões e outras particularidades, como horário, nível de bateria e rede. Do lado direito, ainda há um menu vertical com opções.

Considerando a imagem anterior, perceba na janela apresentada que ela traz as mesmas características de um aplicativo rodando em seu próprio aparelho.

Assim, pensando nisso, é muito importante realizar os testes da aplicação no emulador para evitar qualquer problema de componentes na interface.

## Saiba mais



Existem diversas formas de executar um aplicativo em um *device*. Para saber mais sobre isso e se aprofundar quanto às possibilidades que estão à sua disposição, indicamos a leitura do artigo [Criar e executar o app](#). Certamente será uma leitura valiosa para agregar a seus estudos e garantir maiores conhecimentos!

Neste tópico, você conheceu o passo a passo de como testar o projeto do aplicativo, além de selecionar e aplicar o emulador voltado para essa finalidade.

Assim, com o emulador em execução e o teste de aplicação realizado, é necessário entender o conceito de interface. Veja mais sobre o assunto no tópico a seguir!

## Interface gráfica ou do usuário e tipos de atividades

Anteriormente, você criou sua primeira aplicação utilizando as conimagemções base do Android Studio, visto que elas auxiliam no processo de construção de aplicativos. Agora, você verá sobre as interfaces e os tipos de atividades que uma aplicação pode ter, como será organizada e a forma de aproveitarmos o que estará disponível para o projeto.

Ouça o *podcast* que preparamos na sequência para entender o conceito de interface!



### Podcast

Confira o [podcast](#) sobre o conceito de interface.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Agora que você já sabe criar uma aplicação fazendo uso das conimagemções bases do Android Studio, chegou o momento de estudar as interfaces e os tipos de atividades que ela pode ter. Vamos conhecer a diferença entre eles?

A interface do usuário, também conhecida como interface gráfica, nos direciona para uma gama de informações relacionadas a layouts, notificações, barras e diversas outras possibilidades de *user interface*, também conhecida pelas siglas UI.

Já um aplicativo Android é composto por componentes, sendo que, entre eles, encontramos a *activity*, que traduzindo para o português significa “atividade”. De modo geral, trata-se de classes que controlam os eventos da tela de um aplicativo. Sendo assim, para simplificar, uma atividade é uma janela do aplicativo!

Agora com essas definições bem claras, podemos seguir para ver em mais detalhes sobre os componentes, com os conceitos de *View* e *ViewGroup*.

Dante disso, quando você executou o primeiro aplicativo, foram criados, indiretamente, a atividade e todos os componentes que fazem parte dela.



#PraCegoVer: na imagem, temos a tela de um celular com o primeiro App em execução e sua atividade "Hello Word!". Aparecem seus botões e outras particularidades, como horário e nível de bateria.

Você deve estar se perguntando: isso significa que toda janela de celular é uma atividade?

Sim! E você pode, ainda, complementar mencionando que essa atividade pode ser composta por diversos componentes, os quais têm por objetivo auxiliar no processo de construção de uma aplicação.

### Saiba mais



Existem diversas atividades que interagem no aplicativo.

Pensando nisso, sugerimos a leitura do artigo [O conceito de atividades](#), que traz informações mais detalhadas quanto ao assunto, possibilitando o aprofundamento do tema!

Neste tópico abordamos o conceito de interfaces e os tipos de atividades (*activities*), sendo que esta última representa uma janela no *app* que possui um papel específico e é composta por componentes.

Ainda no que diz respeito aos componentes de uma atividade, eles são divididos em dois grupos: *View* e *ViewGroup*. Conheça-os melhor a seguir.

## Conceitos de *View* e *ViewGroup*

Uma *View* consiste naqueles componentes que você utilizou para montar a tela do aplicativo. Ela está ligada a uma *activity* que controla o seu comportamento. Para simplificar, as *Views* são os textos, os botões, a barra de progresso e tudo o que possa interagir ou não com o usuário do aplicativo:

<i>TextView</i>	Componente para mostrar um texto na tela.
<i>Button</i>	Botão para disparar uma ação.
<i>EditText</i>	Entrada de texto.
<i>Spinner</i>	Lista de itens.
<i>Checkbox</i>	Caixa de seleção.

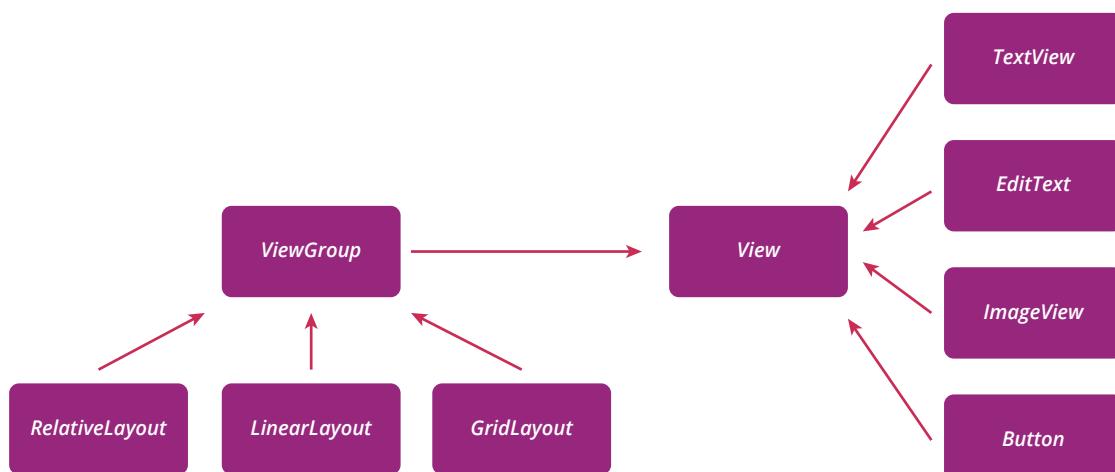
## Saiba mais



Existe uma grande quantidade de componentes que auxiliam na montagem da aplicação. Para conhecê-los e ter em mãos sempre que precisar, sugerimos que acesse o [Material Design: Components!](#)

Para organizar as *Views* na tela do aplicativo, o Android Studio dispõe de um gerenciador de layout, com o objetivo de arrumar e agrupar essas *Views*. A esse agrupamento, damos o nome de layout, que tem a possibilidade de organizar os componentes de modo que fiquem visualmente agradáveis aos usuários.

Observe com atenção o esquema a seguir, ele demonstra a relação dos componentes de uma *View*.



#PraCegoVer: na imagem, temos um esquema indicando a relação dos componentes de uma *View*. A *ViewGroup* é formada por *RelativeLayout*, *LinearLayout* e *GridLayout*. Dela, temos a *View*, que traz *TextView*, *EditText*, *ImageView* e *Button*.

Você irá inserir no aplicativo alguns componentes, mas antes, criaremos um protótipo para exemplificar. Para elaborá-lo, utilizaremos diversos aplicativos on-line, começando pelo Quant-UX.

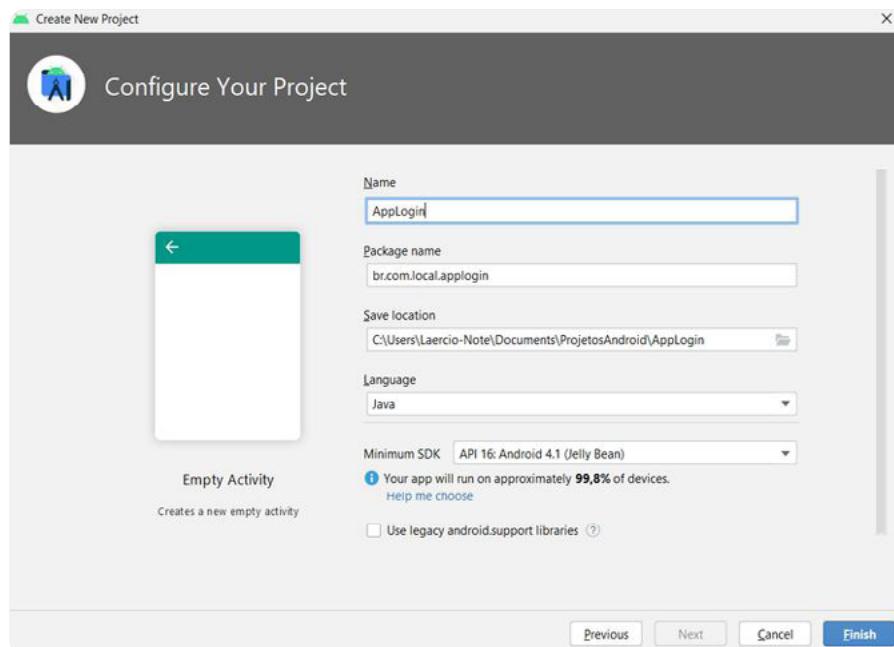


#PraCegoVer: na imagem, temos um protótipo de janela de login. Há um quadrado para foto, um campo para usuário, outro para a senha e os botões "Entrar" e "Sair".

No exemplo, identifique os componentes que fazem parte do layout. Seguindo o gráfico de *ViewGroup* apresentado acima, eles estão sob um layout principal, o qual agrupa de forma organizada e facilita a visualização.

Agora, crie esses componentes no Android Studio. Na janela principal do programa, selecione “Create a New Project”. Depois, selecione o template “Empty Activity”, como feito anteriormente.

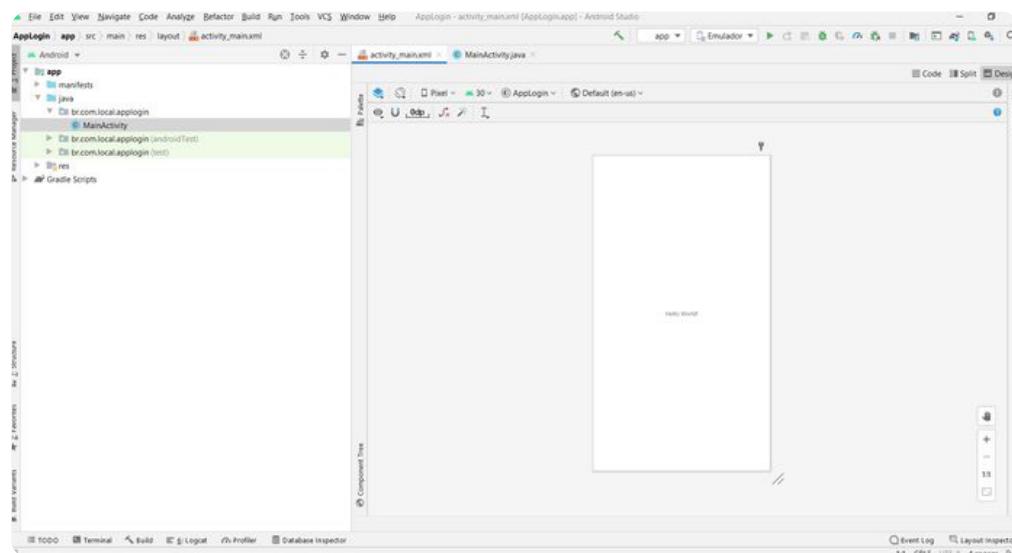
Em seguida, crie o nome do projeto “*AppLogin*”, que definirá o pacote e sua localização. O restante deixe como default, pois, no momento, não há necessidade de alterações.



#PraCegoVer: na imagem, temos um *print* do Android Studio na área de conimagemção do projeto. O *template* escolhido está selecionado à esquerda, com campos para preenchimento à direita, como nome do projeto, local de salvamento etc.

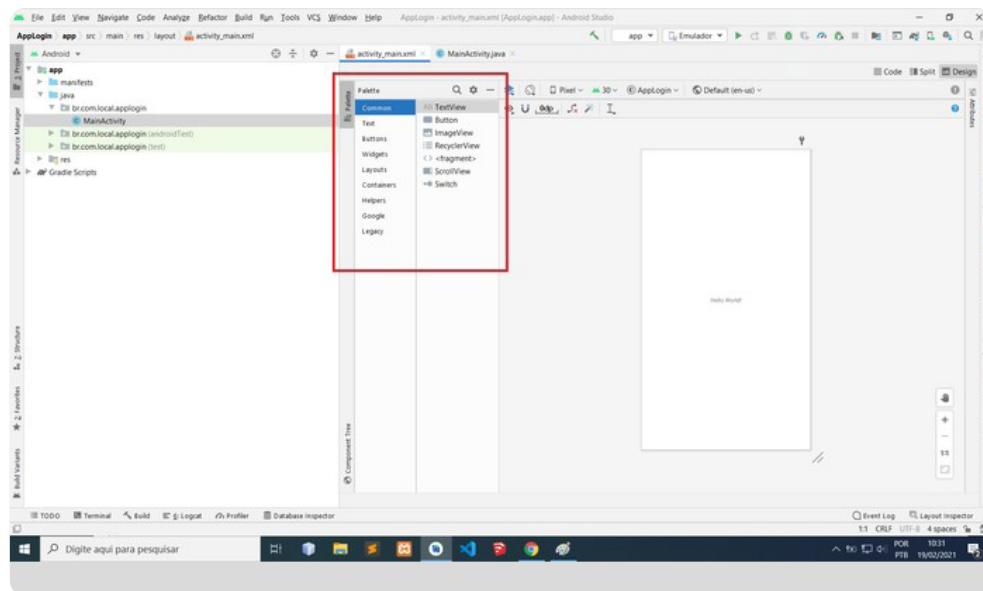
Basta, então, selecionar “*Finish*” e aguardar o processo de finalização da montagem do projeto. Como você já bem sabe, essa etapa demora um pouco, mas é preciso esperar a finalização para não haver problemas.

Acompanhe, no recurso, os próximos passos que devem ser seguidos.



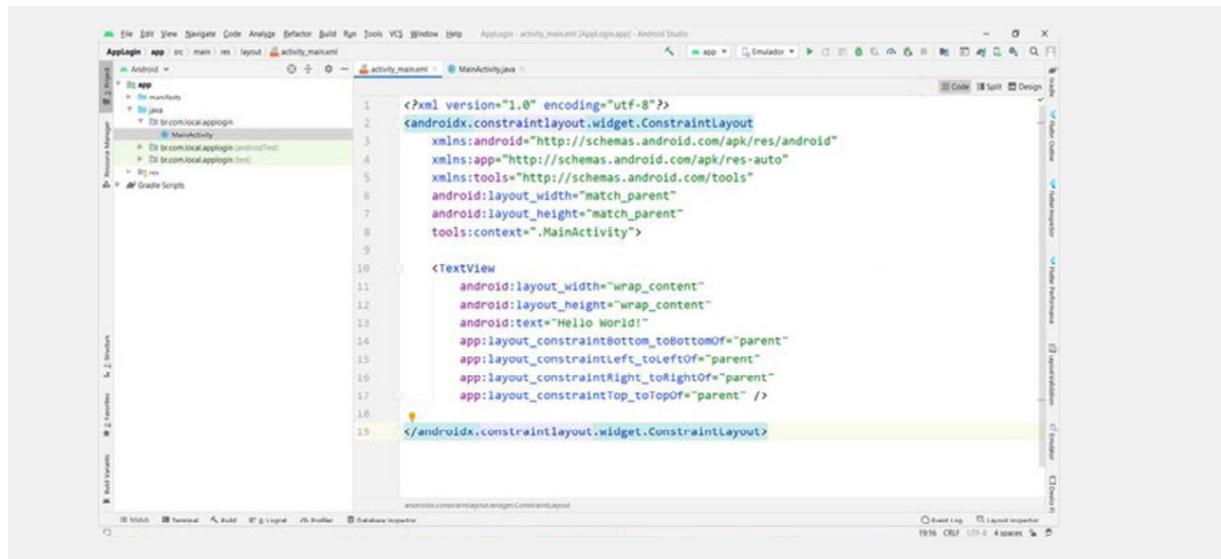
#PraCegoVer: na imagem, temos um *print* da janela do Android Studio para Windows. Nela, existem uma barra superior contendo o painel de opções, um *dashboard* à esquerda e uma tela onde ocorre a modelagem.

Após carregado, o projeto ficará com as estruturas exibidas e estará pronto para inserirmos os componentes, conforme o protótipo criado anteriormente.



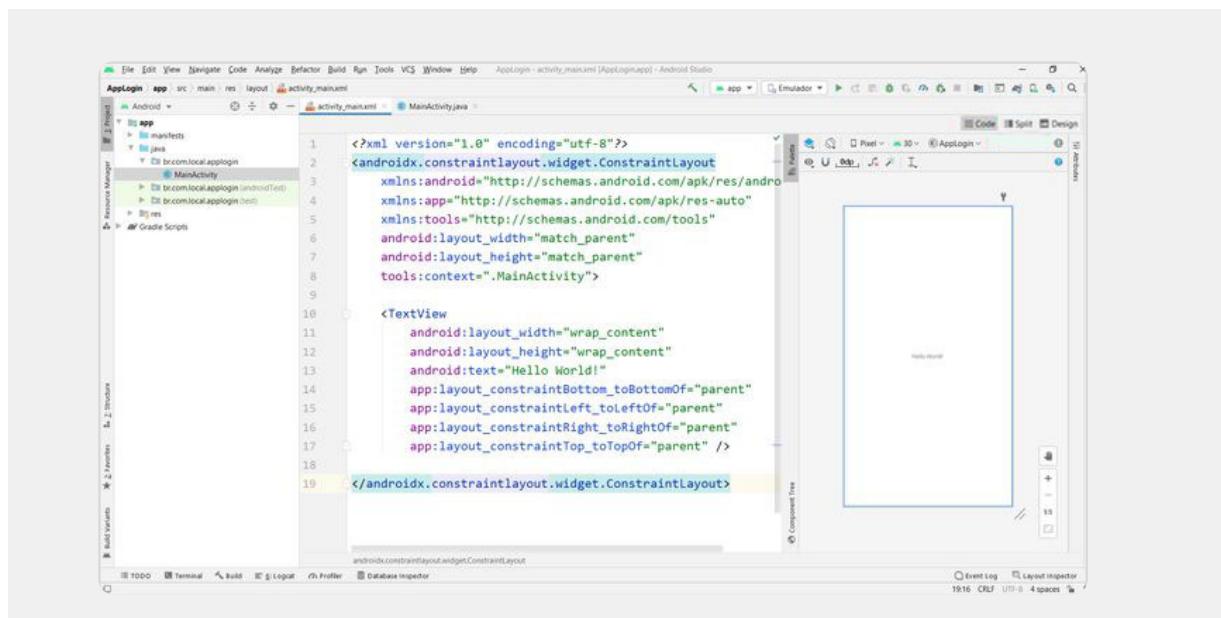
#PraCegoVer: na imagem, temos um *print* da janela do Android Studio para Windows. Nela, existem uma barra superior contendo o painel de opções, um *dashboard* à esquerda e uma tela onde ocorre a modelagem. Nesta última, há um trecho destacado em vermelho no qual está o painel de componentes.

Os componentes podem ser inseridos diretamente no layout a partir do painel de componentes (*palette*) ou pela codificação.



#PraCegoVer: na imagem, temos um *print* da janela do Android Studio para Windows. Nela, existem uma barra superior contendo o painel de opções, um *dashboard* à esquerda e uma tela onde ocorre a modelagem. Nesta última, há uma linha de código de programação e, no canto superior direito, há um trecho destacado em vermelho no qual está escrito “code”.

Observe na imagem para entender melhor, onde está o painel “Code”, ele apresenta a codificação do layout selecionado.



#PraCegoVer: na imagem, temos um *print* do modo de visualização “split”, em que temos a tela dividida em três. Do lado esquerdo, encontramos alguns repositórios. No centro, temos o código. Já à direita, temos a visualização do processo.

Ainda é possível deixar o código e design “split” apresentados no mesmo instante. Isso facilitará a observação das alterações em tempo de execução.

Até esse ponto, você aprendeu sobre os dois grupos de componentes de uma atividade, os *Views* e os *Grupviews*. Após isso, pôde criar um projeto a partir do tutorial apresentado e configurou o modo de visualização que será trabalhado. Não foi tão complicado, não é mesmo?

A seguir, você verá outro recurso muito relevante para o desenvolvimento de aplicações *mobile*, os *Widgets*.

## Widgets

Os *widgets* são componentes que permitem aos usuários interagirem com uma janela de aplicativo. Por haver inúmeras funções envolvidas nessa interação, logicamente existem diversos tipos de *widgets* em uma janela de aplicativo.

Esses *widgets* podem ser inseridos por meio da ferramenta “*Palette*”, como explicamos anteriormente, ou pelo código. Aqui, realizaremos pelo código. Aliás, alguns dos *widgets* mencionados já foram vistos, mas entraremos em mais detalhes para que possa fixar seus conhecimentos, combinado?

O *TextView* serve para mostrar um texto na janela de um aplicativo no sistema Android. Sua propriedade principal é a “*text*”. No slide a seguir, observe o código desenvolvido e em seguida como fica sua visualização após ser executado.

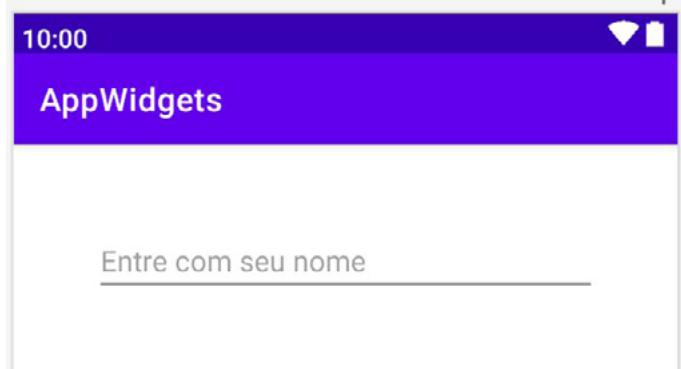
```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Inserindo um texto na janela"  
/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de *TextView*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com o trecho escrito "inserindo um texto na janela".

O *EditText*, por sua vez, permite a inserção de dados em um campo pelo teclado do dispositivo. Sua propriedade principal é o "*inputType*", que possibilita a definição quanto ao tipo de entrada e à conimagemão do teclado do dispositivo. Veja como fica o código e sua visualização quando o executamos!

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Entre com seu nome"  
    android:inputType="textPersonName"/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de *EditText*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com o trecho escrito "Entre com o seu nome".

Ficou claro até aqui?

Vamos seguir em frente!

Analise o quadro na sequência, ele indica os tipos de entradas do *EditText* que podem ser utilizadas. Confira!

inputType	Tipo de entrada de dados aceito
text	Texto
phone	Número de telefone
textPassword	Senha alfa-numérica
numberPassword	Senha numérica
number	Números
date	Data
textMultiLine	Texto com quebra de linha
textEmailAddress	Endereço de e-mail

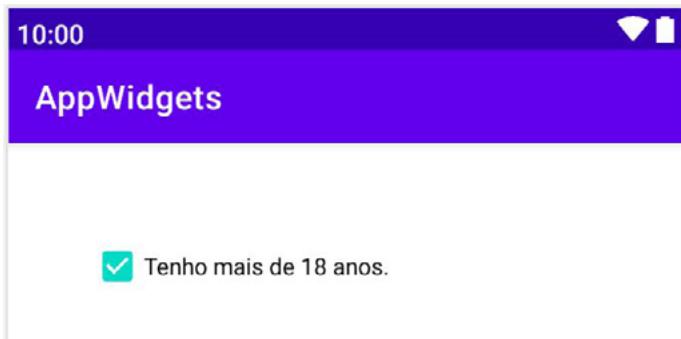
#PraCegoVer: no quadro, temos os tipos de entradas de *EditText*. Na primeira coluna, encontramos os *inputType*, incluindo *text*, *phone*, *textPassword*, *numberPassword*, *number*, *date*, *textMultiLine* e *textEmailAddress*. Já na segunda coluna, encontramos os tipos de entrada de dados aceitos, sendo, respectivamente, texto, número de telefone, senha alfanumérica, senha numérica, números, data, texto com quebra de linha e endereço de e-mail.

Agora conheça os demais componentes que você pode utilizar em sua aplicação.

Já o *CheckBox* serve para inserir uma caixa de seleção com definição de estado de selecionado ou não. Ele apresenta as opções *checked* “true” ou “false”.

A seguir, observe as opções de *checked* “true” em código e como fica na visualização. Veja:

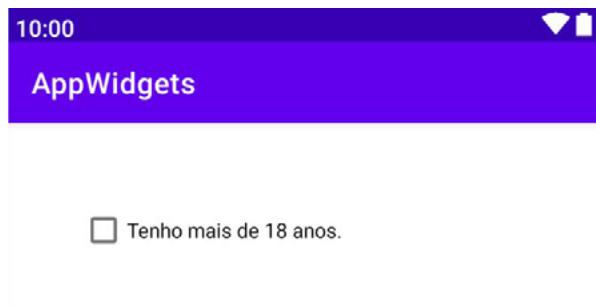
```
<CheckBox  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Tenho mais de 18 anos"  
    android:checked="true"  
/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de CheckBox. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com o trecho escrito "Tenho mais de 18 anos" e um ícone com um check marcado.

O recurso apresentado abaixo nos traz as opções de *checked* "false". Veja como é aplicado no código e como ele fica na visualização após a execução. Observe atentamente as diferenças!

```
<CheckBox  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Tenho mais de 18 anos"  
    android:checked="false"  
/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de CheckBox. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com o trecho escrito "Tenho mais de 18 anos" e um ícone com um check desmarcado.

Ainda temos o *RadioButton*, que serve para inserir um grupo de botões em que você poderá selecionar apenas uma opção disponível. Para tanto, deve-se inserir o *RadioButton* em um *widget* chamado *RadioGroup*. A propriedade *checked* pode ser

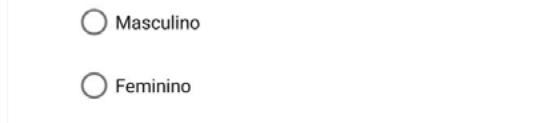
"true" ou "false", assim como o *widget* anterior. Veja a seguir!

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <RadioGroup
        android:id="@+id/sexo_masculino"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Masculino"/>

    <RadioGroup
        android:id="@+id/sexo_feminino"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Feminino"/>

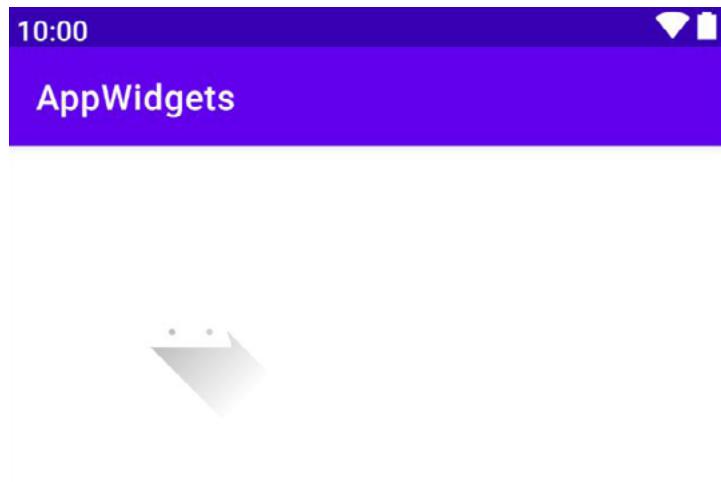
</RadioGroup>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de *RadioButton*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com duas opções desmarcadas. A primeira, de cima para baixo, tem o trecho escrito "Masculino" e a de baixo tem o trecho escrito "Feminino".

O *ImageView* serve para apresentar uma imagem na janela do aplicativo. O código mostrará a imagem na tela chamada "*ic\_launcher\_foreground*", a qual deve estar dentro da pasta "*res/drawable*" em "*Drawable Resources*". Veja como aplicá-lo no código e a visualização final.

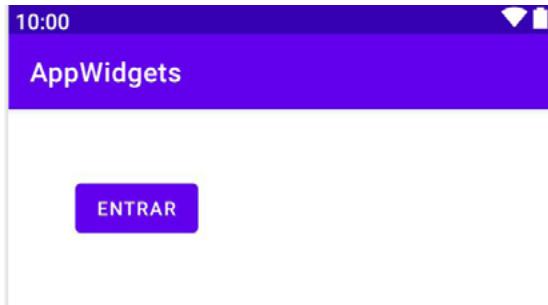
```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher_foreground"/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de *ImageView*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um painel com a logo da empresa Android.

O *Button* nos mostra um botão na janela do aplicativo, o qual, ao ser clicado, gera um evento. Observe o código e o resultado:

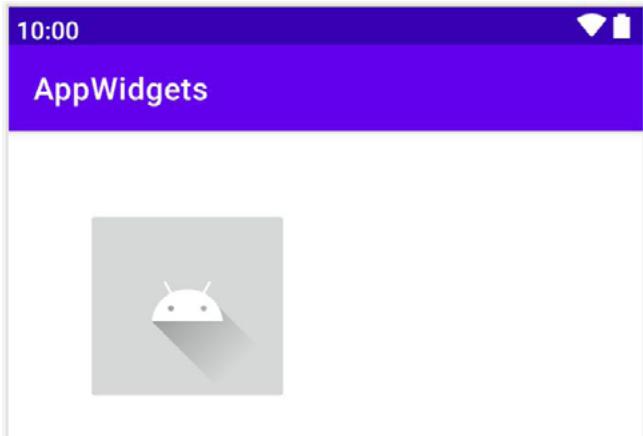
```
<Button  
    android:id="@+id	btnEntrar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Entrar"/>
```



#PraCegoVer: na imagem, temos a tela de um celular com design de *Button*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um botão no qual está escrito "entrar".

Já o *ImageButton* apresenta as mesmas funções do *Button*, porém, em vez de exibir o botão com um texto, exibirá uma imagem e permitirá que uma ação seja disparada quando o usuário clicar nessa imagem.

```
<ImageButton  
    android:id="@+id/btnEntrar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/ic_launcher_foreground"/>
```

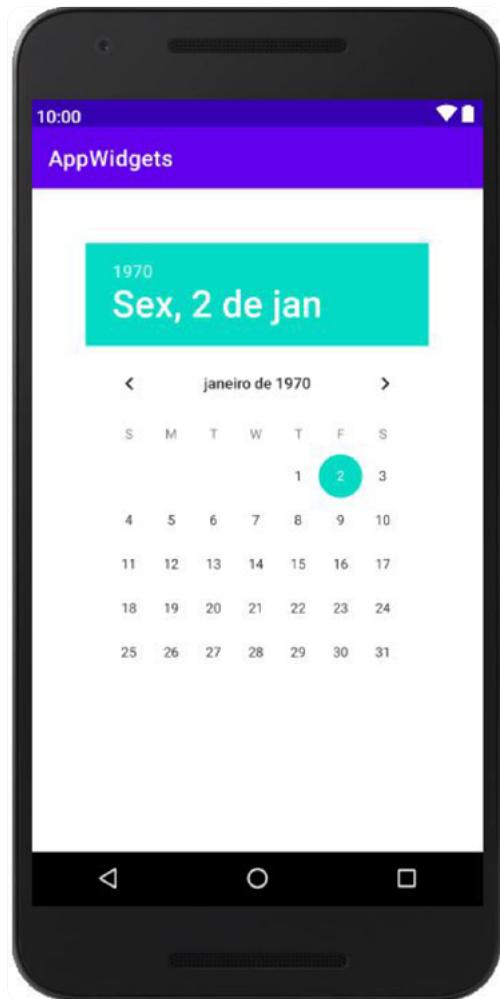


#PraCegoVer: na imagem, temos a tela de um celular com design de *ImageButton*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando onde está escrito "AppWidgets". Abaixo, há um botão que contém a logo da Android.

Também precisamos citar o *DatePicker*, que insere um calendário permitindo a seleção de uma data qualquer. É possível recuperar as informações e os eventos do calendário para utilizar no aplicativo. Assim, ao fazer uso da propriedade "*datepickermode*", é possível alterar o tipo de calendário. O código ficará assim:

```
<DatePicker  
    android:id="@+id/dtpAgenda"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

Já o modelo de visualização pode ser observado na próxima imagem:



#PraCegoVer: na imagem, temos a tela de um celular com design de *DatePicker*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando o ano, o dia da semana e a data. Abaixo, há um calendário com todos os dias do mês de janeiro do ano de 1970.

Por fim, o *TimePicker* insere um relógio na janela do aplicativo. Dessa forma, ao utilizar a propriedade “*timepickermode*”, pode-se alterar o tipo de relógio.

```
<TimePicker  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

O modelo de visualização do *TimePicker* é o seguinte:



#PraCegoVer: na imagem, temos a tela de um celular com design de *TimePicker*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma barra marcando o horário e se é manhã ou tarde. Abaixo, há um relógio analógico.

Neste tópico, você conheceu não só o conceito de *widgets*, como também pôde acompanhar o tutorial de como programá-los utilizando códigos específicos para cada tipo.

Continuando seus estudos, você verá agora quais são os principais tipos de layout, principalmente para aplicar em um sistema operacional como o Android.

# Os principais tipos de layouts para a aplicação no Android

Após estudar sobre os *widgets* e suas formas de codificações, vamos progredir para o estudo dos layouts, esse recurso é fundamental para os aplicativos *mobile*.

## Entenda o Conceito



Os layouts são utilizados no agrupamento dos componentes ou *Views* para auxiliar no processo de criação de uma janela de aplicativo. Dessa forma, é necessário conhecer os tipos de layouts existentes e entender como utilizá-los na criação de um aplicativo adequado para o projeto e que ofereça uma boa experiência para o usuário.

Em geral, existem alguns tipos de layouts principais para a criação dos aplicativos. O *LinearLayout*, por exemplo, é utilizado para alinhar os componentes na vertical ou horizontal, fazendo com que as *Views* fiquem uma embaixo da outra ou uma do lado da outra.

Acompanhe a disposição do código *LinearLayout* na vertical!

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

```

res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="30dp"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <ImageView
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="50dp"
        android:src="#2d2d2d" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="200px"
        android:hint="Entre com usuário"
        android:inputType="textPersonName" />

```

```

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:hint="Entre com sua senha"
    android:inputType="textPassword"/>
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Entrar"/>
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Sair"/>
</LinearLayout>

```

Veja como fica o design do código aplicado, quando o executamos:



#PraCegoVer: na imagem, temos a tela de um celular com design de *LinearLayout* na vertical. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos um quadrado para foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, um de "Entrar" e outro de "Sair".

Outro tipo de layout é o *RelativeLayout*, utilizado para alinhar os componentes de forma relativa a outro componente, permitindo declarar que determinado componente fique à esquerda, direita, abaixo ou acima de outro.

Esse tipo de layout merece uma atenção especial, pois o posicionamento dos componentes utiliza como referência o componente que foi criado anteriormente. Desse modo, chama-se “componente pai” o criado primeiro e “componente filho” aquele que foi criado posteriormente.

Observe a disposição do código *RelativeLayout*.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
res/android"
    <EditText
        android:id="@+id/edtSenha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edtUsuario"
        android:layout_marginTop="20dp"
        android:hint="Entre com sua senha"
        android:inputType="textPassword" />
    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:layout_margin="30dp"
    tools:context=".MainActivity">

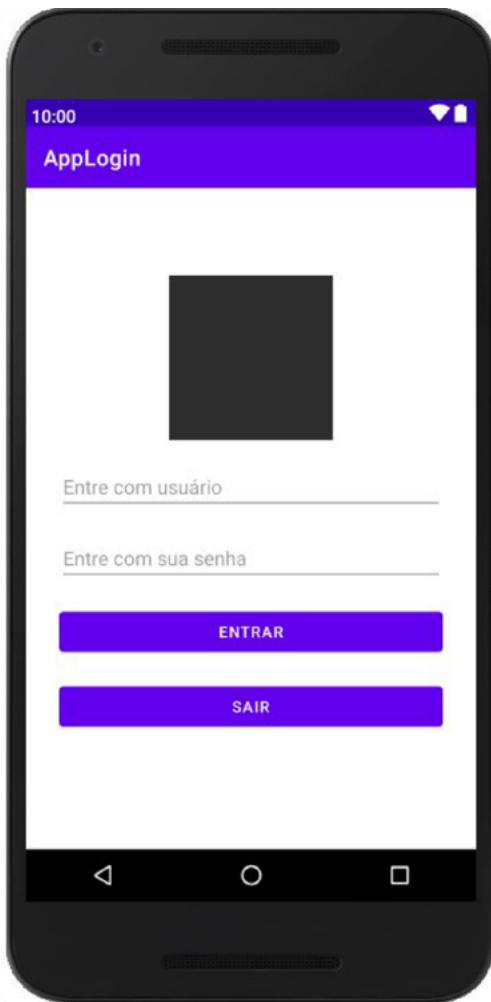
    <ImageView
        android:id="@+id/imgLogo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:src="#2d2d2d"/>
```

Observe a disposição do código *RelativeLayout*.

```
<Edit Text
    android:id="@+id/edtUsuario"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/imgLogo"
    android:layout_marginTop="20dp"
    android:hint="Entre com usuário"
    android:inputType="textPersonName" />

<Button
    android:id="@+id(btnEntrar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/edtSenha"
    android:layout_marginTop="20dp"
    android:text="Entrar" />
<Button
    android:id="@+id	btnSair"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id	btnEntrar"
    android:layout_marginTop="20dp"
    android:text="Sair" />
</RelativeLayout>
```

Para visualizar o design de *RelativeLayout*, temos o seguinte:



#PraCegoVer: na imagem, temos a tela de um celular com design de *RelativeLayout*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos um quadrado para foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, um de "Entrar" e outro de "Sair".

Já o *TableLayout* é utilizado para alinhar os componentes em forma de tabela. Como toda tabela, ele será definido por linhas e colunas. Veja a disposição do seu código na sequência.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout 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:layout_margin="30dp"
    android:stretchColumns="2"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imgLogo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_marginTop="50dp"
        android:src="#2d2d2d" />

    <EditText
        android:id="@+id/edtUsuario"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:hint="Entre com usuário"
        android:inputType="textPersonName" />

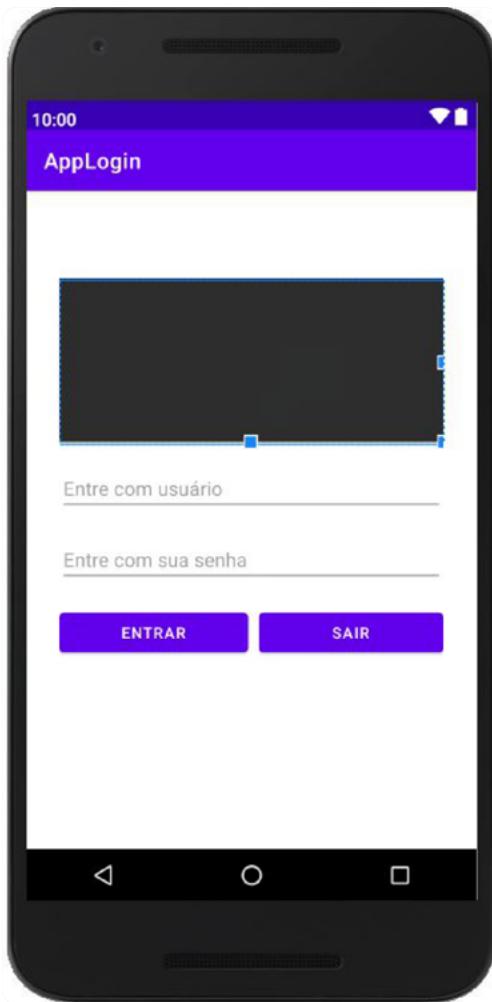
    <EditText
        android:id="@+id/edtSenha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:hint="Entre com sua senha"
        android:inputType="textPassword" />
```

```
<TableRow>

    <Button
        android:id="@+id/btnEntrar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_weight="1"
        android:text="Entrar" />

    <Button
        android:layout_marginLeft="10dp"
        android:id="@+id/btnSair"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_weight="1"
        android:text="Sair" />
</TableRow>
</TableLayout>
```

Vamos, então, analisar como fica o design de *TableLayout*? Confira!



#PraCegoVer: na imagem, temos a tela de um celular com design de *TableLayout*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos um retângulo para foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, agora um do lado do outro, de "Entrar" e "Sair".

Ainda temos o *FrameLayout*, normalmente aplicado para acomodar somente um componente na tela do dispositivo, bem como na criação de janelas com uma imagem ou barra de progresso.

Nesse caso, veja como fica sua disposição em apenas um código, acompanhe o recurso abaixo:

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

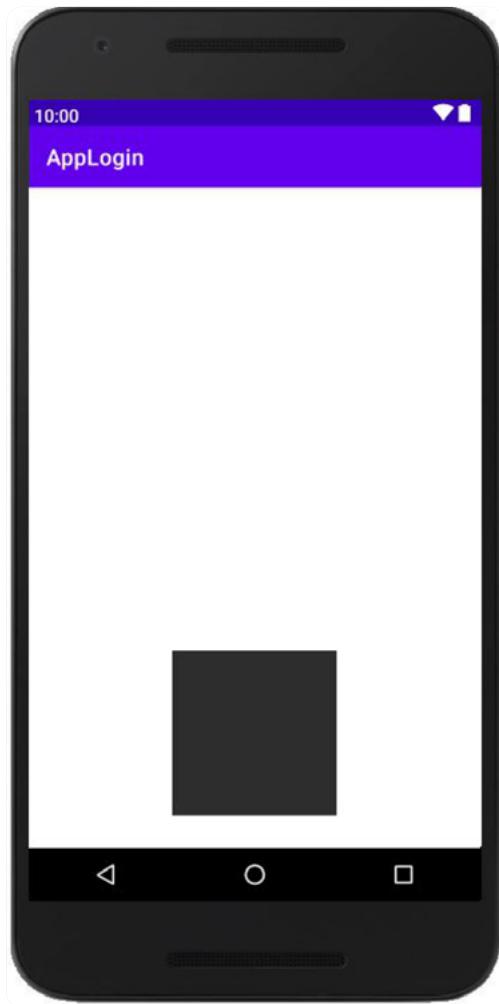
```
res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="30dp"
    tools: context=".MainActivity">

    <ImageView
        android:layout_gravity="center_horizontal
        android:id="@+id/imgLogo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_margin_top="50dp"
        android:src="#2d2d2d" />

</FrameLayout>
```

Você pode observar a aplicação do código que gera o *FrameLayout*.

Agora você está visualizando o design como fica após executar o código utilizando o *FrameLayout*.



#PraCegoVer: na imagem, temos a tela de um celular com design de *FrameLayout*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. Na parte inferior, no centro, encontramos um quadrado para foto.

Por último, mas não menos importante, o *ScrollView* é utilizado quando a tela necessita de uma barra de rolagem ou *scrollbar*, pois os componentes não cabem em uma só janela da tela do dispositivo. Seu código ficará conforme mostrado a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView 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=".MainActivity">

    <LinearLayout
        android:layout_margin="30dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

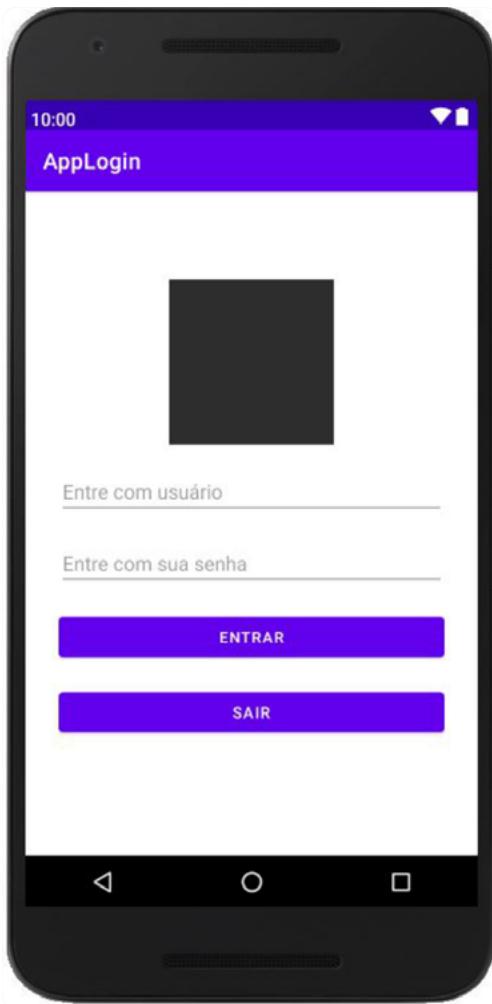
        <ImageView
            android:layout_gravity="center_horizontal"
            android:id="@+id/img_Logo"
            android:layout_width="150dp"
            android:layout_height="150dp"
            android:layout_marginTop="56dp"
            android:src="#2d2d2d" />

        <EditText
            android:id="@+id/edtUsuario"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:hint="Entre com usuário"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/edtSenha"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:hint="Entre com sua senha"
            android:inputType="textPassword" />
```

```
<Button  
    android:id="@+id	btnEntrar"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="20dp"  
    android:text="Entrar" />  
  
<Button  
    android:id="@+id	btnSair"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="20dp"  
    android:text="Sair" />  
</LinearLayout>  
  
</ScrollView>
```

No caso do design de *ScrollView*, temos duas partes, conforme a imagem abaixo:



#PraCegoVer: na imagem, temos a tela de um celular com design de *ScrollView*. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos um quadrado para foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, um de "Entrar" e outro de "Sair".



#PraCegoVer: na imagem, temos a tela de um celular com design de ScrollView. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos um quadrado para foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, um de "Entrar" e outro de "Sair". O destaque vai para uma barra de rolagem à direita, quando o celular está no modo horizontal.

O *ScrollView* ou a barra de rolagem aparece quando os componentes não cabem na tela, conforme apresentado anteriormente. Ele é muito útil para quando necessitamos de visualizações mais dinâmicas, com diversos posicionamentos de tela no *device*.

## Saiba mais



Existem outros layouts no Android Studio, sendo que dominar esse conceito é de suma importância para criarmos layouts mais dinâmicos e com um design que melhore a experiência do usuário. Diante dessa relevância, sugerimos a leitura do artigo [Como Dominar os Android Layouts em 07 Passos](#), escrito por Filipe Cordeiro.

Desta forma, após retomar o conceito de layout, você pôde se aprofundar um pouco mais nesse recurso, analisando os seus vários tipos, como também qual é o melhor a depender do projeto demandado. Além disso, você também viu as linhas de código de cada tipo de layout, as quais futuramente, você poderá aplicar.

Assim, conhecendo esses modelos de layouts, também precisamos ter atenção a outro componente relacionado a eles: a eles que veremos a seguir.

## Notificações

Como você viu, estudamos sobre os tipos de layout, bem como suas respectivas linhas de código. Agora, será apresentado um outro recurso importante: as notificações.

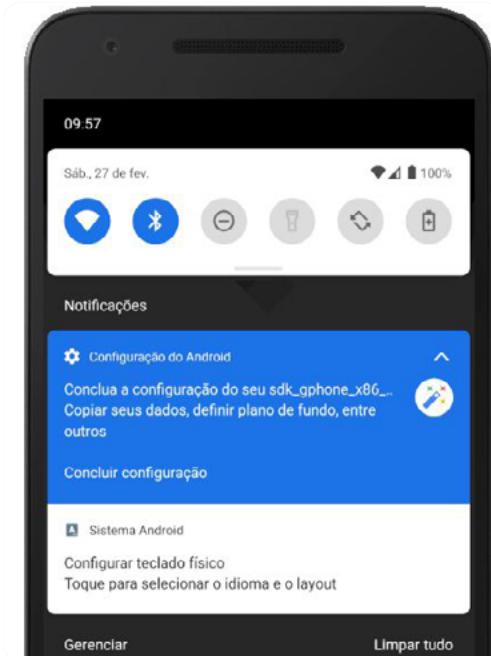
### Entenda o Conceito



As notificações são mensagens que aparecem distantes da interface do usuário, como um ícone na barra do aplicativo. Para que o indivíduo possa ver a notificação, ele precisa abrir a barra de notificações, pois as informações são controladas pelo sistema..



#PraCegoVer: na imagem, temos parte da ilustração de um smartphone. O destaque está em sua barra de notificações. Nela, encontramos o horário, dois ícones de notificação, sinal de Wi-Fi, rede telefônica e nível de bateria do aparelho.



#PraCegoVer: na imagem, temos parte da ilustração de um smartphone. O destaque está em sua barra de notificações aberta. Nela, encontramos o horário acima, o dia da semana e a data mais abaixo, junto dos ícones de sinal de Wi-Fi, rede telefônica e nível de bateria do aparelho. Há, ainda, opções a serem selecionadas, como *bluetooth* e lanterna. Mais abaixo, encontramos duas notificações descritas, havendo a opção de gerenciar ou limpar tudo.

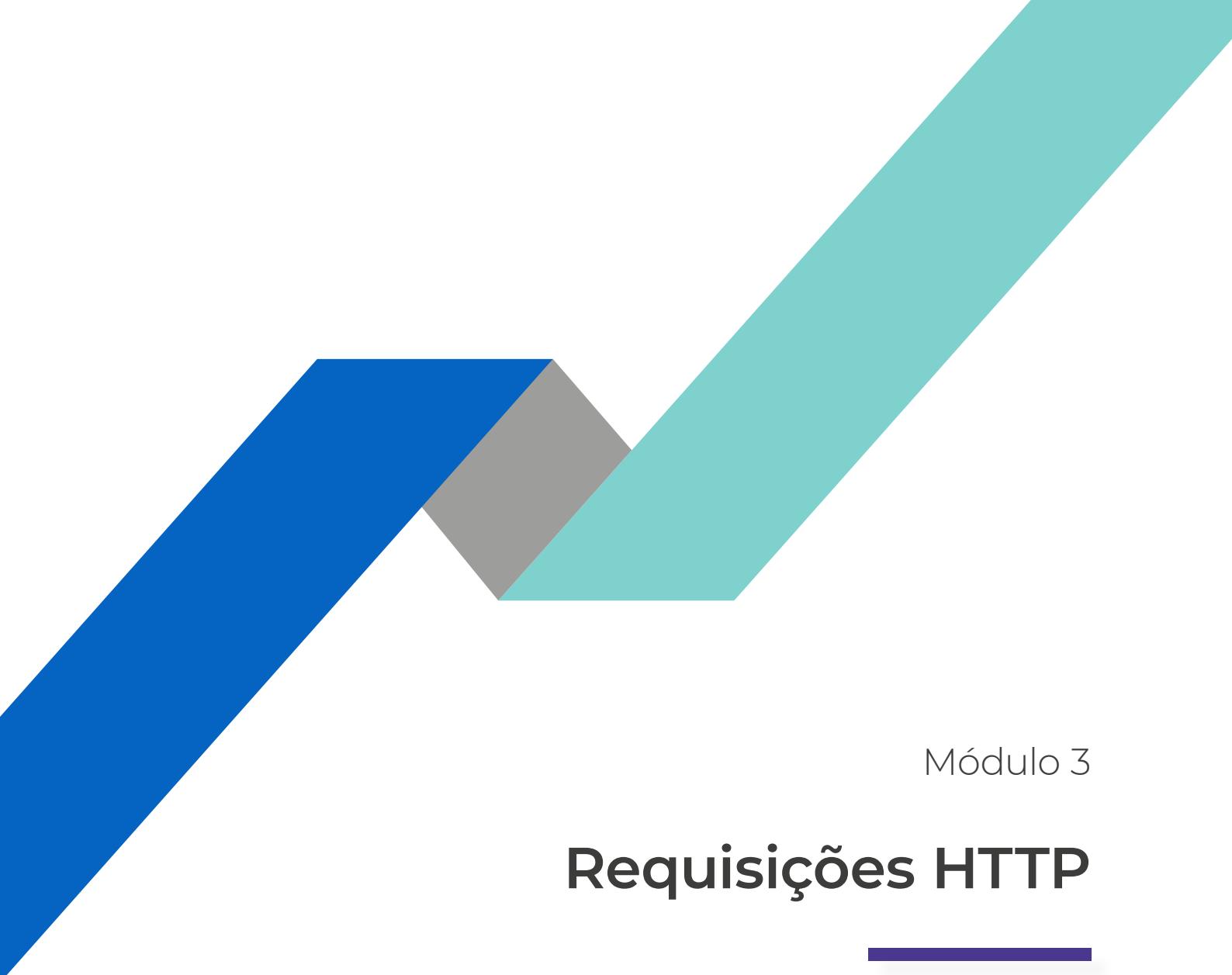
Outro tipo de notificação é o de sistema/aplicativo, acionada a partir de uma ação do usuário no aplicativo. Essa notificação é muito utilizada, por exemplo, quando tentamos acessar o sistema. Veja como ela é aplicada na prática:

```
//Criando um Toast para exibir mensagem de ação do sistema
Toast.makeText(getApplicationContext(),
    text: "Bem vindo ao sistema",
    Toast.LENGTH_SHORT).show();
```



#PraCegoVer: na imagem, temos a tela de um celular com design da janela (*toast*). Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma foto e dois campos, sendo um para usuário e outro para senha. Abaixo, há dois botões, um de "Entrar" e outro de "Sair". Por fim, há destaque para o recado "Bem-vindo ao sistema".

O *toast* é utilizado quando queremos passar uma informação para o usuário sem a interação direta sobre a tela. Isso porque ele aparece e tem um tempo determinado para se manter ativo, que pode até mesmo ser de alguns segundos.



Módulo 3

# Requisições HTTP

---

## Saiba mais



A notificação diz respeito a mensagens curtas que aparecem ao longo do processo de utilização do sistema do aplicativo ou provenientes de uso do aplicativo. Para obter maiores informações referentes à criação de uma notificação básica, não deixe de ler o artigo [Criar uma Notificação Básica](#), que será de grande ajuda!

Parabéns! Você chegou ao final do módulo 2.

Neste módulo, você compreendeu a maneira pela qual é possível criar, executar e testar um aplicativo a partir do Android Studio. Assim, primeiramente, você foi apresentado à interface primária dessa plataforma de desenvolvimento

Em seguida, chegou a vez de compreender como iniciar o projeto de criação do aplicativo, sobretudo, a partir dos *templates* disponíveis pela plataforma. Viu sobre a etapa de como fazer o processo de testagem do projeto do aplicativo, isto é, analisar em qual *device* (smartphone, relógio, televisão etc.) o *app* será mais eficaz. Depois disso, passamos a tratar sobre as interfaces, os tipos de atividades de um aplicativo e como essas serão dispostas

Você aprendeu sobre os dois grupos de componentes de uma atividade, os *Views* e os *GrupViews*. Depois disso, tendo como base o tutorial, você iniciou a criação de um protótipo de *app* a partir dos modelos de *widgets*, de layout e de notificações disponíveis na plataforma. Por último, experimentou o processo de testagem desse aplicativo.

No próximo, analisaremos as requisições que o *app* poderá fazer utilizando o protocolo HTTP e o carregamento de arquivos, aspectos essenciais para que você possa desempenhar de modo eficaz a sua atividade profissional de aplicação *mobile*.

# Requisições HTTP

No módulo anterior, você viu na prática a maneira pela qual é possível criar, executar e testar os projetos de aplicativo *mobile* com base na plataforma Android Studio.

Agora, nesse último módulo, você conhecerá os principais protocolos de comunicação e entenderá como carregar os arquivos e conectar o banco de dados *SQLite*. Posteriormente, você aprenderá de que maneira é publicado um aplicativo no Google Play. Vamos lá!

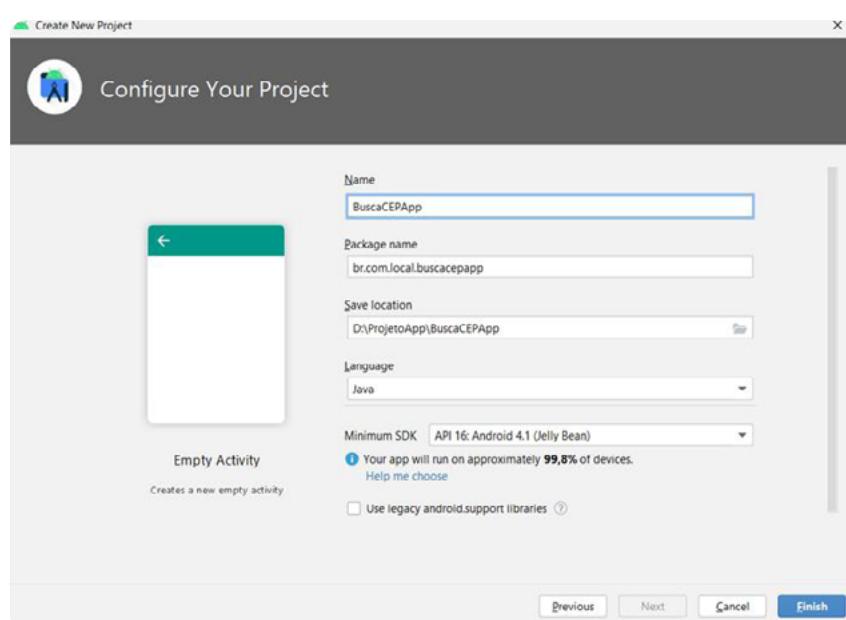
## Carregando arquivos externos

Para o acesso aos arquivos externos da aplicação, é necessário utilizar os protocolos de comunicação capazes de garantir a confiabilidade, integridade, disponibilidade e segurança dos dados transmitidos. Por isso, você utilizará aqui uma *Application Programming Interface* (API) para fazer uma requisição HTTP segura e retornar com os dados para o aplicativo.

É válido citar que uma API é definida como um conjunto de regras preestabelecidas, as quais fornecem comunicação por meio de protocolos para fins distintos do aplicativo.

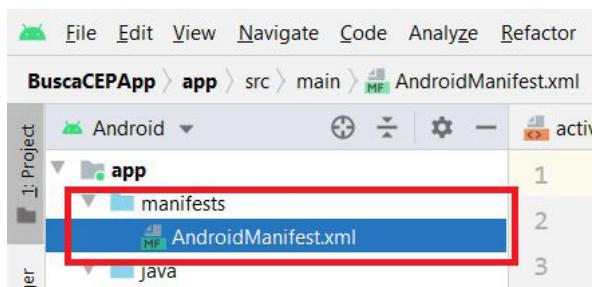


#PraCegoVer: Na imagem, existe uma mesa de escritório, sob a qual a um notebook aberto, ao lado esquerdo, há uma mão com um celular ligado, à direita, há uma xícara de café com fones de ouvido abaixo dela. No fundo, há um cacto em um pequeno vaso.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior no qual está escrito "Configure Your Project". Abaixo dele, existe uma tela contendo itens de configuração do projeto.

Após completado o carregamento do projeto, para acessar os arquivos externos, configure o acesso do aplicativo a internet, abrindo a pasta "manifests". Agora clique em "AndroidManifest.xml".



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Abaixo dele, existe um dashboard com um trecho destacado em vermelho contendo dois ícones, um de uma pasta com o nome "manifests" e o segundo com um ícone de um programa "xml" cujo nome é "AndroidManifest.xml".

Na sequência, faça a inserção do código destacado.

```
<uses-permission android:name="android.permission.  
INTERNET" />
```

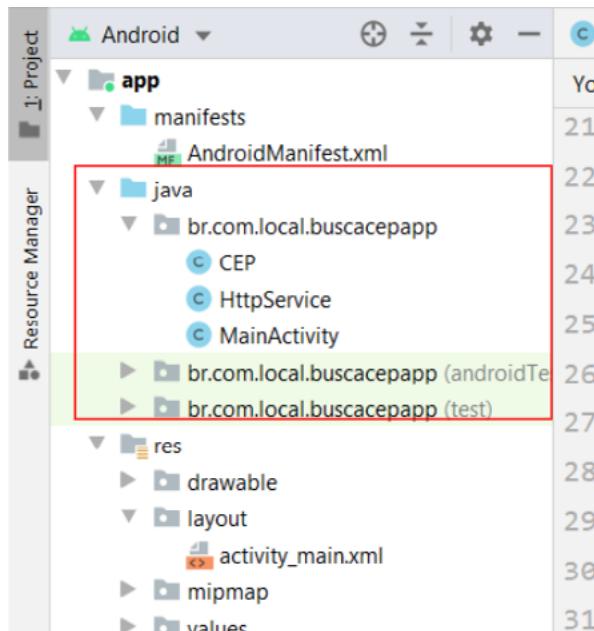
Com as configurações realizadas, monte o aplicativo para acessar a API de busca por CEP.

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.  
com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:orientation="vertical"  
        android:padding="50dp"  
        tools:context=".MainActivity">  
  
    <EditText  
        android:id="@+id/txtCep"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:digits="0123456789"  
        android:hint="Insira seu CEP"  
        android:inputType="number"  
        android:textColor="#595959"  
        android:textSize="25sp" />  
  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:gravity="center"  
        android:orientation="horizontal">  
  
        <Button  
            android:id="@+id/btnBuscaCep"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:layout_marginTop="20dp"  
            android:layout_marginBottom="10dp"  
            android:background="@color/design_  
            default_color_on_primary"
```

```
        android:padding="10dp"
        android:text="Buscar CEP"
        android:textSize="2sp"/>
    </LinearLayout>
    <TextView
        android:id="@+id/lblResposta"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        android:textColor="#000000"
        android:textSize="20sp"/>
    </LinearLayout>
```

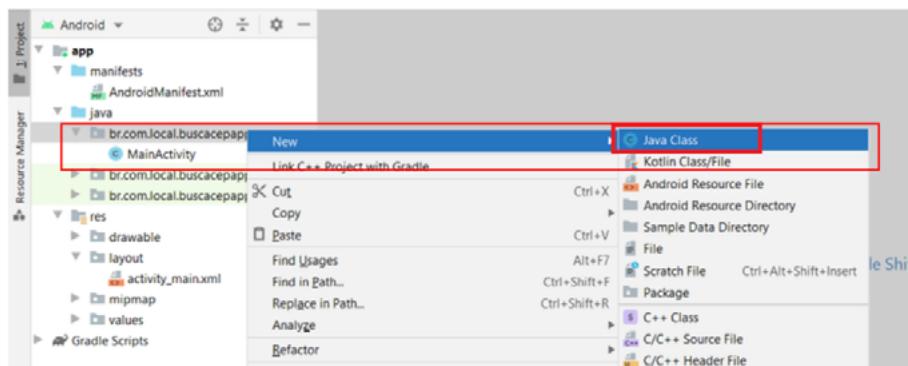
Finalizada a codificação do layout, avance para o código da parte que acessa os arquivos e retorna o resultado para o aplicativo. Faça isso em “java” e “br.com.local.buscacepapp”.

Nesse caso, será apresentado três arquivos para o desenvolvimento do trabalho.



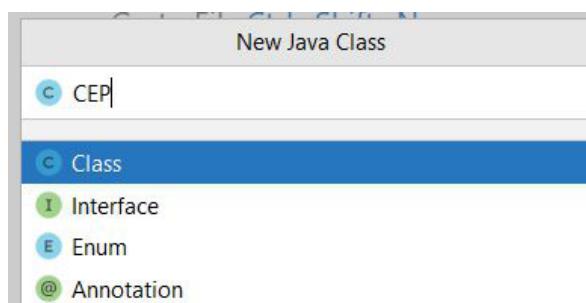
#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma.

Para criar os arquivos, clique com o botão direito do mouse.



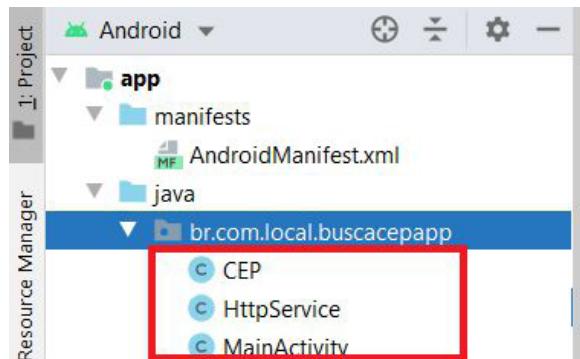
#PraCegoVer: na imagem, temos um print da tela do Android Studio. Nela, há um dashboard à esquerda e, do meio até à direita, há duas janelas com opções de conimagemção.

Após clicar Java Class abrirá a aba Class e você poderá criar o “CEP.java”.



#PraCegoVer: na imagem, temos um print da tela do Android Studio. Nela, há uma janela com o título “New Java Class” no topo e opções de conimagemção abaixo.

Observe com atenção o resultado. É importante destacar que o “CEP.java” é o objeto para troca de informações.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Abaixo dele, existe um dashboard com um trecho destacado em vermelho contendo três ícones, cada um formado por um círculo com a letra "c" dentro, com os seguintes trechos escritos: "CEP". "HttpService" e "MainActivity".

```

package br.com.local.buscacepapp;

public class CEP {
    private String cep;
    private String logradouro;
    private String complemento;
    private String bairro;
    private String localidade;
    private String uf;
    public CEP() {

    }

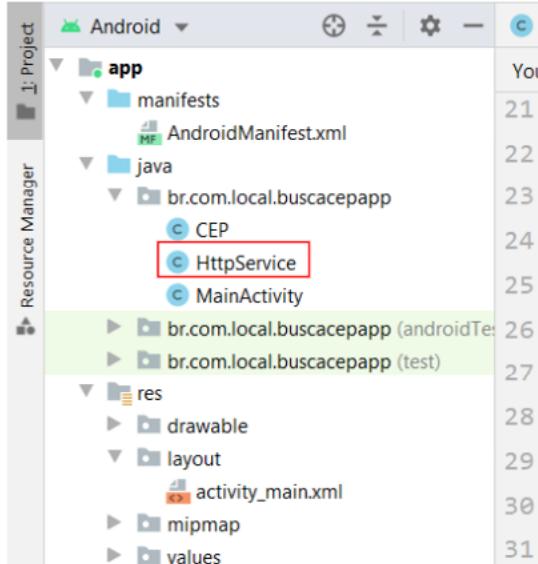
    public String getCep() { return cep; }
    public void setCep(String cep) { this.cep = cep; }
    public String getLogradouro() { return logradouro; }
    public void setLogradouro (String logradouro) {
        this.logradouro = logradouro; }
    public String getComplemento() { return complemento; }
    public void setComplemento(String complemento) { this.
complemento = complemento; }
    public String getBairro() { return bairro; }
    public void setBairro(String bairro) { this.bairro =
bairro; }
    public String getLocalidade() { return localidade; }
    public void setLocalidade (String localidade) { this.
localidade = localidade;}
    public String getUf() { return uf; }

    public void setUf(String uf) { this.uf = uf; }

    @Override
    public String toString() {
        return "CEP: " + getCep()
        + "\nLogradouro: " + getLogradouro()
        + "\nComplemento: " + getComplemento()
        + "\nBairro: " + getBairro()
        + "\nCidade:" + getLocalidade()
        + "\nEstado: " + getUf();
    }
}

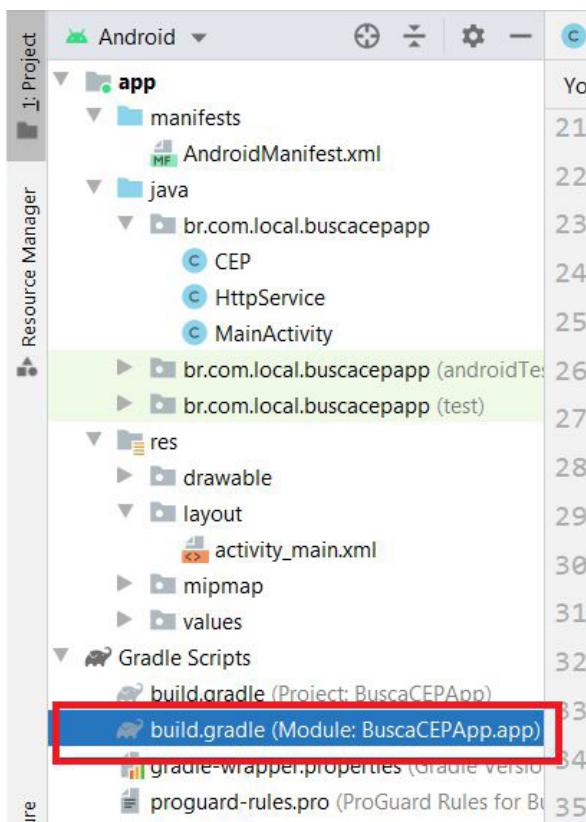
```

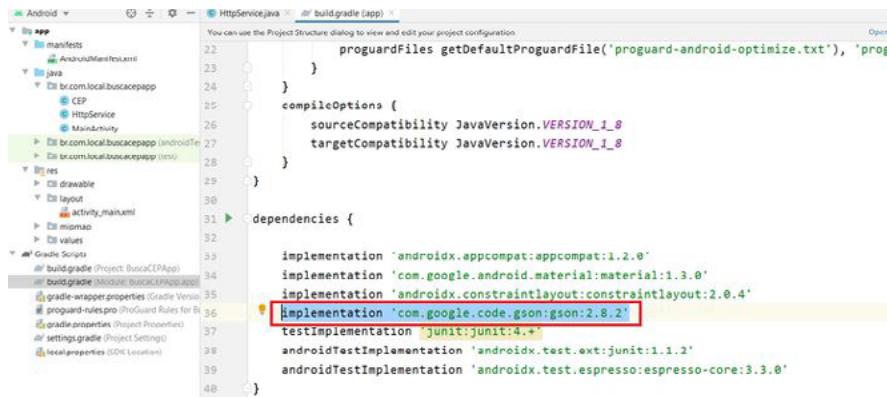
Já o `HTTPService` traz as conimagemções necessárias para o acesso à API CEP.



#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma.

A fim de utilizar essa classe, deve-se implementar uma API externa, onde, no projeto do Google, é a `GSON`, o que requer a conimagemção do arquivo.





```

    ...
    dependencies {
        implementation 'androidx.appcompat:appcompat:1.2.0'
        implementation 'com.google.android.material:material:1.3.0'
        implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
        implementation 'com.google.code.gson:gson:2.8.2' // Line 36
        testImplementation 'junit:junit:4.+'
        androidTestImplementation 'androidx.test.ext:junit:1.1.2'
        androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    }
}

```

#PraCegoVer: na imagem, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Abaixo temos um print da janela do Android Studio para Windows. Nela, existe o dashboard à esquerda. Do centro para a esquerda, há o painel de modelagem com várias linhas de código escrita.

Após inserir essa linha, conforme indicado, sincronize o arquivo com o projeto, clicando em “Sync Now”, localizado na parte superior à direita da mesma janela.



#PraCegoVer: temos um print da janela do Android Studio para Windows. Nela, existe o painel de modelagem com várias linhas de código escrita. No canto superior direito está um destaque no botão de “Sync Now”.

Em seguida, configure a Classe “*HTTPService.class*”.

```

package br.com.local.buscacepapp;

import ...

public class HttpService extends AsyncTask<Void, Void, CEP>
{
    private final String cepInserido;
    public HttpService(String cep) { this.cepInserido = cep; }

    @Override
    protected CEP doInBackground (Void... voids) {
        StringBuilder resposta = new StringBuilder();
        if (this.cepInserido != null && this.cepInserido.
length() == 8) {
            try {
                URL url = new URL( spec: "https://viacep.com.br/ws/" + this.
cepInserido + "/json/");
                HttpURLConnection connection = (HttpURLConnection) url.
openConnection();
                connection.setRequestMethod("GET");
                connection.setRequestProperty("Content-type",
"application/json");
                connection.setRequestProperty("Accept", "application/
json");
                connection.setDoOutput(true);
                connection.setConnectTimeout(5000);
                connection.connect();

                Scanner scanner = new Scanner(url.openStream());

                while (scanner.hasNext()) {
                    resposta.append(scanner.next());
                }
            }catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return new Gson().fromJson(resposta.toString(), CEP.class);
    }
}

```

Note que o *MainActivity* é o objeto responsável pelos eventos e pela montagem das informações.

```
btnBuscarCep.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        try {  
            CEP retorno = new HttpService(txtCep.getText().toString().  
                trim()).execute().get();  
            lblResposta.setText(retorno.toString());  
        } catch (ExecutionException e) {  
            e.printStackTrace();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
});
```

Agora, é só executar o emulador e realizar os testes.



#PraCegoVer: na imagem, temos a tela de um celular com design do app “BuscaCEPApp”. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma caixa de texto “insira seu CEP” e, abaixo dela, um botão com o trecho escrito “buscar CEP”.



#PraCegoVer: na imagem, temos a tela de um celular com design do app “BuscaCEPApp”. Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma caixa de texto “04029000” e, abaixo dela, um botão com o trecho escrito “buscar CEP”. Abaixo disso, há dados de CEP, Logradouro, Complemento, Bairro, Cidadeee Estado.

Após realizado todos os passos apresentados no vídeo, temos nosso projeto testado e funcionando!

Até aqui, você estudou sobre os protocolos de comunicação e o carregamento de arquivos. Nesse sentido, introduzimos o programa API, o qual representa um conjunto de regras e normas voltadas à comunicação para a requisição HTTP.

Somado a isso, você compreendeu, a partir do aplicativo hipotético “BuscaCEPApp”, como ocorre a utilização do API e como empregar as linhas de código voltadas para gerá-la.

Com isso, a parte de serviços e manipulação de dados utilizando protocolos HTTP está finalizada. Agora, você manipulará os dados a partir de um banco de dados nativo do Android Studio, chamado SQLite.

## Banco de dados *SQLite*

Por vezes — quase sempre! — existe a necessidade de armazenarmos informações relevantes para os aplicativos. Diante disso, o Android Studio oferece um suporte nativo para os usuários, chamado *SQLite*.

Para melhor entender do que se trata um *SQLite*, ouça o *podcast* a seguir!



### **Podcast**

Confira o [podcast](#) sobre banco de dados.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! O *SQLite* do inglês *Structured Query Language*, ou Linguagem de Consulta Estruturada é um banco de dados livre que permite a manipulação de dados no aplicativo a partir de comandos SQL tradicionais.

Uma informação importante é que, por ser um banco de dados nativo da plataforma Android Studio, não tem a característica de troca de informações entre os aplicativos externos.

Além disso, ele apresenta segurança, pois apenas o aplicativo que criou o banco de dados poderá manipular suas informações.

Viu só como o suporte *SQLite* é útil para manter a segurança das informações dos aplicativos?

Assim fica mais fácil analisar o conceito e colocá-lo em prática, não é mesmo?

## Saiba mais



Para criar, alterar, excluir e consultar o banco de dados *SQLite* nativo, sugerimos a leitura do artigo [Salvar dados usando o SQLite](#), que traz informações complementares ao seu conhecimento!

Por mais que o banco de dados *SQLite* seja nativo no Android Studio, é importante conhecer sua estrutura.

A classe *SQLiteOpenHelper*, por exemplo, é a classe principal para a criação e manipulação de dados no aplicativo. Ela é responsável por criar a estrutura necessária no momento em que o aplicativo for iniciado.

Considerando isso, acompanhe a seguir a aplicação desse código:

```
package br.com.local.sqliteapp;

import ...

public class SQLDBHelper extends SQLiteOpenHelper {
    private static final String NOME_DO_BANCO =
"BDEmpregados";
    private static int VERSAO_DO_BANCO = 1;

    @Requires(Api(api = Build.VERSION_CODES.P))
    public SQLDBHelper (@Nullable Context context) {
        super(context, NOME_DO_BANCO, VERSAO_DO_BANCO,
openParams: null);

    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //Cria a estrutura do banco de dados

    }
    @Override
    public void onUpgrade (SQLiteDatabase db, int oldVersion,
int newVersion) {
        //Atualiza a estrutura do banco de dados
    }
}
```

Para exemplificar os processos de criação, alteração, inserção e consulta do banco de dados *SQLite* no aplicativo, veja o passo a passo abaixo: iniciando com a criação do banco de dados “empregados”, seguindo com a criação do *SQLite* e finalizando com a criação do banco de dados. Observe com atenção:

### Código 1

```
public static final String NOME _ BANCO _ DE _ DADOS =  
"bdEmpregados";
```

### Código 2

```
SQLiteDatabase meuBancoDeDados;
```

### Código 3

```
meuBancoDeDados = openOrCreateDatabase (NOME _ BANCO _ DE _  
DADOS, MODE _ PRIVATE, factory: null);
```

Para inserir os dados no banco de dados, será necessária a criação de uma tabela no aplicativo, conforme aplicado no código abaixo:

```
private void criarTabelaEmpregado() {  
    meuBancoDeDados.execSQL (  
        "CREATE TABLE IF NOT EXISTS funcionarios (" +  
        "id integer PRIMARY KEY AUTOINCREMENT," +  
        "nome varchar(200) NOT NULL," +  
        "departamento varchar(200) NOT NULL," +  
        "dataEntrada datetime NOT NULL," +  
        "salario double NOT NULL );"  
    );  
}
```

Para realizar consulta, inserção, exclusão e alteração, siga os próximos passos, de acordo com os códigos aplicados a seguir:

Consultando dados da tabela.

```
Cursor cursorEmpregados=meuBancoDeDados.rawQuery(sql:  
"SELECT * FROM funcionarios", selectionArgs:null);
```

Inserindo dados na tabela.

```
String insertSQL = "INSERT INTO funcionarios (" +  
    "nome, " +  
    "departamento, " +  
    "dataEntrada," +  
    "salario)" +  
    "VALUES (?, ?, ?, ?);";  
  
meuBancoDeDados.execSQL(insertSQL, new String[]{nome Empr,  
deptoEmpr, dataEntrada, salarioEmpr});  
  
Toast.makeText(getApplicationContext(), text: "Funcionário  
adicionado com sucesso!!!", Toast.LENGTH _ SHORT).show();
```

Excluindo dados da tabela.

```
String sql = "DELETE FROM funcionarios WHERE id = ?";  
  
meuBancoDeDados.execSQL (sql, new Integer[]{empregados.  
getId()});
```

Alterando dados da tabela.

```
String sql="UPDATE funcionarios SET nome= ?, departamento=  
?, salario= ? WHERE id= ?";  
meuBancoDeDados.execSQL(sql,  
    new String[]{nome, depto, salario, String.  
valueOf(empregados.getId())});  
Toast.makeText(mCtx, text "Empregado alterado com  
sucesso!!!", Toast.LENGTH _ LONG).show();
```

Desta forma, você conheceu o *SQLite*, banco de dados nativo do Android Studio e pôde aprender o processo de criação tendo em vista o protocolo HTTP, por meio de um tutorial.

Agora que os dados foram armazenados na aplicação, você adicionará o aplicativo no Google Play Store, a fim de distribuí-lo para a comunidade.

## Publicação do aplicativo no Google Play Store

No tópico anterior, você estudou sobre os aspectos relacionados ao banco de dados e a sua relação com o aplicativo *mobile*. Agora, vamos passar para uma etapa muito relevante associada à visibilidade e exposição ao mercado do *app* desenvolvido, a publicação no Google Play Store.

O Google Play Store é um portal que o Google oferece para disponibilizar seus aplicativos, a fim de que os usuários com dispositivos Android possam baixá-los. Dessa forma, existem inúmeras vantagens em optar pelo uso do Google Play.

Ouça o próximo *podcast* e conheça alguns desses benefícios:



### Podcast

Confira o [podcast](#) sobre publicação do aplicativo no Google Play Store.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Agora que você já criou seu aplicativo, chegou o momento de dar visibilidade e expor ao mercado do APP. Para isso, você pode utilizar o Google Play Store.

Neste portal você terá inúmeros benefícios, como: vender ou distribuir os aplicativos, assim como acesso a um maior público-alvo, visto que o dispositivo com Android tem, por padrão, o aplicativo que oferece o acesso ao Google Play Store. Terá também maior credibilidade por parte do usuário, o qual tem confiança em baixar os aplicativos do Google Play, em vez de precisar se arriscar em sites desconhecidos da internet. Terá uma maior segurança para o usuário, uma vez que os aplicativos cadastrados no Google Play são analisados contra códigos maliciosos.

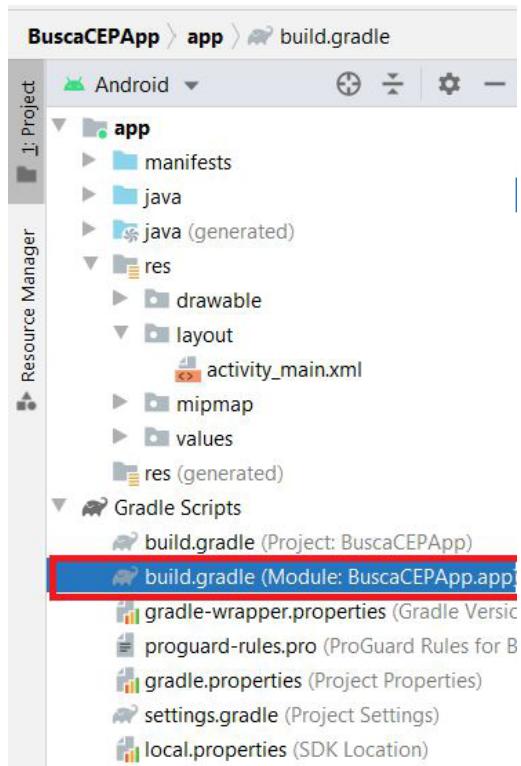
Além de ter compatibilidade garantida, pois o Google Play mostra para o usuário apenas os aplicativos que seu dispositivo com Android consegue executar, e facilidade de atualização, visto que o Google Play oferece um mecanismo para avisar ao usuário de que existe uma nova versão do aplicativo.

Viu só as diferentes vantagens que você terá com este portal? Agora você já pode conferir como publicar o seu aplicativo na plataforma.

Assim, você conheceu um pouco sobre esse portal e as vantagens que ele proporciona: a facilidade de comercializar aplicativos; o alcance de um público-alvo maior; a segurança que a plataforma oferece aos usuários para fazer os downloads dos *apps*; bem como as suas respectivas atualizações. No entanto, você deve estar se perguntando: “**como publicar o aplicativo na plataforma?**”. Veja no próximo tópico.

## Preparação do aplicativo para upload

Você viu as características principais do portal, bem como suas vantagens. Agora, abordaremos o processo de publicação do *app*. Para publicar o aplicativo no Google Play, você definirá a versão do aplicativo em um primeiro momento. Assim, conforme a imagem adiante, você precisa selecionar e clicar na opção “*build.gradle (Module: BuscaCEPApp.app)*” na aba *Gradle Scripts*.



#PraCegoVer: na imagem, temos o print do projeto na área de repositórios para conimagemr as versões criadas. O destaque vai para a opção "build.gradle (Module: BuscaCEPApp.app)".

Abaixo, estão listadas as opções disponíveis:

#### **Version code**

Utilizado pelo Google Play para instalar e atualizar o aplicativo.

#### **Version name**

Utilizado para mostrar ao usuário a versão do aplicativo.

#### **Min SDK version**

Deixa conimagemdo para uma menor versão que o aplicativo suportará.

**Target SDK version**

Utilizado na última versão para otimizar e ativar novos recursos para o dispositivo.

Abaixo, veja como você pode configurar o *build gradle*.

```
plugins
    id 'com.android application'
}

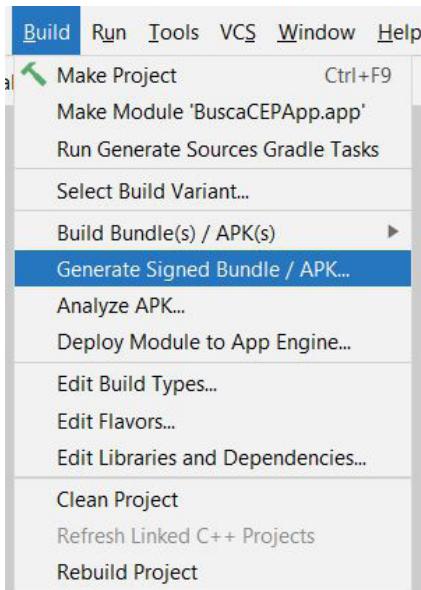
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "br.com. local buscacepapp"
        minSdkVersion 16
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx. test.runner.
AndroidJUnitRunner"
    }
}
```

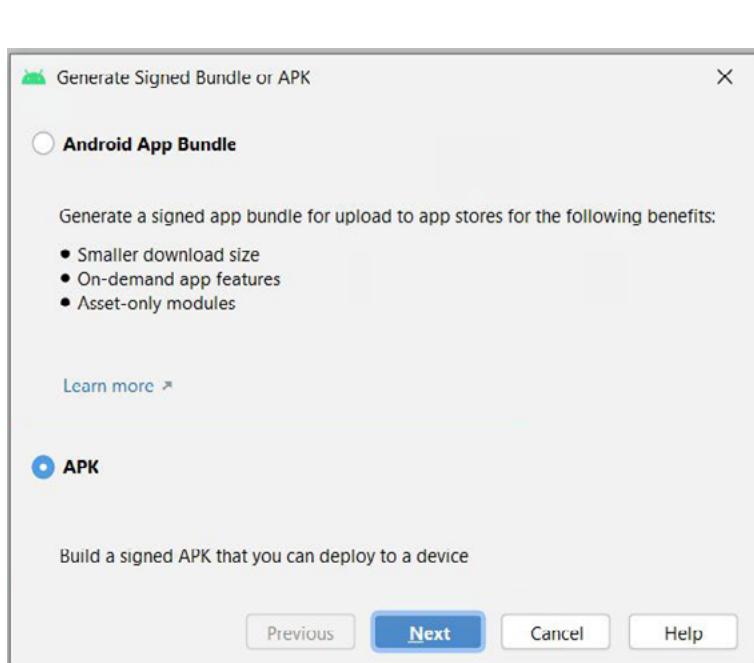
Feito isso, para criar o APK do aplicativo para publicação, siga os passos apresentados:

O primeiro é gerar o APK do aplicativo em questão.



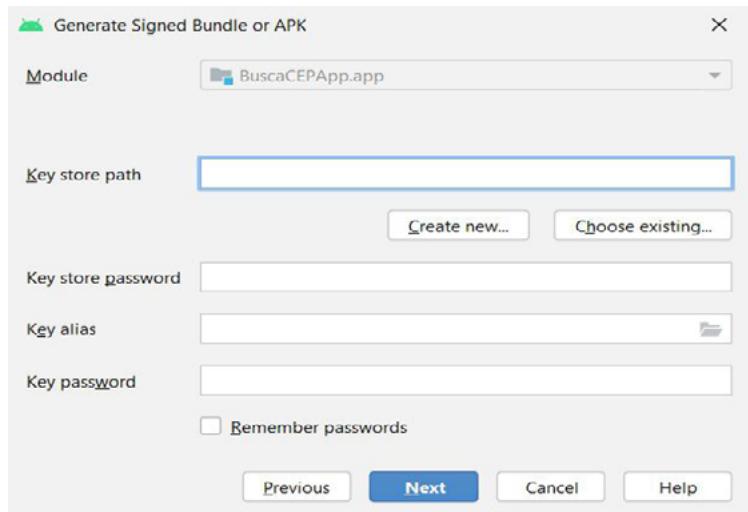
#PraCegoVer: na imagem, temos um print indicando a área para geração de chave APK. Está selecionada a opção "Build", seguida da opção "Generate Signed Bundle / APK...".

Na sequência, escolha o APK e selecione "Next".



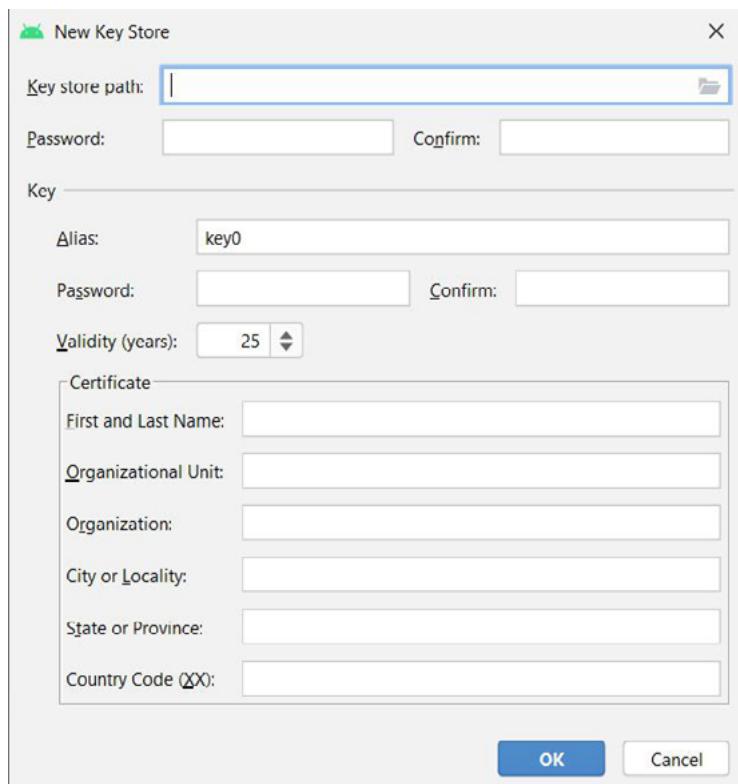
#PraCegoVer: na imagem, temos um print da opção "Generate Signed Bundle or APK", em que devemos selecionar a opção APK. Há três botões acessíveis abaixo, sendo "Next", "Cancel" e "Help", um do lado do outro, com o primeiro em destaque.

Depois, inicie a criação da chave de assinatura da APK.



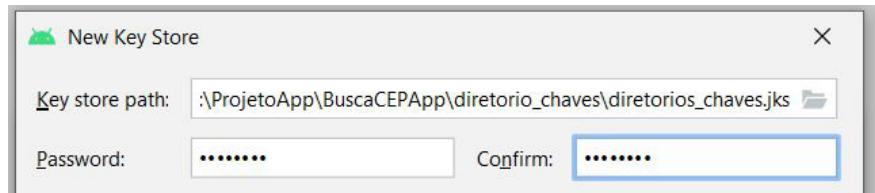
#PraCegoVer: na imagem, temos um print da opção "Generate Signed Bundle or APK", em que temos alguns campos em branco, como "Key store path". Há quatro botões acessíveis abaixo, sendo "Previous", "Next", "Cancel" e "Help", um do lado do outro, com o segundo em destaque.

Como você ainda não possui a chave, selecione a opção de criar uma nova para poder seguir com o processo adequadamente.



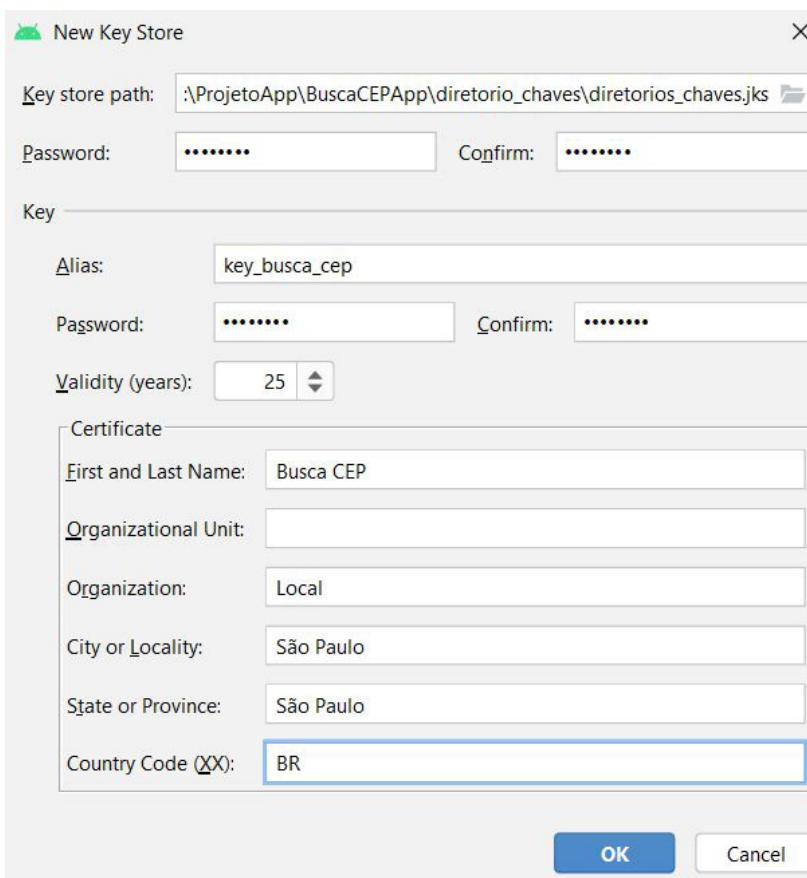
#PraCegoVer: na imagem, temos um print da opção "New Key Store", em que temos alguns campos em branco, como "Key store path". Há dois botões acessíveis abaixo, sendo "OK" e "Cancel", um do lado do outro, com o primeiro em destaque.

Indicamos, então, o caminho para salvar o repositório de chaves. O diretório será inserido no local que você julgar necessário e deverá ser fechado com uma senha, que não deverá ser esquecida.



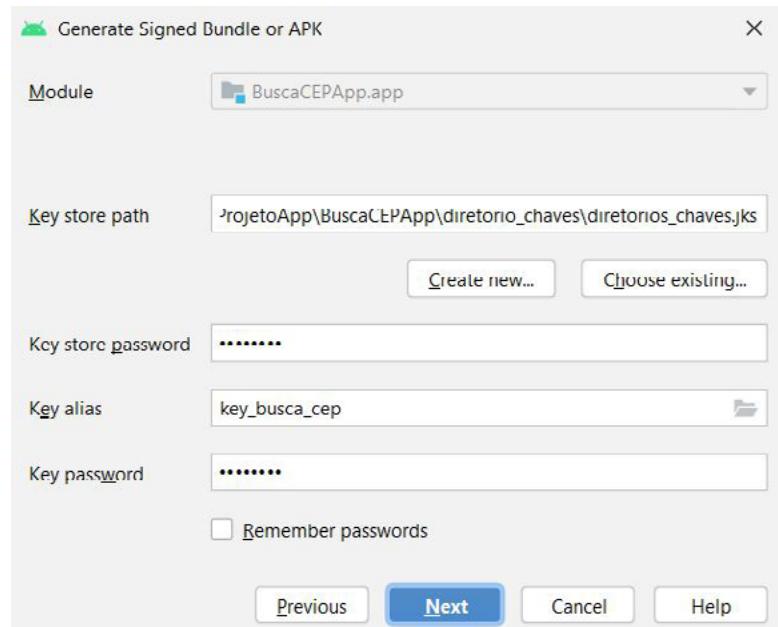
#PraCegoVer: na imagem, temos um print da opção "New Key Store", em que temos alguns campos preenchidos, como "Key store path", "Password" e "Confirm".

Após a definição do local da chave, informe a chave que você irá criar. Vale ter muita atenção neste momento, pois a chave criada será utilizada para o lançamento do aplicativo, sendo que o Google Play verificará quando quiser alterar o aplicativo publicado. Por isso, guarde a senha de criação da chave com cuidado. Depois, preencha os campos e selecione "OK".



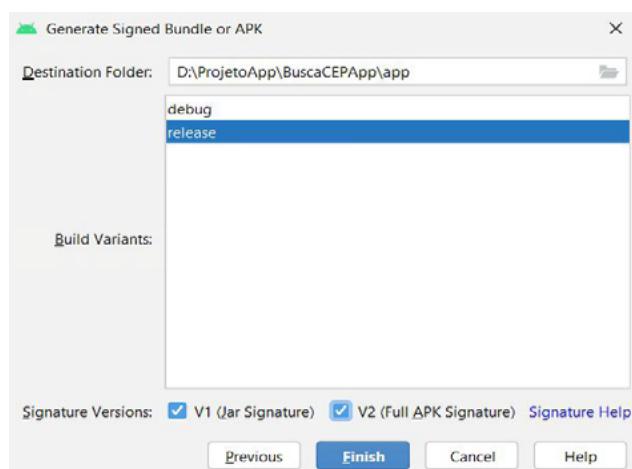
#PraCegoVer: na imagem, temos um print da opção "New Key Store", em que temos alguns campos já preenchidos, como "Key store path", "Password" e "Confirm". Há dois botões acessíveis abaixo, sendo "OK" e "Cancel", um do lado do outro, com o primeiro em destaque.

Agora selecione “OK” para adicionar a chave ao projeto.



#PraCegoVer: na imagem, temos um print da opção “Generate Signed Bundle or APK”, em que temos alguns campos já preenchidos, como “Key store path”, “Key store password”, “Key alias” e “Key password”. Há quatro botões acessíveis abaixo, sendo “Previous”, “Next”, “Cancel” e “Help”, um do lado do outro, com o segundo em destaque.

Escolha a pasta de destino da APK e o tipo de *build*. Se o caso foi *release*, selecione as duas opções de “Signature Versions”. Por fim, selecione o botão “Finish”, conforme a imagem abaixo:



#PraCegoVer: na imagem, temos um print da opção “Generate Signed Bundle or APK”, em que temos a opção “Destination Folder”, “Build Variants” (com destaque para “release”) e “Signature Versions”. Há quatro botões acessíveis abaixo, sendo “Previous”, “Finish”, “Cancel” e “Help”, um do lado do outro, com o segundo em destaque.

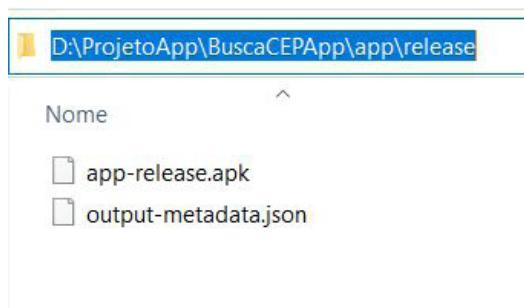
O projeto irá gerar a APK e informar o local.

#### Generate Signed APK

APK(s) generated successfully for module 'BuscaCEPApp.app' with 1 build variant:  
Build variant 'release': [locate](#) or [analyze](#) the APK.

#PraCegoVer: na imagem, temos um print da mensagem de geração de APK, indicando local.

Selecione "locate", em que será direcionado ao local de criação.



#PraCegoVer: na imagem, temos um print do local de criação.

Assim, o APK está pronto para ser publicado no Google Play Store!

Neste tópico você conheceu os dez passos para a publicação do aplicativo no respectivo portal do Google.

No próximo tópico, você aprenderá sobre o processo seguinte à publicação, ou seja, a distribuição do aplicativo na plataforma.

## Distribuição via Google Play

Para publicar aplicativos no Google Play Store, é preciso ter uma conta cadastrada no Google. Além disso, será necessário seguir alguns passos.

Inicialmente, deve-se criar um perfil de desenvolvedor pelo [Google Play Console](#), que é a página inicial de cadastro do perfil de desenvolvedor. Para isso, veja a imagem adiante.



## Google Play Console

Publique seus apps e jogos com o Google Play Console e expanda seus negócios no Google Play. Aproveite os recursos que ajudarão você a melhorar a qualidade do seu app, engajar seu público, gerar receita e muito mais.

[FAZER LOGIN](#)[SAIBA MAIS](#)

#PraCegoVer: na imagem, temos um print do site do Google Play Console, em que aparece o título, seu enunciado e dois botões, sendo um de "FAZER LOGIN" e outro de "SAIBA MAIS". Para fazer o cadastro, clique na primeira opção.

Depois de acessar o link do cadastro, registre os seus dados, para gerar a sua conta de desenvolvedor, conforme a imagem a seguir:

### Criar uma nova conta de desenvolvedor

A Conta do Google selecionada será proprietária desta nova conta de desenvolvedor. Caso esteja tentando entrar em uma conta de desenvolvedor existente, solicite um convite do administrador.

Se você for de uma organização, não recomendamos usar uma conta pessoal para configurar contas de desenvolvedor. É possível configurar Contas do Google com qualquer endereço de e-mail existente.  
[Saiba mais](#)

Para criar uma conta, é necessário pagar a taxa única de registro de US\$ 25. Para concluir o registro da conta, talvez seja preciso verificar sua identidade com um documento de identificação válido. Se não conseguirmos fazer isso, a taxa de registro não será reembolsada.

Nome público do desenvolvedor \*

Isso ficará visível para os usuários no Google Play  
0 / 50

Endereço de e-mail secundário para contato \*

Esse endereço poderá ser usado para entrar em contato com você, além do e-mail associado à Conta do Google. Ele não ficará visível para usuários do Google Play.

Número de telefone para contato \*

Inclua o símbolo +, o código do país e o código de área. Usaremos essa informação para entrar em contato com você, mas ela não ficará visível para os usuários no Google Play.

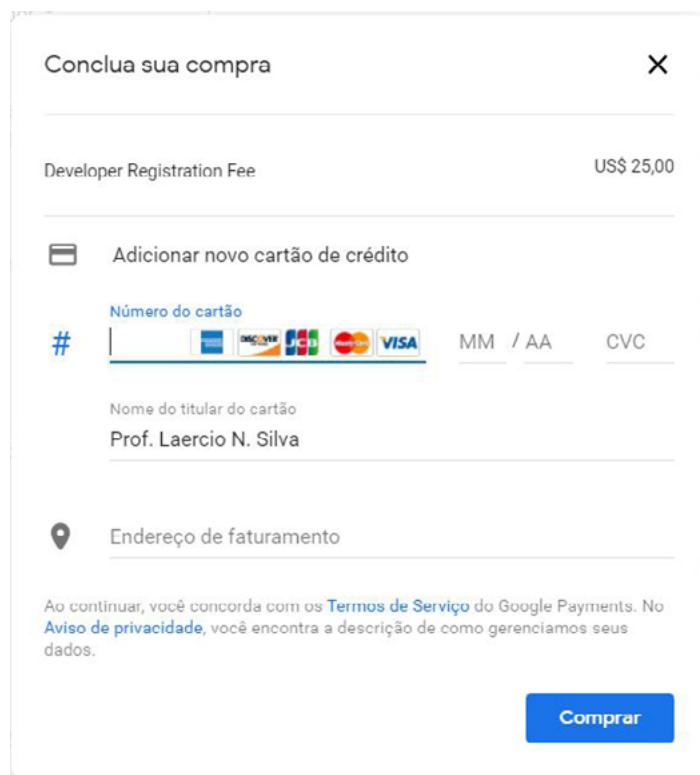
Contrato para desenvolvedores e Termos de Serviço \*

Confirme que li e concordo com o [Contrato de distribuição para desenvolvedores do Google Play](#). Aceito associar minha Conta do Google a esse contrato e confirmo que sou maior de 18 anos de idade.

Confirme que li e concordo com os [Termos de Serviço do Google Play Console](#). Aceito associar minha Conta do Google a esses Termos de Serviço.

#PraCegoVer: na imagem, temos um print do site do Google Play Console para criação de nova conta de desenvolvedor. Há uma breve explicação e alguns campos em brancos para preenchimento, como nome público do desenvolvedor e número de telefone para contato. No fim, encontramos dois termos que devem ser assinalados.

Leia os termos do contrato. É necessário concordar com o contrato de distribuição do desenvolvedor e pagar uma taxa de registro (US\$ 25,00) com cartão de crédito (usando o Google Checkout). Vale lembrar que a taxa é única, ou seja, será preciso pagar somente no momento do cadastro.



#PraCegoVer: na imagem, temos um *print* do site do Google Play Console para pagamento da assinatura. Há o valor de registro e as opções para inclusão de dados de compra. No fim, à direita, encontramos um botão de "Comprar".

Após seguir os passos, é possível acessar o site do Google Play e publicar o aplicativo no [Google Play Console](#), selecionando “Fazer Login” para acessar. Para preencher os detalhes da ficha do aplicativo, considere as seguintes características:

### Detalhes do produto

Utilizar palavras-chave que sejam relevantes para descrever o aplicativo.

### Recursos gráficos

Imagens e vídeos que tragam informações relevantes e as principais funcionalidades do aplicativo.

### Idioma e traduções

Caso o aplicativo seja multi-idiomas.

### Categorização

Tipo de categoria do aplicativo que mais se encaixa.

### Detalhes do contato

O e-mail é obrigatório, mas usar outros tipos de contatos para dar suporte ao usuário do aplicativo pode ser interessante.

### Política de privacidade

Determina a segurança das informações do usuário e dá garantia quanto ao uso adequado dessas informações.

Desta forma, você conheceu a etapa de divulgação da aplicação *mobile*, a qual é realizada a partir da criação da conta no Google Play Store.

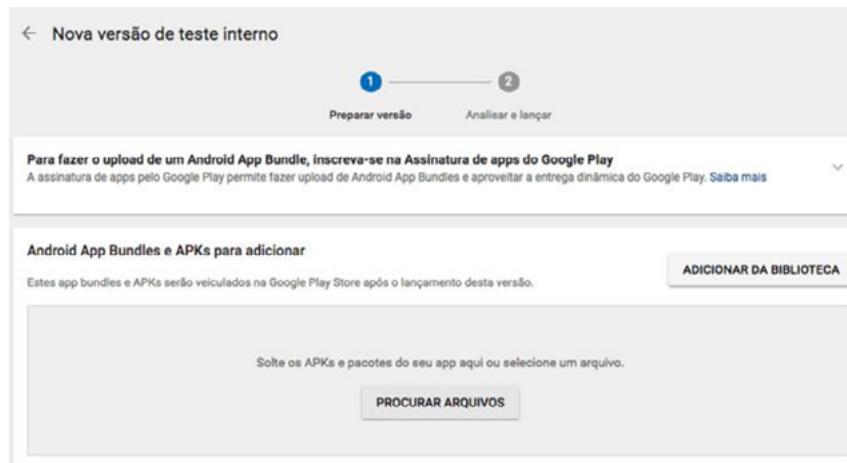
Com tudo isso pronto, é possível enviar o APK.

## Upload do APK

Você enviará agora o APK (*Android Package Kit*), que se caracteriza como a extensão do arquivo que o sistema operacional Android utiliza na distribuição e instalação. De modo geral, ele contém os elementos necessários para que um aplicativo funcione corretamente no dispositivo do usuário.

Para realizar esse upload, acesse o gerenciador de versão, escolha o tipo de *release*, testes fechado e aberto, bem como a versão de produção. Assim que as opções forem escolhidas, basta selecionar “Criar Versão”, escolhendo o APK para upload, criado anteriormente, e nomeando a versão.

Após isso, a tela apresentada abaixo aparecerá para você:



#PraCegoVer: na imagem, temos um *print* de nova versão de teste interno, em que encontrarmos as etapas de preparar versão, analisar e lançar. Além disso, existem trechos escritos na tela explicando o processo de upload do APP na plataforma, além de um botão para procurar arquivos.

Depois, classifique o conteúdo, o preço (dependendo da estratégia de monetização) e a disponibilidade por região. Na seção de gerenciamento de versão, selecione “Revisar e Publicar”. Com isso, finalmente o aplicativo estará pronto para os usuários utilizarem!

## Saiba mais



Caso precise de mais informações para a publicação de um aplicativo, sugerimos a leitura do artigo [Publicar seu App](#), que também nos traz um passo a passo detalhado. Além disso, para garantir que o aplicativo será lançado sem problemas, não deixe de conferir a [Lista de Verificação do Lançamento!](#)

No primeiro tópico deste módulo, você conheceu a API e o banco de dados específico do Android Studio, o *SQLite*, entendendo a partir de um tutorial, como iniciar e finalizar a criação dele.

Depois, você estudou sobre as características principais e as vantagens que o portal Google Play Store proporciona. Depois disso, passou a conhecer o passo a passo de como fazer a publicação nesse ambiente.

Por fim, você conheceu a etapa associada à divulgação do aplicativo mobile e o processo de envio do APK (*Android Package Kit*), ou seja, os elementos necessários para que o *app* funcione adequadamente.

# Fechamento

Parabéns!

Você finalizou o curso Desenvolvendo Aplicações *Mobile* com o Android Studio. Aqui você aprendeu o que é um aplicativo Android, os tipos de layouts existentes, os componentes, as ações, a troca de mensagens, a manipulação de arquivos, a utilização de protocolos HTTP, o banco de dados *SQLite* e, por fim, colocou o aplicativo no Google Play Store. Vale lembrar que as empresas precisam cada vez mais de profissionais qualificados e que buscam conhecimentos para resolver problemas diversos.

Sempre que precisar, você pode retornar e rever o curso.

Boa sorte em sua jornada. Até a próxima!

# Referências

ANDROID. **Material Design**, [s. l.], [s. d.]. Disponível em: <https://material.io/develop/android>. Acesso em: 9 mar. 2021.

ANDROID Studio. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/>. Acesso em: 28 fev. 2021.

APLICATIVO híbrido ou nativo? [S. l.], 17 maio 2019. Publicado pelo canal TecnoSpeed TI. 1 vídeo (2 min). Disponível em: <https://www.youtube.com/watch?v=CMLk0Y-9bUc>. Acesso em: 11 maio 2021.

COMO posso habilitar a virtualização (VT) no meu PC? **BlueStacks**, [s. l.], 11 nov. 2020. Disponível em: <https://support.bluestacks.com/hc/pt-br/articles/115003174386-Como-posso-habilitar-a-virtualiza%C3%A7%C3%A3o-VT-no-meu-PC->. Acesso em: 12 maio 2021.

COMPONENTS. **Material Design**, [s. l.], [s. d.]. Disponível em: <https://material.io/components?platform=android>. Acesso em: 12 maio 2021.

CONImagemR a aceleração de hardware para o Android Emulator. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/studio/run/emulator-acceleration?hl=pt-br>. Acesso em: 14 maio 2021.

CONHEÇA o Android Studio. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 12 maio 2021.

CORDEIRO, F. Como dominar os Android layouts em 07 passos. **AndroidPro**, [s. l.], [s. d.]. Disponível em: <https://www.androidpro.com.br/blog/desenvolvimento-android/android-layouts-viewgroups-intro/>. Acesso em: 12 maio 2021.

CRIAR e executar o app. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/studio/run?hl=pt-br>. Acesso em: 14 maio 2021.

CRIAR um projeto para Android. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/training/basics/firstapp/creating-project>. Acesso em: 14 maio 2021.

ANDROID. **Material Design**, [s. /.], [s. d.]. Disponível em: <https://material.io/develop/android>. Acesso em: 9 mar. 2021.

ANDROID Studio. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/>. Acesso em: 28 fev. 2021.

APLICATIVO híbrido ou nativo? [S. /.], 17 maio 2019. Publicado pelo canal TecnoSpeed TI. 1 vídeo (2 min). Disponível em: <https://www.youtube.com/watch?v=CMLk0Y-9bUc>. Acesso em: 11 maio 2021.

COMO posso habilitar a virtualização (VT) no meu PC? **BlueStacks**, [s. /.], 11 nov. 2020. Disponível em: <https://support.bluestacks.com/hc/pt-br/articles/115003174386-Como-posso-habilitar-a-virtualiza%C3%A7%C3%A3o-VT-no-meu-PC->. Acesso em: 12 maio 2021.

COMPONENTS. **Material Design**, [s. /.], [s. d.]. Disponível em: <https://material.io/components?platform=android>. Acesso em: 12 maio 2021.

CONImagemR a aceleração de hardware para o Android Emulator. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/studio/run/emulator-acceleration?hl=pt-br>. Acesso em: 14 maio 2021.

CONHEÇA o Android Studio. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 12 maio 2021.

CORDEIRO, F. Como dominar os Android layouts em 07 passos. **AndroidPro**, [s. /.], [s. d.]. Disponível em: <https://www.androidpro.com.br/blog/desenvolvimento-android/android-layouts-viewgroups-intro/>. Acesso em: 12 maio 2021.

CRIAR e executar o app. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/studio/run?hl=pt-br>. Acesso em: 14 maio 2021.

CRIAR um projeto para Android. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/training/basics/firstapp/creating-project>. Acesso em: 14 maio 2021.

CRIAR uma notificação básica. **Developers**, [s. /.], [s. d.]. Disponível em: <https://developer.android.com/training/notify-user/build-notification#SimpleNotification>. Acesso em: 12 maio 2021.

CRIAR uma nova conta de desenvolvedor. **Google Play Console**, [s. /.], [s. d.]. Disponível em: <https://play.google.com/console/signup>. Acesso em: 12 maio 2021.

GOOGLE. **Criar e conimagemr seu app.** 2021. Disponível em: <https://support.google.com/googleplay/android-developer/answer/9859152?hl=pt-BR>. Acesso em: 28 fev. 2021.

GOOGLE Play Console. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/distribute/console?hl=pt-br>. Acesso em: 12 maio 2021.

LISTA de verificação do lançamento. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/distribute/best-practices/launch/launch-checklist?hl=pt-br>. Acesso em: 12 maio 2021.

O CONCEITO de atividades. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/guide/components/activities/intro-activities?hl=pt-br#tcoa>. Acesso em: 12 maio 2021.

PUBLICAR seu app. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/studio/publish?hl=pt-br>. Acesso em: 12 maio 2021.

SALVAR dados usando o SQLite. **Developers**, [s. l.], [s. d.]. Disponível em: <https://developer.android.com/training/data-storage/sqlite?hl=pt-br#java>. Acesso em: 12 maio 2021.

QUANT-UX. Disponível em: <https://www.quant-ux.com/#/>. Acesso em: 9 mar. 2021.

VIACEP. Disponível em: <https://viacep.com.br/>. Acesso em: 28 fev. 2021.

