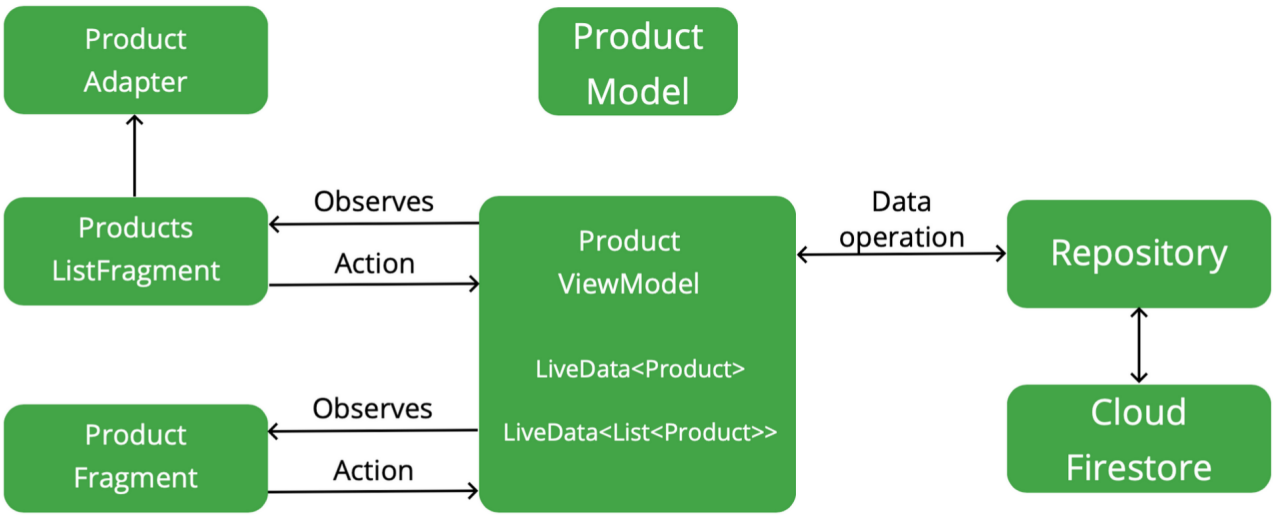
Primeira edição

siecola.com.br

Desenvolvimento Android com Firebase



PAULO SIECOLA

Desenvolvimento Android com Firebase

Desenvolva apps Android em Kotlin com o Firebase e serviços como Authentication, Firestore, Analytics e Remote Config

Paulo Cesar Siecola

Esse livro está à venda em http://leanpub.com/androidcloud

Essa versão foi publicada em 2022-04-13

Para Matilde, Doralice, Hannah e Clotilde, minhas adoráveis cachorras loucas!

Conteúdo

1 - Introdução

Bem vindo ao livro **Desenvolvimento Android com Firebase**! Com ele, o leitor construirá aplicativos para Android utilizando as mais modernas técnicas, tecnologias e arquiteturas existentes. Além disso, utilizará o **Google Firebase**, uma poderosa plataforma de *cloud computing* que oferece, dentre outras coisas, mecanismos de autenticação, banco de dados, registos de eventos e muito mais!

O leitor que decidiu por esse livro já deve saber da importância que aplicativos para dispositivos móveis representam na vida das pessoas, bem como da demanda pelas empresas por desenvolvedores com habilidades nessa área.

Seja por *hobby* ou para alcançar uma vaga de emprego tão sonhada, aprender a desenvolver aplicativos para Android pode ser **desafiador** e **divertido**!

1.1 - A quem se destina esse livro

O público alvo desse livro são desenvolvedores com conhecimento em programação orientada a objetos, que desejam conhecer e desenvolver aplicativos **Android** utilizando serviços do **Google Firebase**.

Todos os aplicativos desenvolvidos aqui serão criados utilizando a linguagem **Kotlin**, criada pela JetBrains. Ter conhecimento nessa linguagem é desejável, mas a estrutura didática do livro considera que o leitor não tem nenhuma experiência prévia. Por isso, alguns conceitos importantes sobre Kotlin serão apresentados ao longo do livro, principalmente aqueles mais utilizados para o desenvolvimento de aplicativos para Android.

Apesar desse livro ser voltado a conceitos e arquiteturas não triviais no âmbito do desenvolvimento de aplicativos para Android, ele não considera como requisito um conhecimento prévio do leitor nesse assunto. O mesmo se aplica ao Google Firebase. Por isso, seu conteúdo passará por assuntos básicos até chegar ao seu propósito final, como pode ser visto em detalhes na seção 1.4 desse capítulo.

1.2 - O que é necessário para desenvolver aplicativos para Android

Para aproveitar esse livro em sua totalidade, acompanhar os exercícios práticos e desenvolver os exercícios propostos, será necessário utilizar a IDE Android Studio. Nesse link[1] é possível visualizar os requisitos mínimos necessários da máquina de desenvolvimento.

[1] https://developer.android.com/studio

Também será necessário um dispositivo Android com versão 5.0 - Lollipop ou superior. Caso não possua um aparelho com Android, é possível criar um dispositivo emulado, porém, os requisitos da máquina de desenvolvimento deverão ser maiores, principalmente em termos de memória RAM.

Também será necessário criar uma conta no Firebase[2], para criação dos recursos que serão utilizados nele.

O capítulo 3 irá detalhar o processo de preparação da máquina de desenvolvimento, instalando as ferramentas e configurando-as.

Todas as ferramentas e contas necessárias para o acompanhamento desse livro podem ser obtidas ou criadas de forma gratuita.

1.3 - Estrutura didática do livro

A estrutura didática desse livro permeia um conceito conhecido como **aprendizado baseado em problemas**, onde o leitor é apresentado e conduzido aos conceitos chaves através de problemas que devem ser resolvidos utilizando tais conhecimentos.

Obviamente, nem todos os conceitos podem ser apresentados dessa forma, mas para aqueles que permitem, será utilizado uma abordagem mais prática de aprendizado, fazendo com que o leitor sempre tenha em mente um problema a ser resolvido utilizando a tecnologia que é detalhadamente apresentada. Dessa forma, tendo-se sempre em mente o problema alvo a ser resolvido, o leitor pode absorver os conceitos que são apresentados de forma mais eficiente.

1.4 - Capítulos do livro

Os capítulos serão apresentados da seguinte forma:

O **capítulo 2** apresenta um pouco sobre o sistema operacional Android e o **capítulo 3** mostra conceitos por trás da plataforma Firebase que será utilizada pelos aplicativos desenvolvidos nesse livro.

Apesar desse ser um livro totalmente voltado à prática, é importante apresentar alguns conceitos iniciais sobre as plataformas que serão utilizadas, para que o leitor tome conhecimento sobre tudo o que será utilizado, antes de mergulhar no código.

O **capítulo 4** instrui o leitor a preparar seu ambiente de desenvolvimento, no processo de instalação das ferramentas e bibliotecas necessárias, bem como suas configurações.

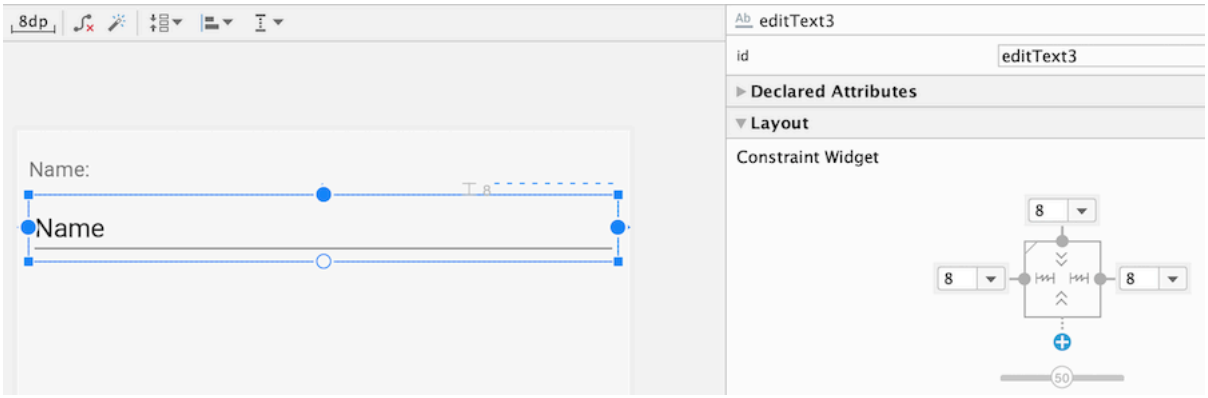
O processo de preparação do ambiente de desenvolvimento pode ser bem demorado. Se desejar, pule para o **capítulo 4** e comece esse trabalho, antes de continuar sua leitura.

[2] http://firebase.google.com/

No **capítulo 5** são apresentados alguns conceitos da linguagem Kotlin, focado no que será utilizado para o desenvolvimentos dos aplicativos nesse livro.

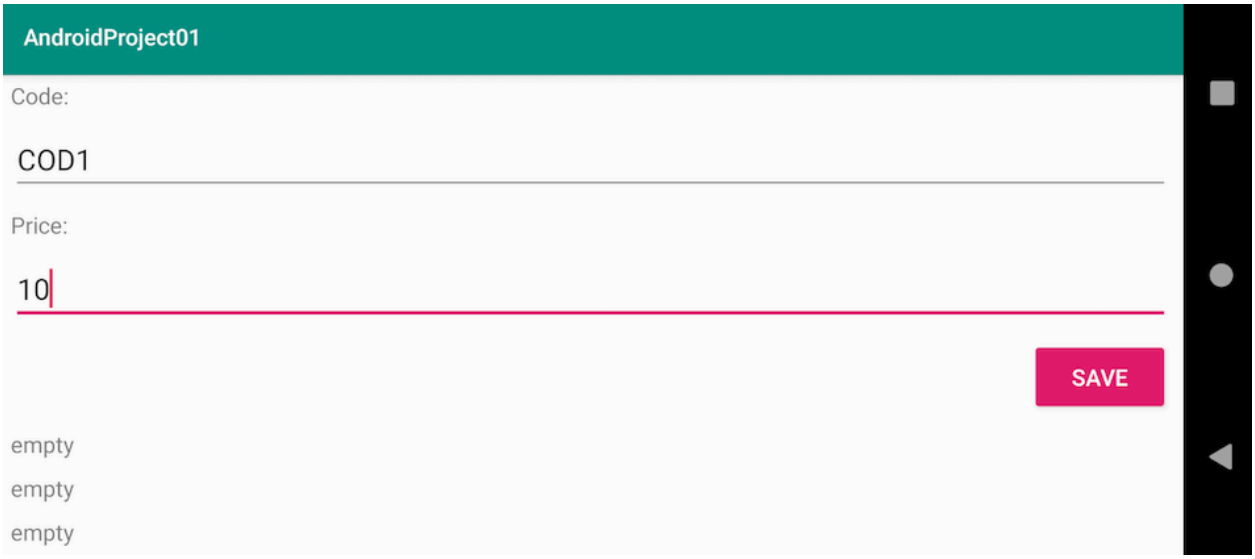
```
var dogs = listOf("Matilde", "Doralice", "Hannah", "Clotilde")
for (dog in dogs) {
        println("New dog is: ${dog}")
}
```

No **capítulo 6** começa o desenvolvimento de um aplicativo bem simples, somente com o intuito de demonstrar a ferramenta Android Studio, a linguagem Kotlin e a estrutura de um projeto inicial, utilizando `ConstraintLayout` para a construção da interface gráfica:



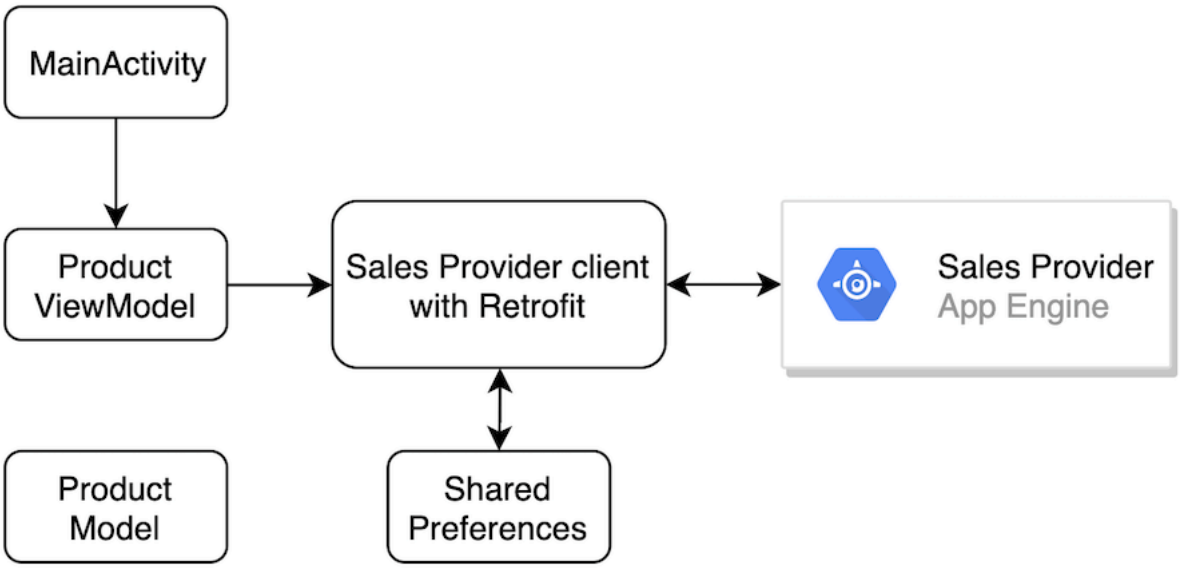
Constraints do EditText

Ainda com o mesmo aplicativo desenvolvido, o **capítulo 7** apresenta alguns desafios do mundo de dispositivos móveis e como resolvê-los com técnicas simples. Aqui, serão apresentados conceitos como `Activity`, seu ciclo de vida e como lidar com seus estados.



Alteração de configuração do dispositivo

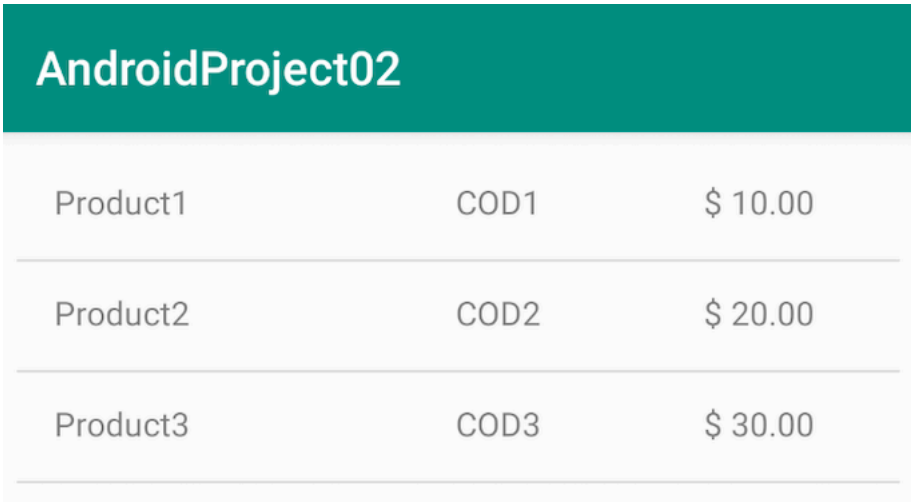
O **capítulo 8** inicia um novo aplicativo para consumir serviços REST com OAuth utilizando o framework Retrofit.



Arquitetura do projeto

Nele também serão utilizados conceitos como `ViewModel` e Kotlin coroutines.

O **capítulo 9** continua o segundo aplicativo, introduzindo conceitos de listas criadas com o componente `RecyclerView`, assim como a navegação entre telas utilizando o `NavigationController`, que permite a utilização de *safe args* entre fragmentos.



Tela inicial com a lista de produtos

No **capítulo 10** um novo aplicativo será criado para receber mensagens através do **Firebase Cloud Messaging**, dando início a parte do livro que trata sobre a interação de aplicações Android com a plataforma Firebase.

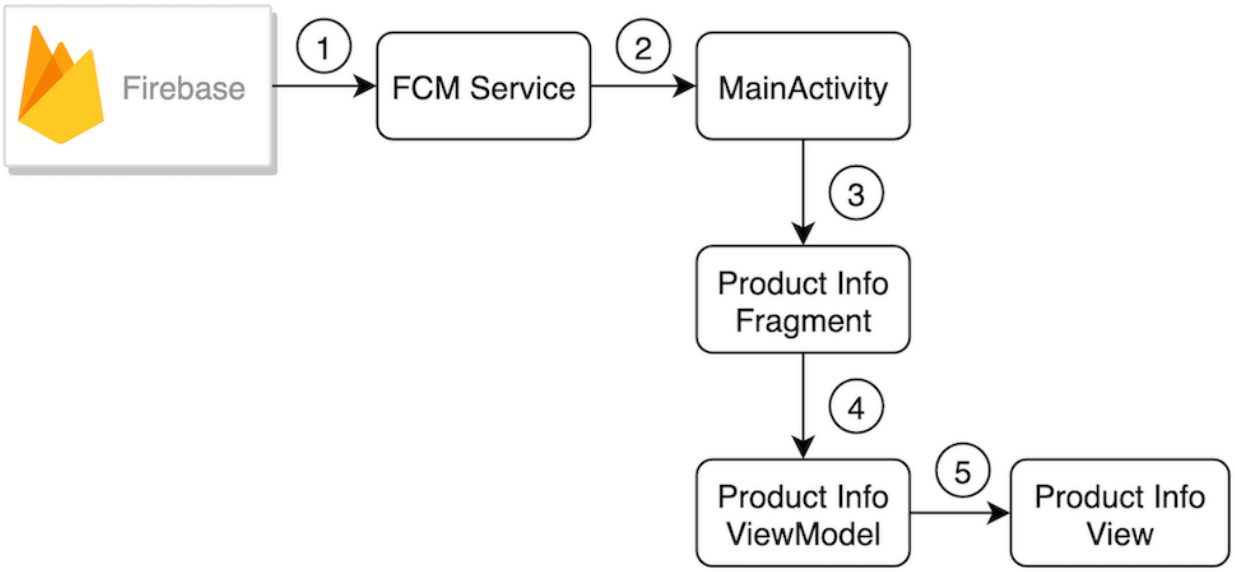
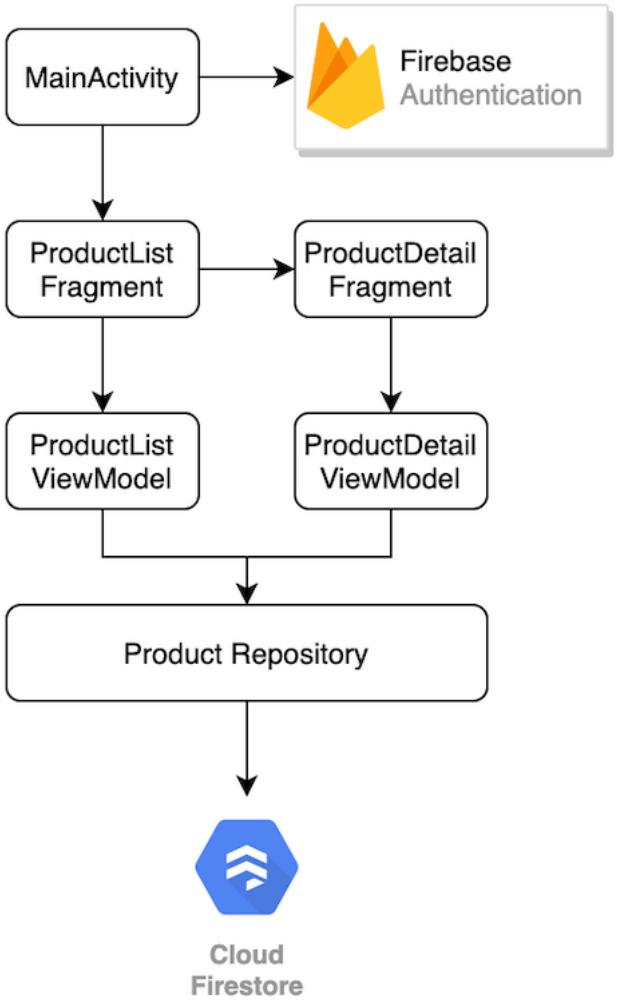


Diagrama do projeto

Com o Firebase Cloud Messaging é possível enviar notificações aos dispositivos, utilizando essa plataforma do Google.

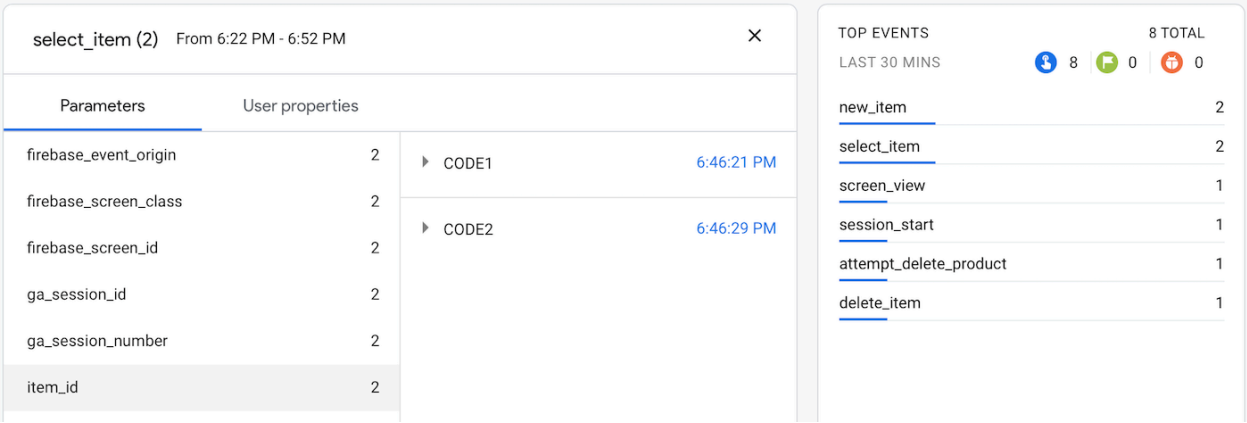
O **capítulo 11** inicia um novo aplicativo para utilizar o **Firebase Authentication**, para autenticação de usuários utilizando a conta do Google, sem a necessidade da criação de infraestruturas complexas.



Arquitetura do projeto

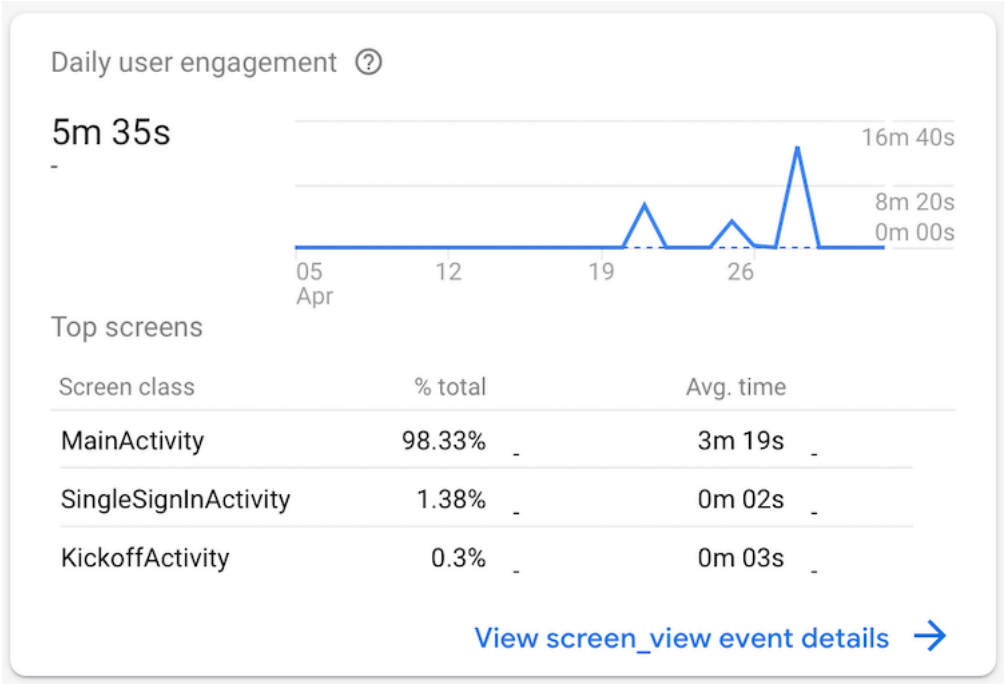
Nessa aplicação também será utilizado o **Firestore**, que é um banco de dados não-relacional na nuvem, que permite o armazenamento de coleções de documentos, com sincronismo em tempo real entre dispositivos.

O **capítulo 12** utiliza uma ferramenta extremamente poderosa para o entendimento do comportamento do usuário com um aplicativo: **Google Analytics for Firebase**. Com ele é possível gerar eventos da utilização de cada parte da aplicação, concentrando em um dashboard analítico.



Relatório de eventos

Com o Googe Analytics for Firebase também é possível obter informações do engajamento dos usuários, bem como informações sobre localização, modelos de dispositivos e versões de sistema operacional.



Engajamento de usuários

Finalmente o **capítulo 13** introduz o **Firebase Remote Config**, que possibilita a alteração de partes do aplicativo mediante variáveis de configuração que podem ser alteradas no console do Firebase. Isso faz com que uma série de possibilidades possam ser criadas, desde a liberação de funcionalidades a um pequeno grupo de usuários, testes A/B, gratificações para usuários VIPs ou até mesmo a mudança do comportamento da aplicação. Tudo isso sem que haja a necessidade da publicação de uma nova versão do aplicativo.

Está pronto para embarcar nessa jornada pelo mundo do Android? Então vamos lá que o livro está repleto de novidades a serem descobertas por você!

2 - Sobre o sistema operacional Android

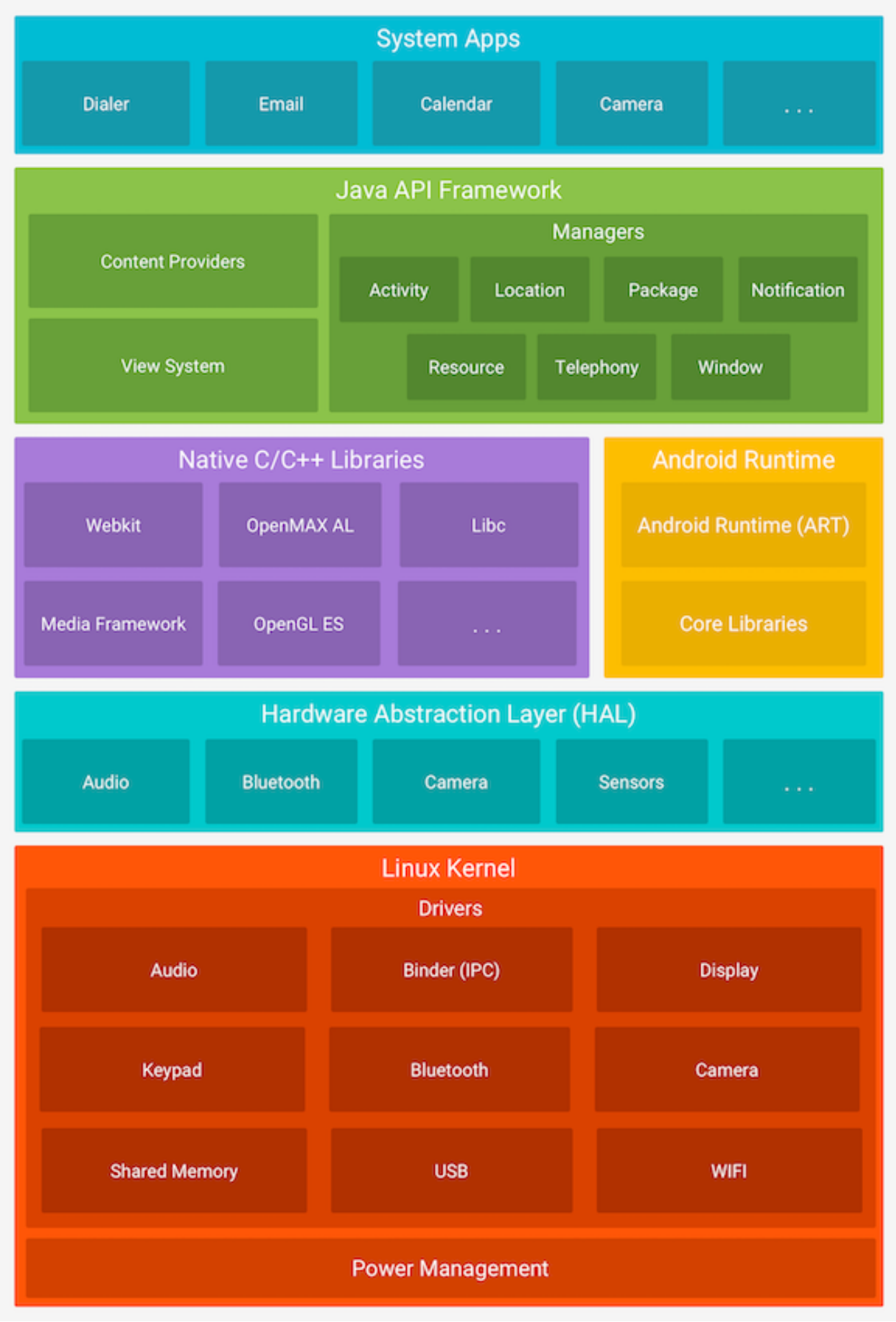
O Android é um sistema baseado em Linux, de código fonte aberto e capaz de ser utilizado em uma variedade de dispositivos, como *smartphones*, *tablets*, relógios, aparelhos de TV e de entretenimento em carros.

Ao longos dos anos o sistema Android tem se tornado complexo e cheio de funcionalidades, tanto para desenvolvedores como para usuários.

Tal complexidade traz um desafio em particular para aqueles que se aventuram no mundo do desenvolvimento de aplicativos para esse ecossistema.

2.1 - Um sistema feito em camadas

O sistema Android é feito em camadas, como é exemplificado pela figura a seguir:



Camadas do Android - Fonte: developer.android.com

Felizmente, não é necessário conhecer todas as as camadas, tão pouco todas as suas partes em profundidade, porém, é interessante ao desenvolvedor saber que elas existem e como estão conectadas entre si. Para aqueles que desejam se aprofundar nessa arquitetura, existe uma documentação[3] interessante que aborda esse assunto.

Basicamente, o conteúdo desse livro se concentra na primeira camada, chamada de **System Apps**, onde os aplicativos do próprio Android estão, como calendário, câmera e outros, e também onde estão os que o próprio usuário instala.

Para o desenvolvedor que vai criar aplicativos para o usuário final de Android, é importante ter um bom conhecimento sobre os recursos utilizados da camada seguinte, chamada **Java API Framework**, uma vez que é ela quem provê toda a interface de interação com o sistema Android. Obviamente não é necessário conhecer todas as partes dessa camada, mas sim estar a par dos recursos que serão utilizados dela.

À medida em que os conteúdos desse livro forem sendo apresentados, as partes significantes da API (Application Programming Interface) do Android serão detalhadas quando necessário.

2.2 - Android virtual device

Como parte das ferramentas necessárias para o desenvolvimento de aplicativos Android, que serão detalhadas no capítulo 4, está o Android virtual device, ou **AVD**, que é um dispositivo que pode ser emulado em um computador com Windows, Linux ou MacOS. Com ele é possível depurar e testar aplicativos em tempo de desenvolvimento, simulando várias situações e ambientes, sem a necessidade de um dispositivo real.

Embora essa abordagem exija um máquina de desenvolvimento mais potente, ela pode ser interessante para evitar a necessidade de vários dispositivos para testes em tempo de codificação.

2.3 - Como desenvolver aplicativos para o sistema Android

Existem algumas possibilidades para o desenvolvimento de aplicativos para Android, mas elas podem ser concentradas em duas grandes áreas:

a) Desenvolvimento híbrido: nessa modalidade é possível reaproveitar partes do projeto para criar aplicativos para outros sistemas, como o iOS. Isso traz uma agilidade maior no processo de desenvolvimento de produtos que almejam alcançar um número maior de usuários em várias plataformas. Porém, essa abordagem pode trazer complicações em aplicativos complexos ou que necessitem de interações profundas com o sistema operacional.

b) Desenvolvimento nativo: essa abordagem utiliza as ferramentas, linguagens e frameworks oficiais do Android, oferecidos pela própria Google. O código fonte e os projetos criados nessa

[3] https://developer.android.com/guide/platform

modalidade não pode ser reaproveitados para outras plataformas. Essa abordagem acaba fazendo com que empresas tenham que ter equipes multidisciplinares ou até mesmo distintas para a criação de produtos para as plataformas de diferentes sistemas operacionais.

A ideia aqui não é estabelecer uma comparação entre as duas modalidades, tão pouco apontar o melhor caminho, mas sim esclarecer ao leitor as opções disponíveis.

Esse livro concentra-se na abordagem do desenvolvimento nativo, utilizando a linguagem **Kotlin** e a ferramenta **Android Studio**.

O processo de criação de uma aplicação Android é bem diferente, por exemplo, do desenvolvimento de uma aplicação Web, por isso é importante reconhecer que conhecimentos prévios da linguagem escolhida são importantes, mas, não unicamente suficientes. A razão disso é que o framework para o desenvolvimento de aplicativos Android é complexo e vasto, e muitas das vezes, diferente de outras plataformas.

2.3.1 - A criação da interface gráfica

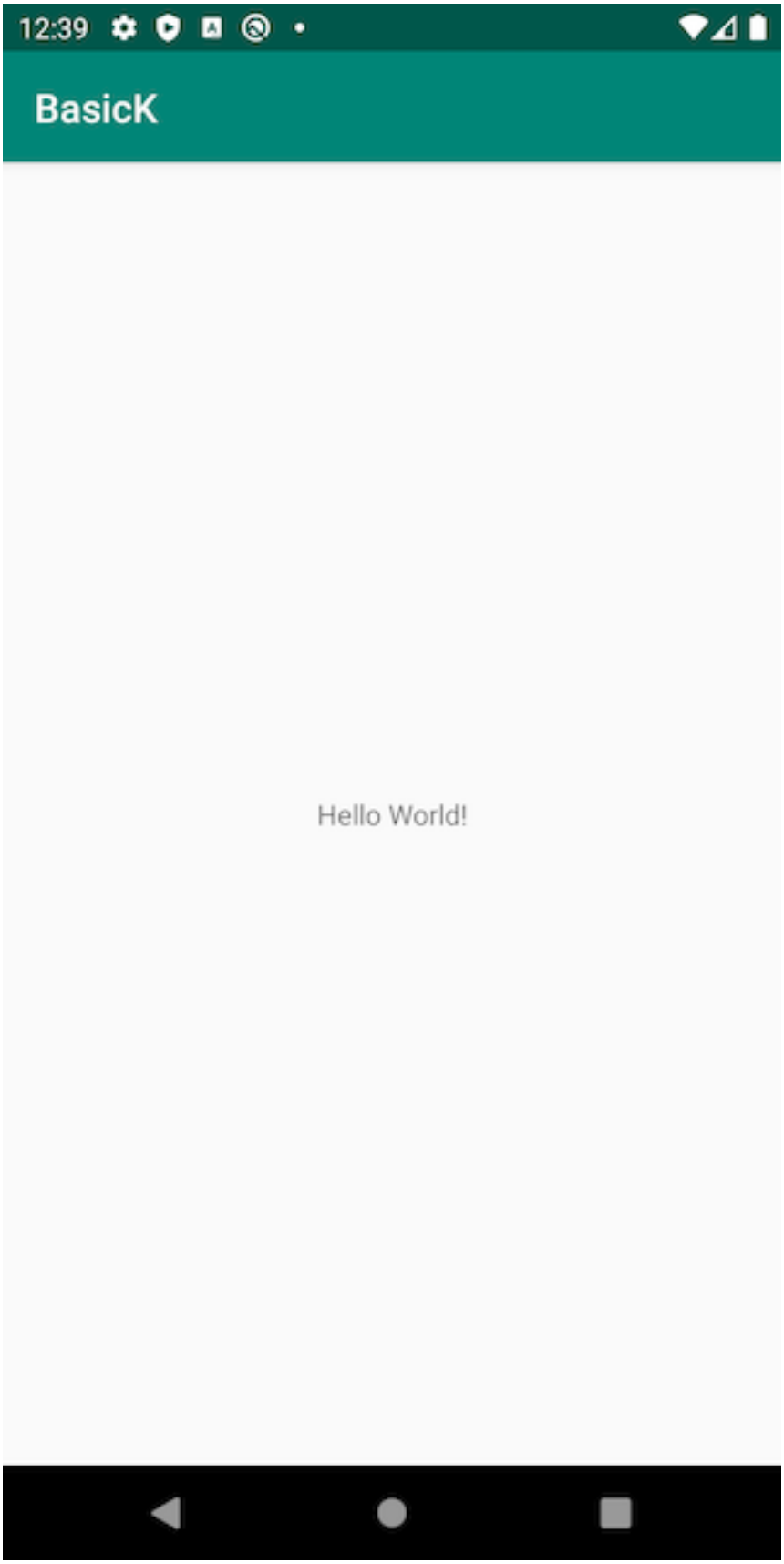
Em um projeto de um aplicativo Android, a interface gráfica do usuário é criada através de arquivos XML, que podem ser editados diretamente ou com a ajuda de ferramentas gráficas no Android Studio. Versões recentes dessa IDE têm tornado esse trabalho cada vez mais eficiente, graças ao lançamentos de novos componentes e funcionalidades.

Como exemplo, segue um arquivo XML de uma tela simples de um aplicativo Android:

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Como pode ser observado, essa tela contém somente dois componentes: um gerenciador de layout e uma caixa de texto, com algumas propriedades pré-definidas.

Esse arquivo é responsável por criar uma tela como a da figura a seguir:



Exemplo de tela de um aplicativo

Embora simples de se construir, essa tela já está preparada para se adaptar a vários tipos de dispositivos, com diferentes tamanhos e resoluções de tela, algo que, para telas mais complexas, pode ser considerado como uma arte!

Porém, essa tela não está adaptada a dispositivos com idiomas diferentes do Inglês, mas isso será detalhado mais adiante.

No fim das contas, essa tela será representada como um **recurso de layout** dentro do projeto Android.

2.3.2 - A criação do comportamento

Para que a tela anterior apareça dentro de uma aplicação Android, é necessário escrever um código em Kotlin capaz de exibi-la, como no trecho a seguir:

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(this, R.layout.activity_main)
    }
}
```

Essa classe, diante de algumas configurações feitas no projeto, pode ser considerada o ponto de entrada da aplicação. Ela é responsável por exibir um **recurso de layout**, que é o arquivo XML criado com a caixa de texto **Hello World!** da seção anterior.

Tecnicamente o ponto de entrada de uma aplicação Android não é esse, mas até esse momento do livro, é apropriado dizer isso do ponto de vista didático.

Aqui já é possível perceber que existem particulares únicas do Android para a construção de aplicativos.

A partir desses dois arquivos, é possível construir o projeto para a aplicação Android, obviamente contendo outros arquivos de configuração e com uma estrutura singular.

Pelas razões apresentadas, é importante conhecer bem o framework de desenvolvimento do sistema Android, independente da abordagem escolhida: nativo ou híbrido.

2.4 - Conclusão

Esse capítulo deu uma brevíssima introdução ao universo do sistema Android, que como já foi dito, é vasto e complexo.

O capítulo seguinte fará algo semelhante, introduzir o leitor ao **Google Firebase**, peça essencial dos objetivos desse livro.

3 - Sobre o Google Firebase

O Firebase é a plataforma do Google que oferece serviços para aplicações web e para dispositivos móveis, como **Android** e **iOS**, permitindo a rápida criação de sistemas baseados em *cloud computing* sem a necessidade de se gerenciar infraestrutura de servidores ou sistemas computacionais.

A integração de uma aplicação Android com o Firebase é muito simples e pode ser feita através de bibliotecas oferecidas pela própria Google, além de algumas passos e configurações que serão detalhados no momento oportuno nesse livro.

Como um dos objetivos principais desse livro é fazer com que o leitor possa construir aplicativos Android utilizando o Firebase, é interessante conhecer alguns de seus serviços oferecidos por essa plataforma, como descritos a seguir:

3.1 - Firebase Cloud Messaging

Talvez esse seja um dos serviços mais conhecidos do Firebase. Com ele é possível enviar notificações para uma aplicação em um dispositivo móvel, permitindo que o usuário possa ser notificado sobre algum evento.

Esse mecanismo pode ser invocado por uma aplicação de backend ou até mesmo através de eventos automáticos ou agendados.

O Firebase Cloud Messaging se encarrega totalmente de entregar a mensagem, cuidando da garantia de entrega, bem como o armazenamento da mesma, caso o dispositivo não esteja conectado à Internet no momento da entrega da mensagem.

3.2 - Firebase Authentication

Em uma aplicação onde é necessário autenticar o usuário, normalmente é necessário manter uma base desses usuários e também oferecer um mecanismo onde o dispositivo móvel possa verificar as suas credenciais. Com o Firebase Authentication é possível fazer isso de forma simples e direta, tanto para o desenvolvedor, como para o usuário da aplicação.

3.3 - Google Analytics for Firebase

Analisar eventos, logs e entender comportamentos de aplicativos para dispositivos móveis pode ser um desafio, uma vez que eles não estão concentrados, como em um serviço na nuvem. Para isso existe

o Google Analytics, um poderoso coletor, concentrador e analisador de eventos das mais diversas naturezas.

Com o Google Analytics, é possível gerar logs e eventos de dentro de uma aplicação Android e enviá-los para o Firebase, onde os desenvolvedores poderão analisá-los. Tal mecanismo também pode ser utilizado para enviar exceções durante a execução do código, permitindo a geração de evidências de defeitos na aplicação.

Uma das coisas interessantes desse serviço, do ponto de vista da aplicação no dispositivo móvel, é que mesmo que ele não esteja conectado à Internet no momento da geração do evento, ele será enviado para o Firebase, quando o dispositivo voltar a ficar conectado. Tudo sem a necessidade do desenvolvedor criar mecanismos complexos de armazenamento local de eventos e retentativas de entrega, ou seja, esse trabalho é totalmente feito pelo SDK do Firebase Analytics.

3.4 - Firebase Remote Config

Com o Firebase Remote Config é possível criar configurações para as aplicações, de tal forma que possam ser aplicadas sem a necessidade do usuário baixar uma nova versão. Tais configurações podem ser aplicadas a todos os usuários ou para segmentos que atendam algum critério de seleção, como país, gênero do usuário ou idioma.

Utilizando a interface do Firebase Remote Config, é possível alterar os valores das configurações existentes a cada aplicação. Essas são imediatamente enviadas a todos os dispositivos selecionados. Isso faz com que o comportamento da aplicação seja alterado, mediante o valor de cada configuração.

Esse recurso também pode ser utilizado para testes A/B ou mesmo para liberação de novas funcionalidades para grupos restritos de usuários.

3.5 - Firebase Cloud Firestore

Talvez esse seja o mais interessante e poderoso recurso do Firebase. Com ele é possível criar um banco de dados NoSQL para as aplicações, sem a necessidade de se manter um backend para prover o acesso pelos dispositivos móveis.

Com ele é possível restringir o acesso aos dados do banco através de políticas de segurança baseadas na autenticação do usuário.

Outra característica que torna esse serviço interessante, é o fato de ser considerado um banco de dados *realtime*, ou seja, caso um valor seja alterado, ele é imediatamente refletido em todos os dispositivos que estejam interessados nele.

3.6 - Conclusão

Esses, são apenas alguns dos serviços oferecidos pelo Firebase e que serão utilizados ao longo desse livro para criar aplicações para dispositivos Android conectadas à nuvem.

O próximo capítulo irá conduzir o leitor a preparar seu ambiente de desenvolvimento e tudo o que for necessário para começar a desenvolver sua primeira aplicação Android.

4 - Preparando o ambiente de desenvolvimento

Esse capítulo é dedicado à preparação do ambiente de desenvolvimento necessário para se trabalhar com Android, bem como a criação da conta do Firebase.

Esse processo pode demorar bastante, dependendo da velocidade de conexão com a Internet que a máquina de desenvolvimento possui.

Nesse link[4] é possível visualizar os requisitos mínimos necessários da máquina de desenvolvimento.

4.1 - Instalação do Android Studio

O Android Studio é a IDE oficial para o desenvolvimento de aplicações para Android. Ele pode ser baixado nesse link[5], de acordo com o sistema operacional, que pode ser Windows, Linux ou MacOS.

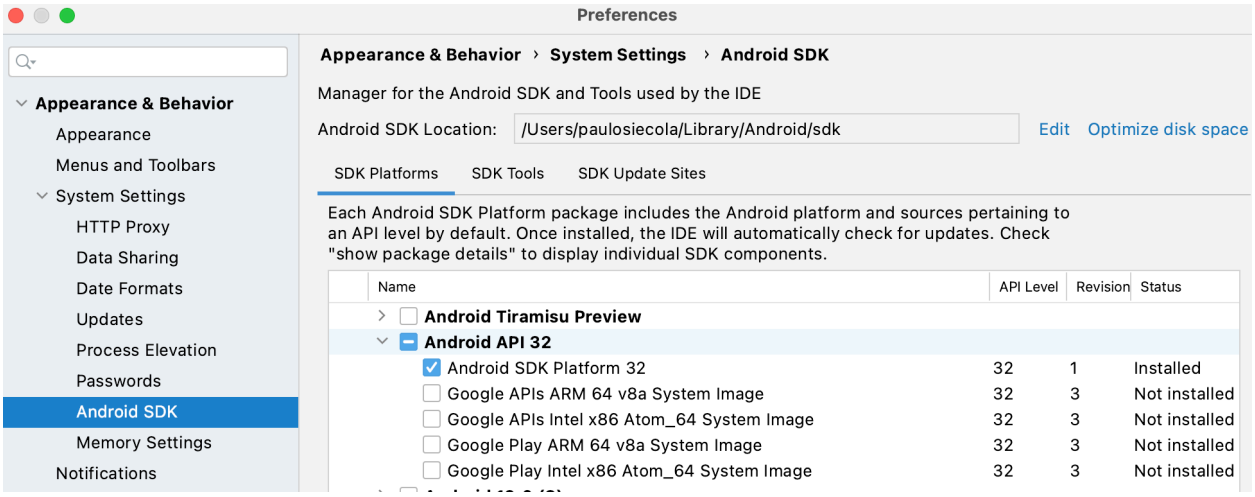
Após baixar o Android Studio, siga as instruções do processo de instalação.

Esse livro foi desenvolvido utilizando o Android Studio Bumblebee. Algumas telas podem ser diferentes em versões superiores.

4.2 - Seleção de pacotes adicionais

Depois de baixar e instalar o Android Studio, confira se os seguintes pacotes estão instalados em sua máquina, acessando o menu `Tools -> SDK Manager` e marcando a opção `Show Package Details` no canto inferior direito da janela que abrir:

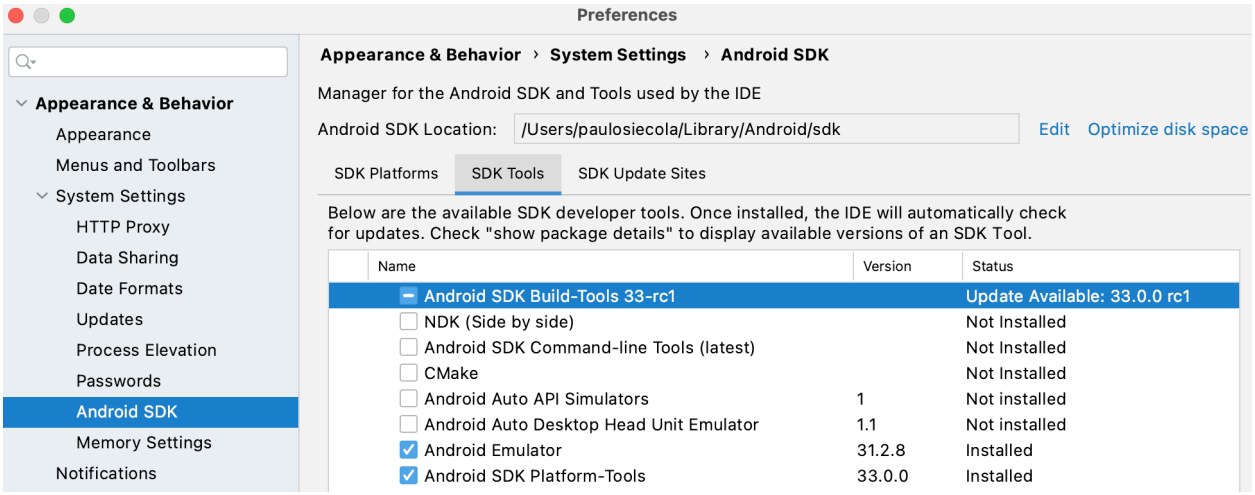
[4] https://developer.android.com/studio
[5] https://developer.android.com/studio



Pacotes necessários do SDK

A revisão pode ser diferente da imagem anterior, mas tenha certeza de que os pacotes marcados estejam instalados em sua máquina de desenvolvimento.

Ainda nessa janela, selecione a opção `SDK Tools` e instale a última versão disponível dos seguintes pacotes marcados, como mostra a figura a seguir:



Ferramentas

Novamente, a versão dos pacotes podem estar diferentes.

O Google libera atualizações constantes dos pacotes e ferramentas e o Android Studio frequentemente sugere atualizações dos itens que estão instalados.

Caso tenha problemas em instalar o pacote `Intel x86 Emulator Accelerator` no Windows, consulte o seguinte link[6].

[6] https://developer.android.com/studio/run/emulator-acceleration#vm-windows

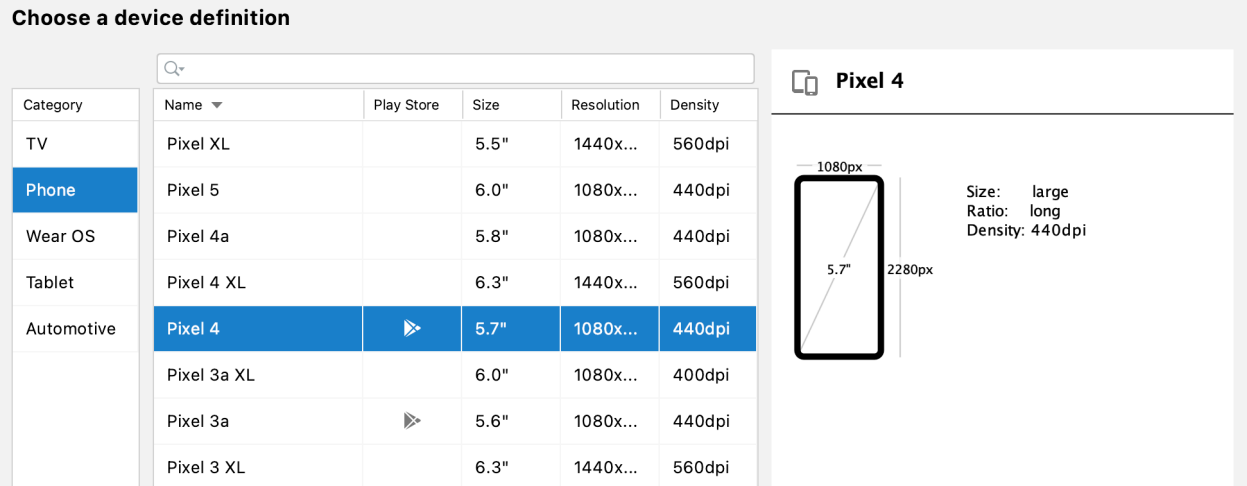
4.3 - Criação e configuração do AVD

É necessário criar um **Android Virtual Device** ou AVD para executar um dispositivo emulado do Android na máquina de desenvolvimento, para testar e executar os aplicativos que serão desenvolvidos ao longo desse livro.

Esse passo é opcional, pois também é possível utilizar um dispositivo real para testar os aplicativos.

Para criar um novo AVD, dentro do Android Studio, acesse o menu `Tools -> Device Manager`. Isso fará com que uma tela apareça, listando todos os dispositivos Android presentes em sua máquina de desenvolvimento.

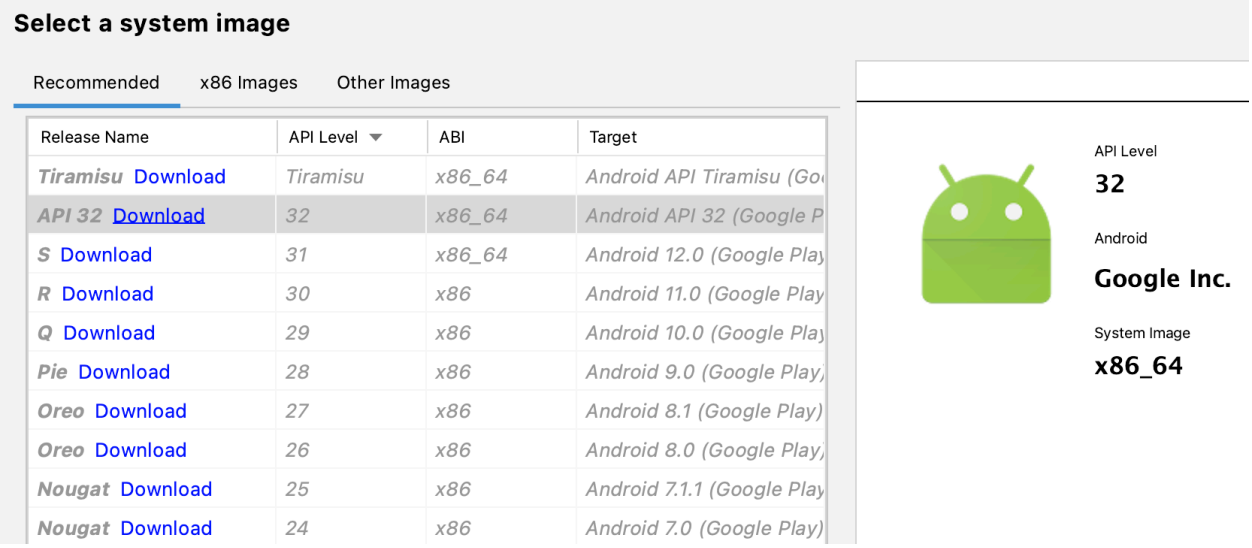
Nessa tela, clique no botão `Create Device` . Na tela que aparecer, selecione a opção `Phone` e escolha a opção **Pixel 4** ou um modelo de dispositivo mais novo, mas que tenha a opção `Play Store` marcada, como mostra a figura a seguir:



Escolha do dispositivo

Para prosseguir, clique no botão `Next`.

Na tela para a escolha das imagens, selecione uma que tenha a coluna `API Level` com o valor 32, ou superior. Caso a imagem a ser selecionada não esteja disponível em sua máquina, clique no link para fazer download:



Seleção da imagem

O nível da API se refere à versão do Android. Isso será detalhado no capítulo 6, mas a API 32 se refere ao Android 13.

Em seguida, clique em `Next` e configure um nome desejado para novo AVD, na última tela de configuração.

É aconselhável deixar as opções avançadas dessa tela nas configurações padrão. Altere algo apenas se tiver certeza do que está fazendo.

Para finalizar, clique no botão `Finish` para a conclusão da criação do novo AVD.

Ao final do processo de criação do AVD, o Android Studio voltará à tela inicial do Device Manager e o novo dispositivo estará disponível para ser iniciado ou alterado, através dos botões localizados no canto direito da lista dessa tela.

Execute o novo AVD para ter certeza de que ele está funcionando na sua máquina de desenvolvimento. Caso tenha problemas para executar o AVD, consulte esse link[7] para obter informações e respostas para seu possível problema.

É possível executar mais de um AVD ao mesmo tempo, dependendo do poder de processamento da máquina de desenvolvimento.

4.5 - Configuração de um dispositivo real para ser utilizado durante o desenvolvimento

Também é possível habilitar um dispositivo real Android para que seja utilizado durante o processo de desenvolvimento de aplicações, ou seja, para que ele sirva como dispositivos de testes. Para isso, é

[7] https://developer.android.com/studio/run/emulator-troubleshooting

necessário habilitar o **modo desenvolvedor** do aparelho. Esse link[8] fornece informações úteis para isso, porém, é importante frisar que as opções variam de acordo com o fabricante, por isso consulte a documentação oficial do seu aparelho antes.

Depois que o dispositivo real for configurado, ele ficará disponível para o Android Studio, quando ele for executar alguma aplicação, como será visto no capítulo 6.

4.5 - Criação da conta do Firebase

Para acompanhar todos os exercícios propostos nesse livro, será necessário criar uma conta no Firebase. Para isso, acesse esse link[9] para poder prosseguir com o processo. Você também pode utilizar sua conta do Google, caso já possua alguma.

4.6 - Conclusão

Esse capítulo mostrou alguns caminhos iniciais que devem ser seguidos para preparar a máquina de desenvolvimento para trabalhar com Android.

O capítulo seguinte oferece uma introdução sutil à linguagem Kotlin, para servir de base teórica ao capítulo 6, onde o primeiro projeto Android será criado.

[8] https://developer.android.com/studio/debug/dev-options
[9] http://console.firebase.google.com/

5 - Um pouco sobre Kotlin

O objetivo desse capítulo não é transformar o leitor em um especialista em Kotlin, mas sim apresentar alguns conceitos importantes que serão necessários ao longo desse livro.

A linguagem Kotlin é semelhante, em alguns aspectos, à linguagem Java. Além disso, Kotlin é interoperável com Java, ou seja, em uma mesma aplicação é possível ter as duas linguagens funcionando juntas.

A documentação oficial de Kotlin é muito boa e está disponível nesse link[10]. Além disso, também existe um curso online gratuito que pode ser acessado aqui[11].

Outro recurso para aprender Kotlin e praticar é esse playground[12] oferecido pelo seu site. Se desejar, utilize esse site para testar trechos de código desse capítulo.

Os tópicos a seguir cobrem alguns pontos básicos de Kotlin, apenas demonstrando conceitos, principalmente os que se diferem de alguma forma de outras linguagens como o Java.

5.1 - O básico de Kotlin

Essa seção é dedicada aos pontos mais básicos de Kotlin, como declaração de variáveis, funções e expressões interessantes com a cláusula **when**.

5.1.1 - Declaração de variáveis

Em Kotlin, as variáveis possuem tipos definidos, mas podem ser declaradas essencialmente de duas formas:

- **implícita**: quando o tipo é inferido pelo valor atribuído durante sua declaração

```
var x = 7
var name = "Matilde"
```

- **explícita**: quando o tipo é definido explicitamente durante sua declaração

[10] https://kotlinlang.org/docs/reference/
[11] https://www.udacity.com/course/kotlin-bootcamp-for-programmers--ud9011
[12] https://play.kotlinlang.org

```
var x: Int = 8
var name: String = "Doralice"
```

Em Kotlin, as variáveis devem possuir um valor quando são declaradas ou terem um valor atribuído antes de serem utilizadas. Obviamente, existem contornos na linguagem para isso.

Nos dois casos apresentados anteriormente, as variáveis podem ter seus valores alterados, mas também é possível declará-las de tal forma que seus valores não possam ser alterados, com a instrução `val`:

```
val y = 7
val name = "Clotilde"
```

Dessa forma, essas duas variáveis não poderão sofrer alterações dentro do escopo que pertencem.

5.1.2 - String templates

Em Kotlin é possível montar strings contendo valores de outras variáveis de forma muito mais intuitiva, como no exemplo a seguir:

```
var z = 3
print("The value is ${z}")
```

Nesse caso, a expressão `${z}` será substituída no resultado final com o valor da variável `z`:

```
The value is 3
```

5.1.3 - Condicionais

As expressões condicionais em Kotlin possuem semelhanças com outras linguagens, como em Java:

```
val x = 3
val y = 5
if (x > y) {
    print("x is greater than y")
} else {
    print("y is greater than x")
}
```

Porém, elas podem possuir condições para intervalos, como no trecho a seguir:

```
var x = 4
if (x in 1..10) {
    print("It's in the range")
}
```

Os operadores de intervalos também podem ser utilizados em loops e em cláusulas `when`.

5.1.4 - Loops

A seguir um exemplo de uma iteração, utilizando a estrutura `for` para varrer uma lista de strings:

```
var dogs = listOf("Matilde", "Doralice", "Hannah", "Clotilde")
for (dog in dogs) {
        println("New dog is: ${dog}")
}
```

E aqui um outro exemplo simples utilizando a estrutura `while`:

```
var x = 0
while (x < 10) {
        println("Value is: ${x}")
        x++
}
```

5.1.5 - Funções

Funções em Kotlin possuem uma sintaxe bem diferente de Java, além de algumas vantagens interessantes, como:

- valores padrões;
- argumentos com nomes.

A seguir um exemplo de uma função simples para somar dois inteiros e retornar seu resultado:

```
fun sum(a: Int, b: Int): Int {
    val result = a + b
    return result
}
```

Ela poderia ser chamada da seguinte forma:

```
fun main() {
    val result = sum(3, 4)
    print("Result: ${result}")
}
```

Na segunda linha, a função `sum` é chamada com seus parâmetros, mas ela também poderia ser invocada da seguinte forma:

```
fun main() {
    val result = sum(a = 3, b = 4)
    print("Result: ${result}")
}
```

Repare que agora os parâmetros são nomeados, fazendo com que o código fique mais legível, principalmente quando se tem argumentos de tipos iguais, como é o caso.

Uma outra forma de definir os parâmetros de uma função é criando parâmetros com valores padrões, como no trecho a seguir:

```
fun sum(a: Int, b: Int = 1): Int {
    val result = a + b
    return result
}
```

Dessa forma, se o segundo parâmetro for omitido em sua invocação, ela assumirá o valor `1`, nesse caso:

```
fun main() {
    val result = sum(a = 3)
    print("Result: ${result}")
}
```

5.1.6 - Expressão When

A expressão **when** em Kotlin substitui o `switch`, presente em outras linguagens, porém, com muitas vantagens, pois ela pode ter condições variadas de testes. Além disso, seu resultado pode ser utilizado em retorno de funções e atribuições de variáveis.

```
fun testNumber(number: Int) {
        when (number) {
        in 1..10 -> print("Between 1 and 10")
        11 -> print("Exactly 11")
        else -> print("Unknown number")
    }
}
```

As várias formas de fazer condicionais em Kotlin podem ser utilizadas dentro da expressão `when`.

Como dito, o resultado da expressão `when` pode ser utilizado para retorno e funções ou atribuições de variáveis:

```
fun main() {
    print(testNumber(15))
}

fun testNumber(number: Int): String {
        return when (number) {
        in 1..10 -> "Between 1 and 10"
        11 -> "Exactly 11"
        else -> "Unknown number"
    }
}
```

Na verdade também é possível chamar outras funções de dentro do `when`, como no trecho a seguir:

```
fun main() {
    testNumber(7)
}

fun testNumber(number: Int) {
        when (number) {
        in 1..10 -> numberCondition1()
        11 -> numberCondition2()
        else -> numberCondition3()
    }
}

fun numberCondition1() {
    print("Between 1 and 10")
}
```

```
fun numberCondition2() {
    print("Exactly 11")
}

fun numberCondition3() {
    print("Unknown number")
}
```

5.1.7 - Valores nulos e checks

Em Kotlin há uma abordagem diferente para condições quando uma variável pode ter um valor nulo ou não. Por exemplo, o seguinte trecho de código não compila:

```
fun main() {
    var name: String = "Matilde"
    name = null //compilation error

    print("Name: ${name}")
}
```

Isso quer dizer que variáveis declaradas dessa forma não podem receber valores nulos. Porém, se houver a necessidade de uma variável receber um valor nulo, durante a execução do programa, ela deve ser declarada com essa questão de forma explícita, da seguinte forma:

```
fun main() {
    var name: String? = "Matilde"
    name = null

    print("Name: ${name}")
}
```

Repare que na declaração da variável há um sinal de `?` indicando que ela pode receber nulo durante a execução do programa.

E para acessar métodos dessa variável, é necessário fazer um *null check* antes:

```
fun main() {
    var name: String? = "Matilde"

    if (name != null) {
        if (name.length > 10) {
                print("Long name...")
        } else {
            print("Short name...")
        }
    } else {
        print("Name is null")
    }
}
```

Uma vez que sem isso, há um erro de compilação:

```
fun main() {
    var name: String? = "Matilde"

    if (name.length > 10) {
        print("Long name...")
    } else {
        print("Short name...")
    }
}
```

Ainda existem outras vantagens dessa abordagem, principalmente quando é necessário atribuir valores em propriedades de objetos, que podem ser nulos.

5.2 - Classes e objetos

Classes em Kotlin são declaradas através da palavra-chave `class`:

```
class Dog {

}
```

E um objeto dessa classe pode ser criado e atribuído em uma variável, como no trecho à seguir:

```
fun main() {
        val dog = Dog()
}
```

A declaração da classe também pode ser feita já com suas propriedades, como no trecho a seguir:

```
class Dog (var name: String, var age: Int, var color: String = "black") {

}
```

Nesse caso o **construtor primário** está presente na declaração da classe, como pode ser visto logo após seu nome, com as propriedade sendo definidas.

Perceba que também é possível utilizar valores padrões para as propriedades da classe durante a criação da sua instância:

```
fun main() {
        val dog = Dog("Matilde", 15)
}
```

Nesse caso as propriedades `name` e `age` receberão os valores `Matilde` e `15`, que foram passados como parâmetros na criação. Já a propriedade `color` receberá o valor `black`, uma vez que ele foi omitido nessa declaração, mas que poderia ter um outro valor se ele fosse passado:

```
fun main() {
        val dog = Dog("Clotilde", 3, "multicolored")
}
```

A seguir serão mostrados alguns outros conceitos úteis e interessantes sobre classes em Kotlin.

5.2.1 - Bloco inicializador

É possível criar um código para ser executado no momento da criação de um objeto de uma classe, através do bloco `init`:

```
class Dog (var name: String, var age: Int, var color: String = "black") {
    init {
        println("Dog name: ${this.name}")
        println("Dog color: ${this.color}")
    }
}
```

Isso é útil uma vez que o construtor primário não pode ter nenhum código para se executado.

5.2.2 - Funções de classes

Obviamente uma classe em Kotlin também pode ter funções, como o exemplo a seguir:

```
class Dog (var name: String, var age: Int, var color: String = "black") {
    init {
        println("Dog name: ${name}")
        println("Dog color: ${this.color}")
    }

    fun dogType(): String {
        return when (color) {
            "black" -> "The real black dog"
            "blond" -> "A special blond dog"
            else -> "Maybe it's colored"
        }
    }
}
```

Perceba que a declaração de uma função de uma classe segue os mesmos padrões de funções comuns.

Para invocar essa função, basta fazer como no trecho a seguir:

```
fun main() {
        val dog = Dog("Clotilde", 15, "multicolored")
  println ("Dog type: ${dog.dogType()}")
}
```

Como a função retorna uma `string`, ela já pode ser utilizada diretamente na impressão de uma mensagem de texto.

5.2.3 - Herança

Para declarar uma nova classe que herda de `Dog`, primeiramente é necessário deixar claro que ela pode ser herdada, através do modificador `open`, como mostra o trecho a seguir:

```
open class Dog (var name: String, var age: Int, var color: String = "black") {

}
```

Dessa forma, é possível criar uma outra classe (BlackDog) que herda dela:

```
class BlackDog (name: String, age: Int): Dog(name, age, "black") {

}
```

Perceba que em sua declaração, os parâmetros são passados para a classe pai (Dog).

5.2.4 - Data classes

Para classes onde a única razão é armazenar dados, é possível utilizar *data classes* em Kotlin, como no trecho a seguir:

```
data class Cat(val name: String, val color: String)
```

Sua criação e utilização são semelhantes a classes comuns, como pode ser visto no trecho a seguir:

```
fun main() {
        val cat = Cat("Eek", "purple")
  print("Cat name: ${cat.name} - color: ${cat.color}")
}
```

5.3 - Conceitos avançados

Ainda existem outros conceitos de Kotlin que serão utilizados ao longo desse livro, principalmente porque muitas bibliotecas do Android fazem uso desses conceitos. Porém, eles serão introduzidos à medida em que forem necessários. Dessa forma o leitor consegue ter uma apresentação mais prática e eficiente. A seguir, alguns desses conceitos:

- inline function;
- lambda expression;
- lateinit.

Quando alguns desses recursos de Kotlin forem necessários pela primeira vez, haverá uma breve explicação no livro para que o leitor possa ter um entendimento mais amplo.

5.4 - Conclusão

Esse capítulo introduziu alguns conceitos básicos de Kotlin, necessários para que, a partir do próximo capítulo, o leitor possa começar a construir o primeiro aplicativo para Android utilizando essa linguagem.

6 - Criação do primeiro projeto

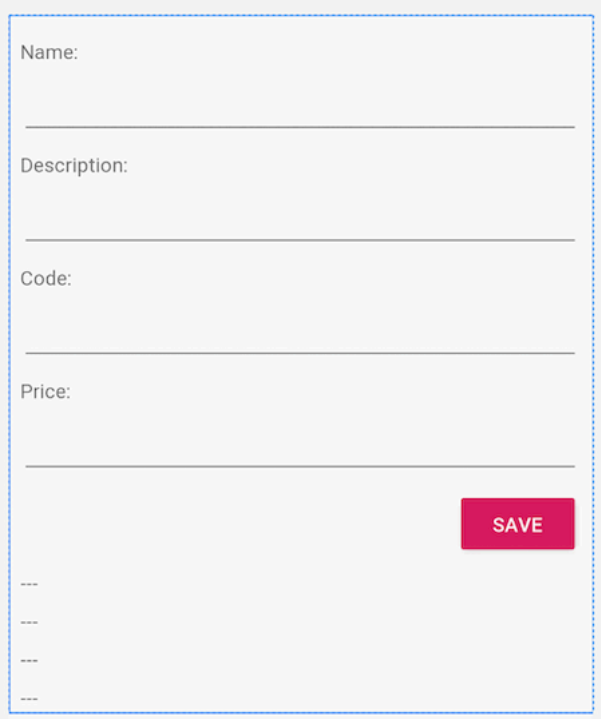
Esse capítulo é dedicado a descrever os passos para a criação do primeiro projeto em Android utilizando Kotlin, além de começar a explorar alguns conceitos básicos, como:

- Estrutura do projeto Android;
- Criação e conceitos iniciais da interface gráfica;
- Definição do comportamento da interface gráfica através do código em Kotlin;
- View binding.

É importante lembrar que o código do projeto completo está disponível nos extras que pode ser baixado junto com o livro.

A ideia desse capítulo é fazer com que o leitor tenha uma introdução desses conceitos para prosseguir com assuntos mais avançados. Por isso, guarde esse projeto, pois ele ainda será bastante utilizado em capítulos adiante.

O aplicativo a ser criado aqui será uma simples tela onde o usuário poderá digitar informações de um produto e, quando desejar, poderá “salvar” (que na verdade não vai salvar em nenhum lugar), exibindo suas informações em caixas de texto na mesma tela:

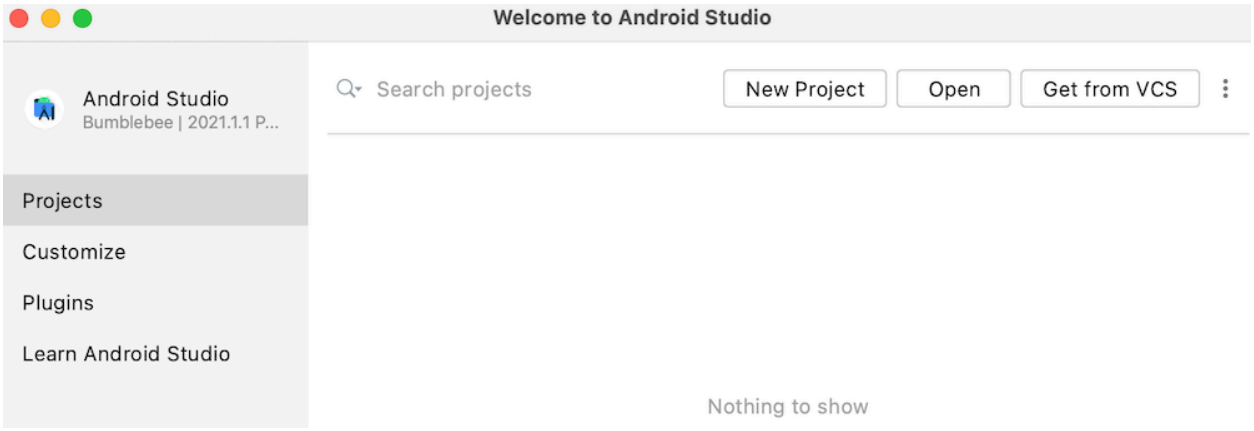


Tela do aplicativo

A ideia é fazer um aplicativo de lista de compras, que só pode ter um produto na lista. Obviamente, isso não tem nenhuma utilidade prática, mas tem muitos detalhes que serão utilizados para ensinar os primeiros conceitos.

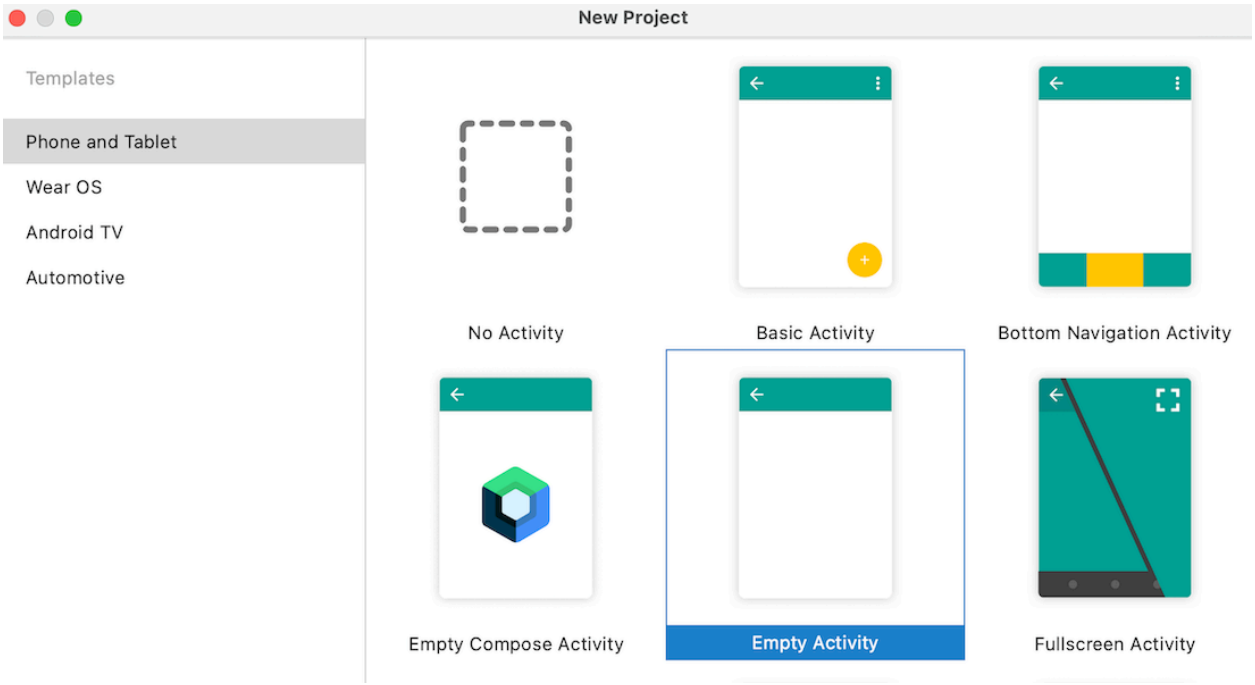
6.1 - Criando o primeiro projeto

Para criar o primeiro projeto, abra o Android Studio e selecione a opção `+ Start a new Android Studio project`:



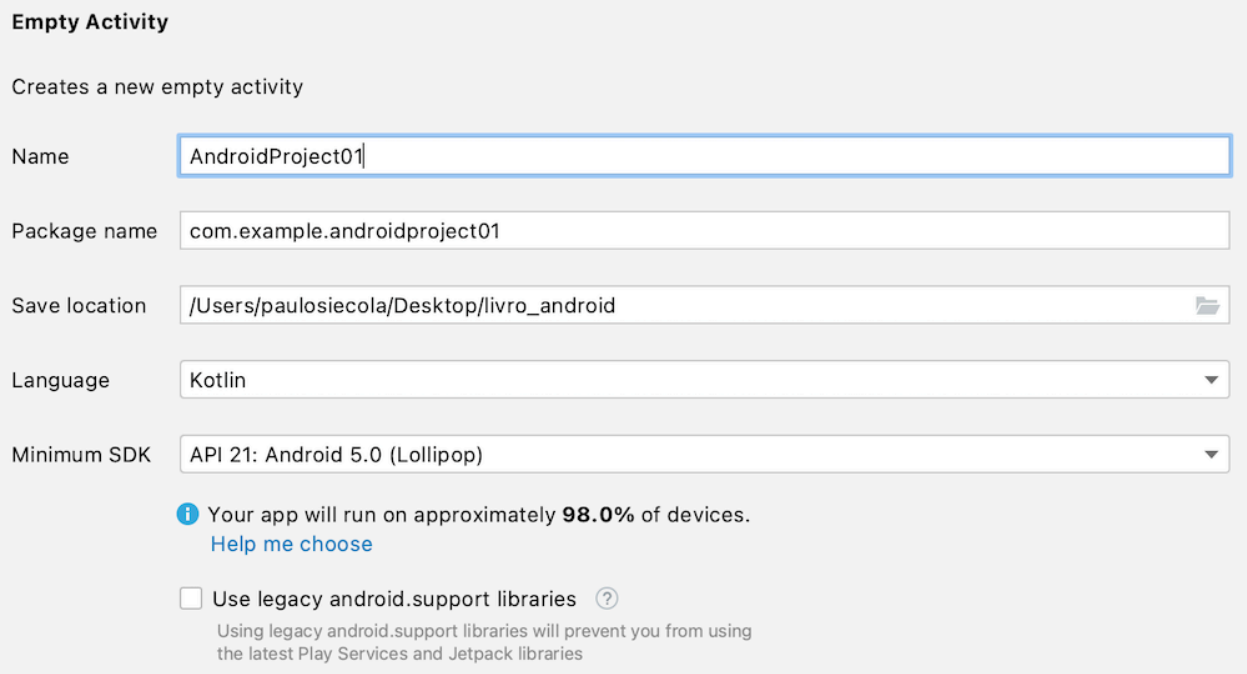
Iniciando um novo projeto no Android Studio

Como a ideia desse projeto é apenas dar os passos iniciais, sua estrutura será a mais simples possível, por isso selecione a opção `Empty Activity` na tela seguinte, dentro da seção `Phone and Tablet`:



Selecionando a estrutura do projeto

Repare que nessa tela existem outras opções para criação de projetos mais complexos, o que facilita o trabalho inicial na construção de aplicativos com opções de navegação mais sofisticados. Porém, esse não é o caso agora, então a opção `Empty Activity` é suficiente. Em seguida, clique em `Next` para passar para a próxima tela:



Configuração do projeto

Nessa última tela existem configurações importantes a serem feitas:

- **Nome do projeto**: Esse é o nome que vai aparecer para o usuário quando ele instalar o aplicativo em seu dispositivo;
- **Nome do pacote**: O nome do pacote deve ser único para ser publicado na loja do Google, por isso é interessante colocar seu site em notação reversa, seguido do nome do projeto. Porém, para facilitar a condução dos exercícios nesse livro, mantenha o mesmo nome que foi configurado na figura anterior;
- **Local a ser salvo**: Escolha um local para salvar o projeto na sua máquina de desenvolvimento;
- **Linguagem**: Escolha a linguagem **Kotlin**;
- **Nível mínimo da API**: Essa é uma configuração importante, que define a versão mínima do Android em que o aplicativo irá rodar. Para esse exemplo, escolha a mesma opção da figura anterior.

Esse livro foi desenvolvido utilizando o Android Studio Bumblebee. Algumas telas podem ser diferentes em versões superiores.

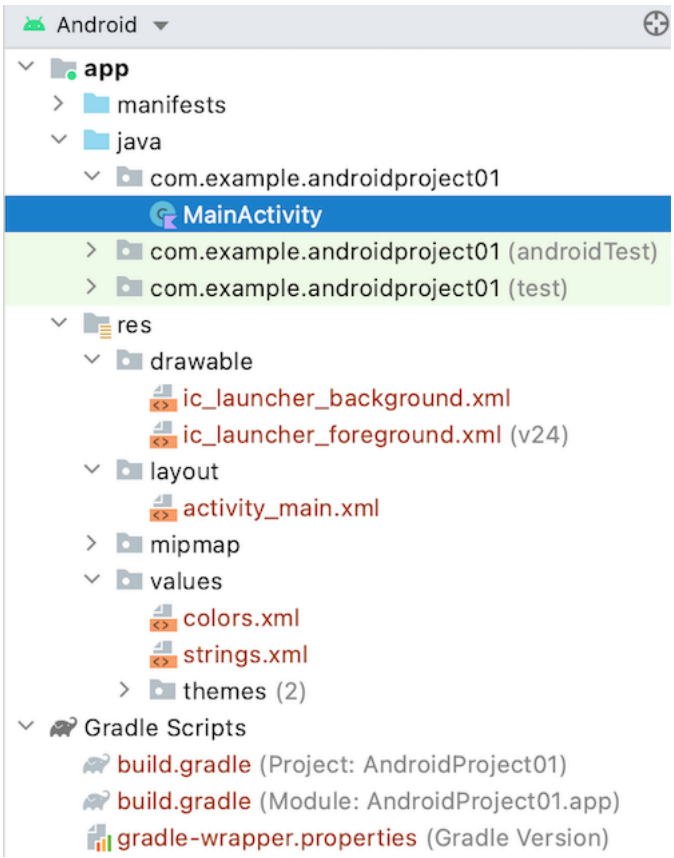
O nível mínimo da API pode ser alterado posteriormente dentro do projeto, porém é importante saber escolher esse nível, pois, ela definirá a versão mínima do sistema operacional que o aplicativo poderá ser executado. O Android Studio oferece uma estatística da porcentagem de dispositivos atuais capazes de rodar o aplicativo em cada nível de API. Para saber mais sobre qual API corresponde a cada versão do Android, consulte esse link[13].

[13] https://source.android.com/setup/start/build-numbers

Finalize a configuração e peça ao Android Studio para criar o projeto. Após algum tempo, ele estará aberto e pronto para ser trabalhado!

6.2 - Estrutura do projeto

Depois que o projeto é criado, é possível observar sua estrutura no Android Studio, na aba lateral esquerda, chamada `Project`:



Estrutura do projeto

Essa é a estrutura básica de um projeto Android. A seguir, algumas informações sobre essa estrutura.

a) Arquivo AndroidManifest.xml: esse arquivo define permissões que o aplicativo necessita para ser executado além de instruções ao sistema Android. Nesse primeiro projeto não será necessário alterá-lo, mas no próximo projeto sim, pois, será necessário solicitar permissão de acesso à Internet;

b) Classe MainActivity: essa é a classe em Kotlin onde será criado o comportamento do aplicativo;

c) Pasta de recursos: imagens, strings, layouts e valores são considerados recursos em projeto Android, e ficam nessa pasta;

d) Scripts do Gradle: aqui estão definidos os arquivos para que o projeto seja compilado utilizando Gradle. A seção seguinte traz mais detalhes sobre esse tópico.

A pasta `res` está dividida pelo tipo do recurso e a o arquivo `activity_main.xml` é o que define a interface gráfica do usuário. Ele é um arquivo no formato `XML` onde os componentes são criados, configurados e ajustados no layout da tela. A seção 6.4 adiante traz mais detalhes sobre esse arquivo.

6.3 - Arquivo build.gradle

O projeto Android utiliza o Gradle como gerenciador de dependências, que são as bibliotecas que podem ser adicionadas para desempenhar diversas funções.

Na pasta `Gradle scripts` existem dois arquivos `build.gradle`: um para todo o projeto e outro para o módulo `app`. Isso faz com que o projeto esteja preparado para múltiplos módulos, que não é alvo desse livro. Para saber mais sobre módulos, consulte esse link[14].

O arquivo `build.gradle (Module: app)` é o que define as configurações do módulo `app`, que nesse caso concentra todo o aplicativo a ser desenvolvido. Nele é possível distinguir 3 seções até o momento:

a) Plugins:

Nessa seção são definidos os plugins necessários para a compilação do projeto. A seguir estão os que foram criados pelo Android Studio:

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}
```

b) Configurações de compilação:

Essa seção foi parcialmente definida durante a criação do projeto, quando a versão mínima da API foi definida:

```
android {
    compileSdk 32
    defaultConfig {
        applicationId "com.example.androidproject01"
        minSdk 21
        targetSdk 32
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
```

[14] https://developer.android.com/studio/projects

```
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), '\
proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}
```

Aqui vale a pena ressaltas as seguintes configurações:

- **applicationId**: essa é a identificação única do aplicativo. Esse valor será utilizado caso ele seja publicado na loja do Google;
- **compileSdk**: esse valor especifica o nível da API em que o projeto será compilado;
- **minSdk**: define o nível mínimo da API que o aplicativo poderá ser executado, que consequentemente define a versão mínima do Android;
- **targetSdk**: esse é o valor que indica a versão da API que o aplicativo será testado.

Algumas configurações ainda serão feitas nessa seção para adicionar funcionalidades ao processo de compilação e desenvolvimento do projeto.

c) Dependências:

Nessa seção são colocadas as dependências de bibliotecas que o projeto necessita:

```
dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

As versão devem ser muito bem escolhidas para não haver incompatibilidade entre elas. Uma dependência que vale a pena ser ressaltada nessa configuração inicial é a biblioteca `ConstraintLayout`, que facilita muito o desenvolvimento de interfaces gráficas.

De preferência, utilize as versões listadas no trecho anterior.

Quando novas bibliotecas forem necessárias ao projeto, sejam do próprio Google ou de terceiros, elas deverão ser adicionadas aqui para serem utilizadas pelo projeto.

Para informações mais detalhadas sobre o arquivo `build.gradle`, consulte a documentação oficial nesse link[15].

6.4 - Criando a primeira interface gráfica

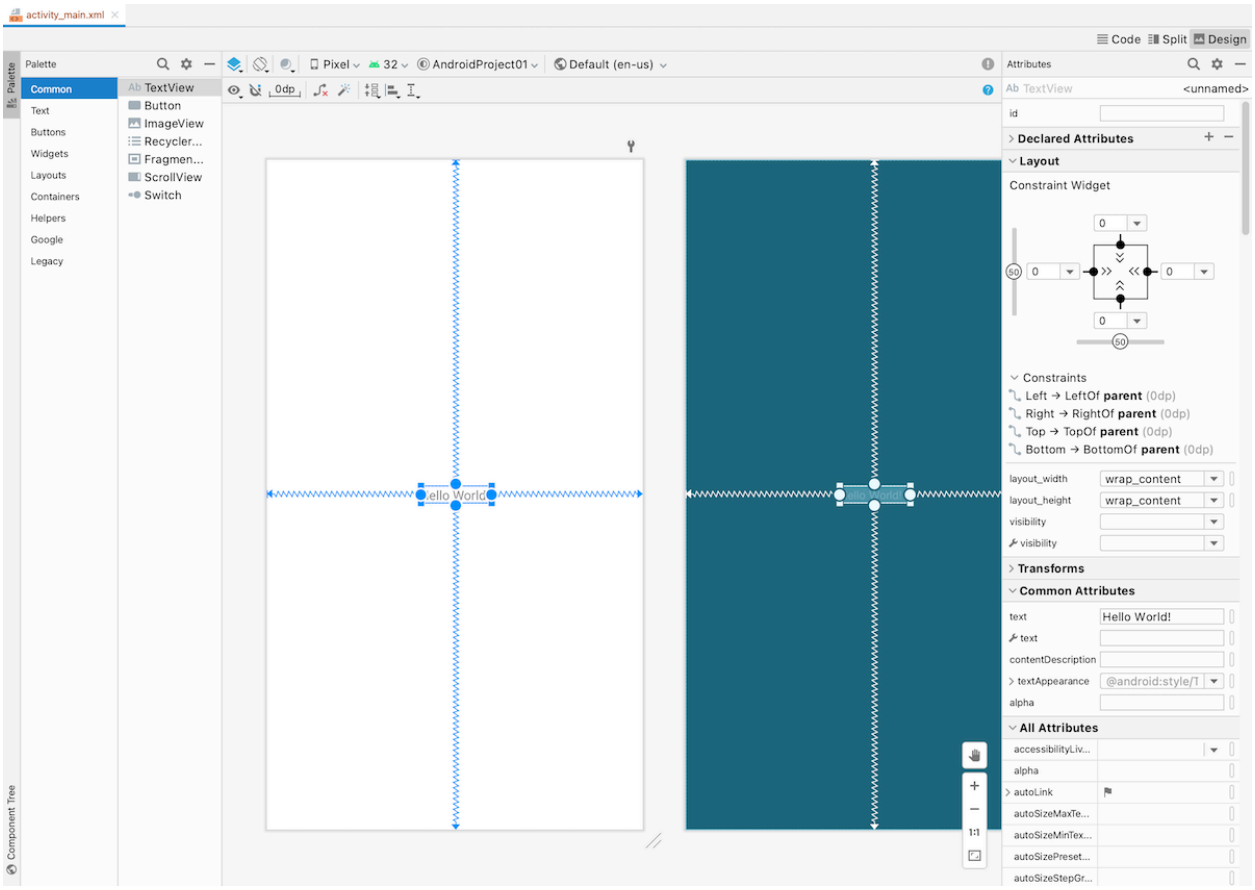
Como dito anteriormente, as interfaces gráficas em projeto Android são definidas em arquivos XML. O projeto criado possui apenas uma tela, que está no arquivo `res\layout\activity_main.xml`.

Ao abrir esse arquivo é possível ter três visões, controladas por abas que estão localizadas na parte superior direita da tela:

- **Code**: nessa visão é possível editar o arquivo XML e ter uma pré-visualização ao mesmo tempo;
- **Split**: aqui é possível visualizar o código e o design da tela ao mesmo tempo;
- **Design**: nessa aba é possível visualizar e editar a interface gráfica utilizando os recursos do Android Studio.

Ao selecionar a aba `Design`, é possível ter uma visualização como a da figura a seguir:

[15] https://developer.android.com/studio/build



Design da tela

À esquerda estão a paleta de componentes gráficos, que podem ser arrastados para dentro da tela e sua árvore de componentes. No centro, está a visualização da tela, bem como um *blue print*, onde configurações de layout e espaçamento podem ser observados. Por fim, do lado direito são exibidos os atributos do componente selecionado, bem como as suas configurações de posicionamento na tela.

No Android, é possível criar telas constituídas de vários tipos de layouts, que permitem a organização dos componentes em forma de listas horizontais, listas verticais, grades com colunas e linhas e também com posicionamento relativo entre os componentes.

A tela criada pelo Android Studio possui um layout chamado `ConstraintLayout`, como pode ser observado dentro do arquivo `XML`, na aba `Text` da figura anterior:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.and\
roid.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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Não se preocupe se existem muitas coisas novas nesse arquivo. Os conceitos ainda serão apresentados.

Esse tipo de layout permite que os componentes fiquem organizados na tela de forma relativa a outros componentes.

Nesse caso, há apenas um componente do tipo `TextView`, que está posicionado no centro de seu pai, que é a tela inteira. Isso permite que ele sempre fique nessa posição, independente do tamanho da tela ou de sua orientação durante a execução da aplicação. Os atributos que definem esse comportamento são os que estão definidos com os nomes iniciando com `app:layout_constraint`.

No Android, todos os componentes gráficos são considerados algum tipo de `View`. Nesse caso, `TextView`é uma `View` especializada em exibição de textos.

Ainda no componente `TextView`, existem 3 outras propriedades importantes:

- **layout_width**: isso define a largura máxima que o componente irá utilizar da tela. A opção `wrap_content` diz que ele não deve ocupar mais do que o necessário para exibir o conteúdo;
- **layout_height**: essa propriedade define a altura máxima que o componente deve ocupar na tela. A opção `wrap_content` diz que tal altura deve ser o suficiente para exibir o conteúdo;
- **text**: esse é o valor que será exibido na tela para o usuário.

Ainda existem outras propriedades que um `TextView` pode assumir, mas que não fazem parte dessa primeira `View`.

Todas as propriedades dos componentes, bem como sua organização e criação, podem ser feitas tanto diretamente no arquivo XML, quanto na interface gráfica, na aba `Design`. O desenvolvedor tem a liberdade de escolher a forma que preferir.

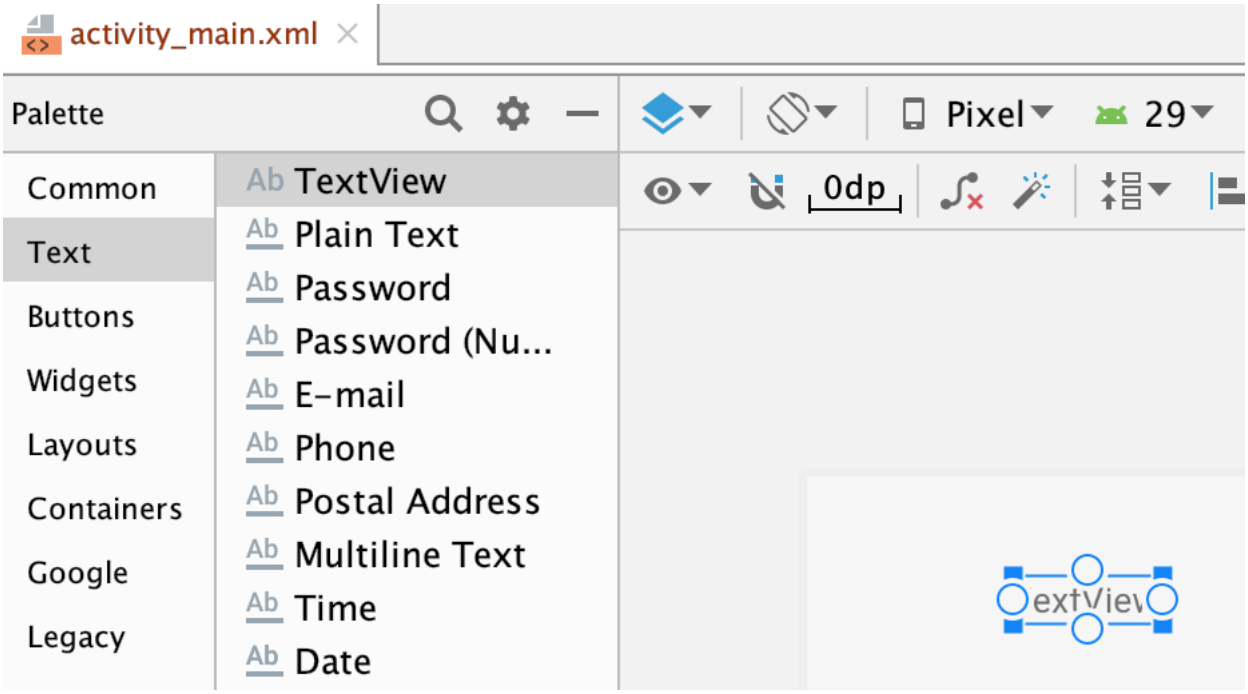
Como o objetivo aqui é desenvolver uma tela, como a do início do capítulo, serão necessários os seguintes componentes:

- 8 TextView;
- 4 EditText;
- 1 Button.

Esses componentes devem ser agrupados de tal forma a criar uma interface gráfica capaz de se adaptar a qualquer tamanho de tela e orientação. E é aí que o `ConstraintLayout` pode ser utilizado para facilitar esse trabalho.

Para começar, mude para a aba `Design` e apague o `TextView` que está na tela.

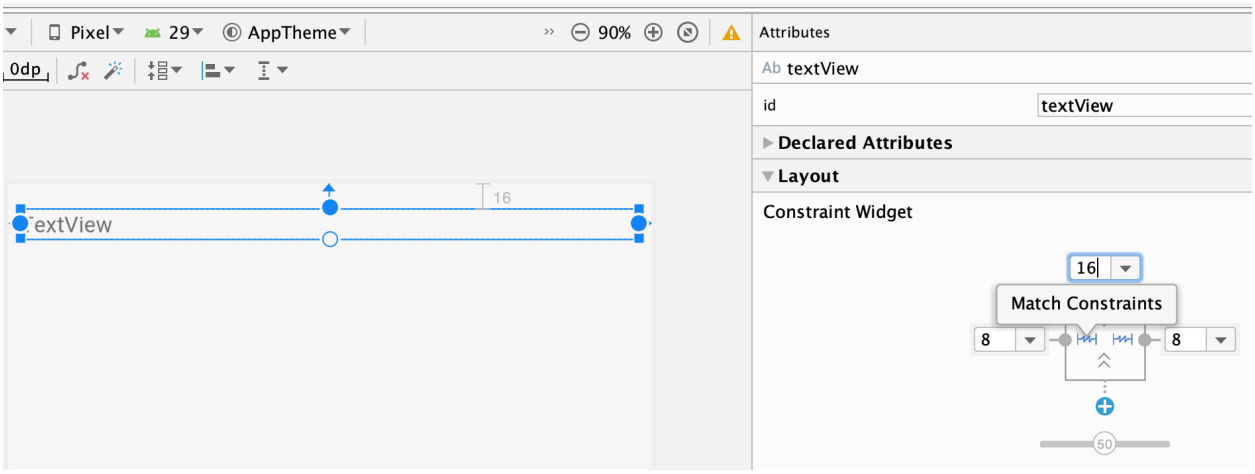
Utilizando a paleta de componentes do lado esquerdo, escolha o componente `TextView` e coloque próximo ao topo da tela. Repare que quando o componente é selecionado, aparecem 4 círculos, como na figura a seguir:



Selecionando o TextView

Esses círculos são utilizados para posicionar o componente em relação a outros componentes ou às bordas da tela, quando se constroi uma interface com base no `ConstraintLayout`.

Conecte os círculos superior, direito e esquerdo às bordas da tela:



Constraints do TextView

Porém, quando as ligações são feitas com as bordas da tela, o Android assume valores padrões para os atributos dessas ligações, que não são as desejadas para esse componente. Para isso, mantendo o `TextView` selecionado, vá até a aba `Attributes`, no canto direito e configure a seção `Layout` com as seguintes características:

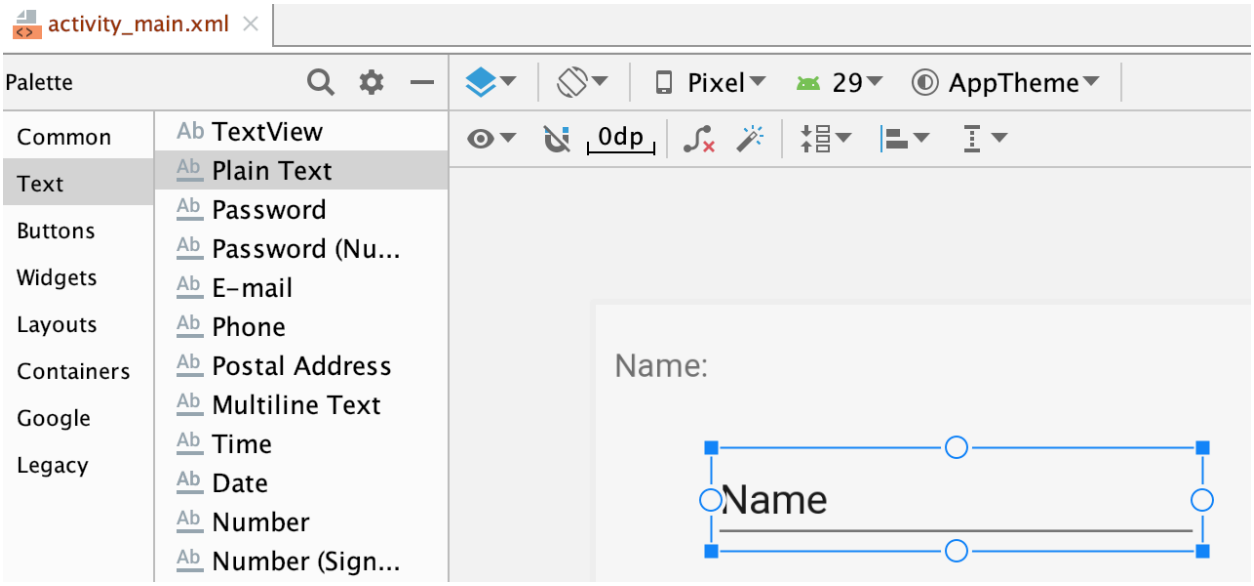
- Espaçamento superior com o valor 16;
- Espaçamentos laterais com valor 8;
- Mude o tipo do espaço lateral para `Match Constraints`. Isso pode ser feito clicando-se no desenho marcado em azul na figura anterior, entre os dois *combo boxes*.

Essas configurações farão com que o componente fique afastado de 16 dp do topo, bem como 8 dp das margens laterais. Além disso, o componente irá se adaptar às variações de largura, de acordo com o tamanho da tela ou sua orientação.

Ainda com o `TextView` selecionado, altere o atributo `text`, localizado na seção `Common Attributes` na aba `Attributes` para o valor `Name:` . Esse é o valor que será exibido para o usuário na interface gráfica.

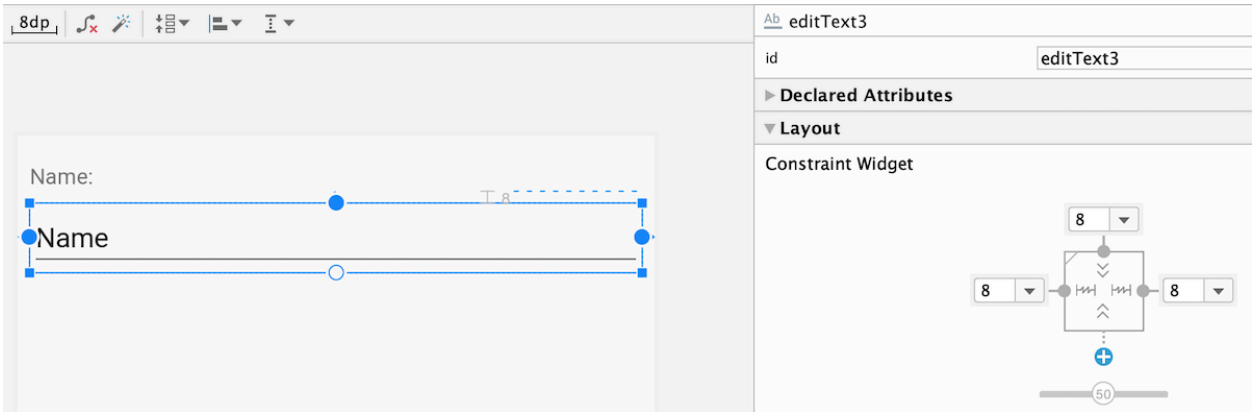
Como esse componente não será referenciado pelo código Kotlin, não é necessário definir um identificação única para ele, além da que o próprio Android Studio já definiu.

O próximo componente a ser posicionado na tela é uma caixa de texto que poderá ser editada pelo usuário. Esse componente chama-se `EditText` , mas aparece com o nome `Plain Text` na seção `Text` da paleta de componentes. Selecione-o e arraste-o para logo abaixo do `TextView`:



Caixa de texto

Conecte os círculos laterais às bordas da tela, da mesma forma como foi feito com o `TextView`. Em seguida, conecte o círculo de sua parte superior ao círculo da parte inferior do `TextView`:



Constraints do EditText

Da mesma forma como foi feito com o `TextView`, o novo `EditText` terá sua largura adaptável ao tamanho da tela e sua orientação. Porém, ele sempre estará próximo ao `TextView`, fazendo que a interface gráfica se mantenha coerente independente dessas mudanças.

Existem apenas mais duas configurações a serem feitas nesse `EditText`:

- Remova o valor do atributo `text`, que está o valor `Name`. Esse é o local onde o usuário irá digitar o nome do produto, por isso deve estar vazio desde o início;
- Configure o atributo `id`, localizado no topo da aba `Attributes`, para o valor `edtName`. Esse será o valor no qual esse componente poderá ser referenciado de dentro do código Kotlin.

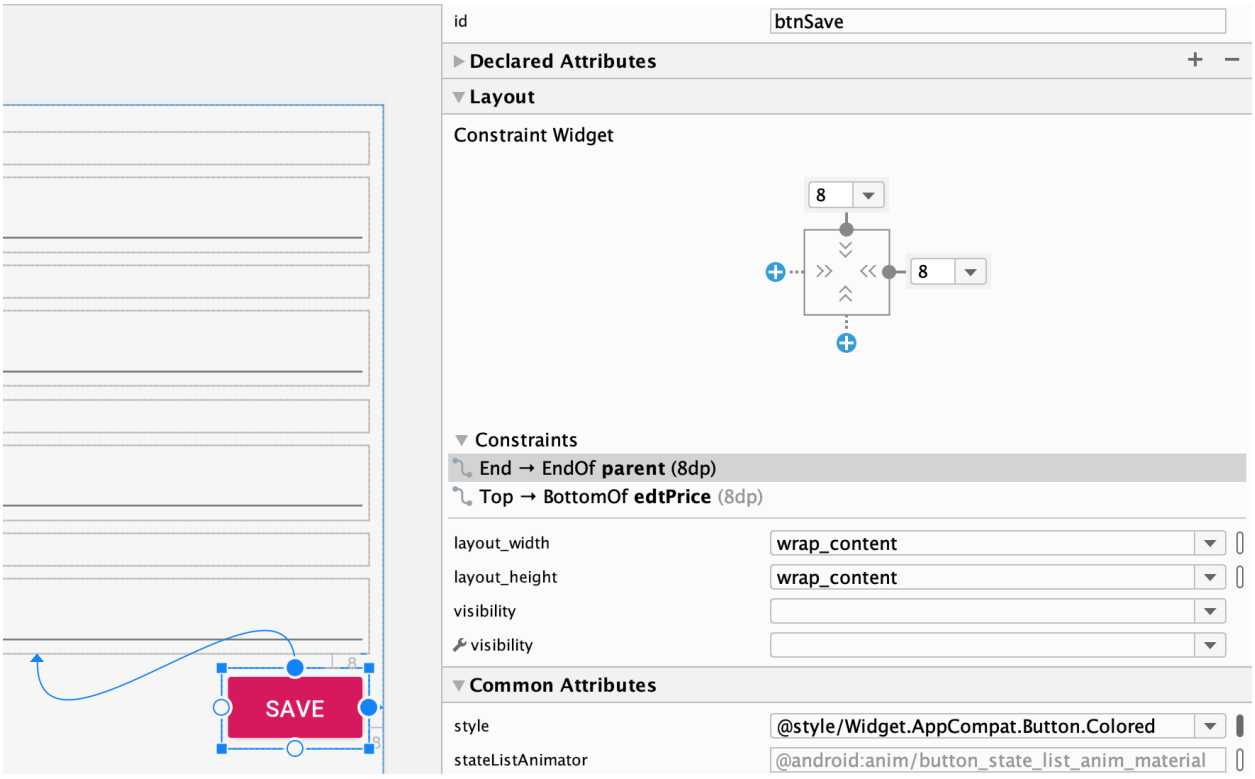
A ideia é ler o valor que o usuário digitou nesse campo, por isso ele deve possuir uma identificação única para ser localizado de dentro do código Kotlin.

Acrescente mais 3 conjuntos de `TextView` e `EditText` para representar os campos de descrição, código e preço do produto. Configure o atributo `id` dos componentes `EditText` seguintes com os seguintes valores:

- Descrição do produto: `edtDescription`
- Código do produto: `edtCode`
- Preço do produto: `edtPrice`

O componente para edição do preço do produto merece uma atenção especial, afinal, seria interessante exibir um teclado diferente ao usuário, uma vez que ele irá digitar um valor decimal. Isso pode ser feito configurando o atributo `inputType` para o valor `numberDecimal`. Isso fará com que o usuário possa digitar apenas números decimais para o preço, ao invés de letras, o que faz todo o sentido.

Logo abaixo do último `EditText`, que é do preço, posicione um botão, como na figura a seguir:



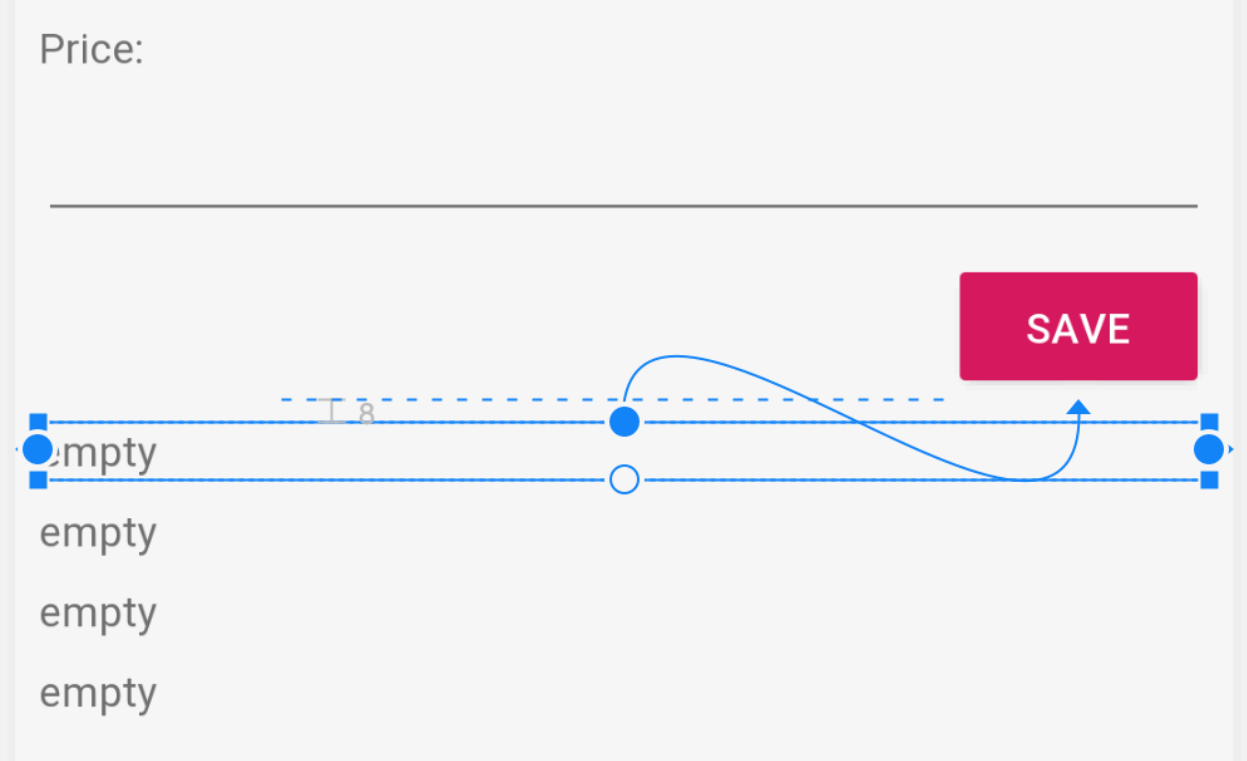
Posicionando o botão

Repare que o botão está está conectado à última caixa de texto e à borda da tela, como pode ser observado na seção `Layout` da seção `Attributes`. Veja também seu `id` foi definido para `btnSave`, pois será necessário referenciá-lo no código Kotlin.

Para deixar o botão com esse estilo, basta alterar o atributo `style` para o valor `@style/Widget.AppCompat.Button.Col`
Esse é um dos estilos pré-definidos do Android.

Além disso, configure também seu texto para `Save`.

Agora posicione mais 4 `TextView` logo após o botão. Eles servirão para exibir os detalhes do produto, depois que o usuário pressionar o botão `Save`. Veja como deve ficar:



Detalhes do produto

Coloque um valor fixo para o atributo `text` desses novos `TextView`. Isso dará um efeito didático para a demonstração do funcionamento do aplicativo, da forma como ele será construído.

Defina um `id` para cada desses novos 4 `TextView`:

- Nome do produto: `txtName`
- Descrição do produto: `txtDescription`
- Código do produto: `txtCode`
- Preço do produto: `txtPrice`

Definir uma identificação única para esses novos 4 `TextView` é essencial, pois eles terão seus valores modificados pelo código Kotlin, quando o usuário clicar no botão `Save`.

Ainda existem algumas configurações necessárias para essa tela, que são:

- Colocar todos os componentes dentro de um outro componente capaz de fazer a tela rolar, caso ela seja exibida em um dispositivo pequeno ou quando simplesmente não houver espaço para mostrar todos os componentes;
- Transformar as strings que aparecem na tela, em recursos capazes para facilitar a tradução.

Essa última ação é algo que deve ser feito desde o início do desenvolvimento de um aplicativo, ainda que a ideia de torná-lo disponível em outros idiomas esteja distante.

O componente capaz de fazer a tela rolar chama-se `ScrollView`, e deve englobar todos os elemento da tela, inclusive o componente do tipo `ConstraintLayout`. Para fazer isso, estando com o arquivo `activity_main.xml` aberto, mude para a aba `Code`, localizada na parte superior direita da tela, para poder alterar o arquivo XML manualmente.

Agora vá até o início do arquivo e altere-o para ficar como o trecho 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">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:context=".MainActivity">
```

Da mesma forma, altere o fim do arquivo para finalizar as tags XML corretamente para esses dois componentes:

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

Lembre-se de ajustar a tabulação do arquivo o final das alterações.

Essa modificação faz com que todos os componentes da tela, incluindo o gerenciador de layouts `ConstraintLayout` fique dentro de `ScrollView`, tornando a tela capaz de rolar caso não haja espaço para mostrar todos os componentes.

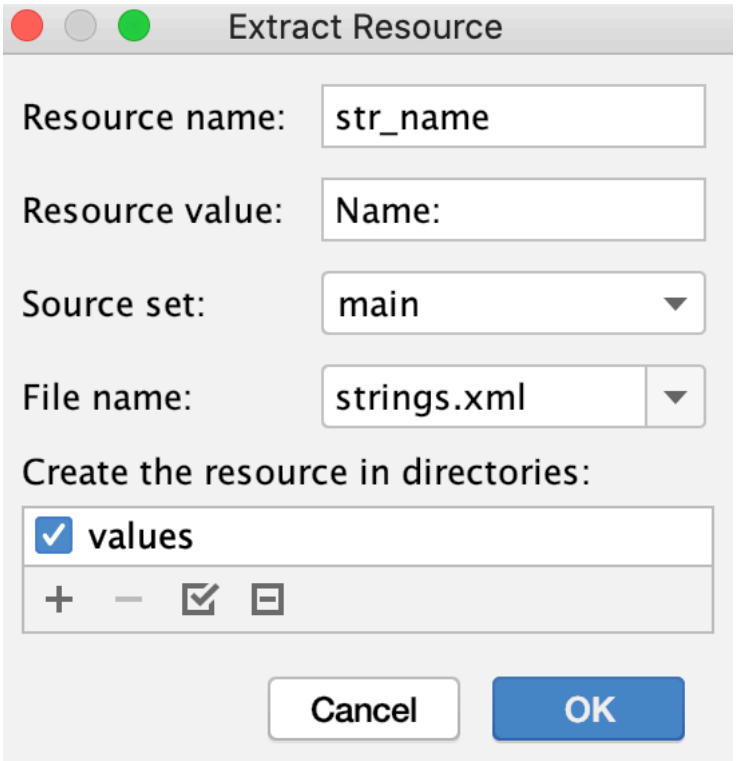
A segunda alteração que deve ser feita no arquivo XML, é a extração das strings que estão fixas no código, tornando-as recursos que podem ser traduzidos para outros idiomas. Perceba que o Android Studio ressalta essas strings no arquivo `activity_main.xml`, como mostra a figura a seguir:

```
13
14      <TextView
15          android:id="@+id/textView"
16          android:layout_width="0dp"
17          android:layout_height="wrap_content"
18          android:layout_marginStart="8dp"
19          android:layout_marginTop="16dp"
20          android:layout_marginEnd="8dp"
21          android:text="Name:"
22          app:layout_constraintEnd_toEndOf="parent"
23          app:layout_constraintStart_toStartOf="parent"
24          app:layout_constraintTop_toTopOf="parent" />
25
```

Strings com valores fixos

Da forma como está, não é possível traduzir o aplicativo para outros idiomas. Então, para extrair essa e as demais strings para a pasta de recursos, clique na lâmpada amarela localizada à esquerda e escolha a opção `Extract string resource`.

Na tela que aparecer, digite o nome do novo recurso, por exemplo `str_name`. Perceba que o arquivo com o nome onde recurso será salvo é `strings.xml`, localizado na pasta `res\values` do projeto:



Criando o recurso de string

Se o projeto já tiver mais que um idioma, ele aparecerá na lista na parte inferior dessa tela. Clique em `OK` e repita o processo para os demais atributos `text` desse arquivo, até que todas as strings que estão fixas nesse arquivo sejam extraídas.

Em seguida, vá até o arquivo `res\values\strings.xml` e perceba que as strings que foram extraídas agora estão lá:

```
<resources>
    <string name="app_name">AndroidProject01</string>
    <string name="str_name">Name:</string>
    <string name="str_description">Description:</string>
    <string name="str_code">Code:</string>
    <string name="str_price">Price:</string>
    <string name="str_save">Save</string>
    <string name="str_empty">empty</string>
</resources>
```

Esse então é o arquivo padrão para os recursos de string. Cada idioma que for acrescentado ao projeto, terá um arquivo igual a esse, traduzido para esse idioma.

O processo descrito aqui para a criação de interfaces gráficas será repetido novamente em ouros capítulos. Por isso, caso tenha alguma dúvida posterior de como isso deve ser feito, volte a essa seção para relembrar.

Para verificar como a tela ficou, execute a aplicação no emulador ou em um dispositivo real. Gire o aparelho e verifique que é possível rolar a tela, para exibir os componentes que possivelmente não cabem na tela.

Para visualizar como o arquivo `activity_main.xml` deve ficar, caso tenha dúvidas, consulte o projeto nos extras que podem ser baixados junto com o livro.

6.5 - Criando o comportamento da interface gráfica

Agora que a interface gráfica já foi construída, é hora de adicionar algum comportamento a ela. A ideia por enquanto é simples, para seguir com a didática proposta no capítulo:

- O usuário irá digitar os detalhes do produto desejado nos 4 campos superiores da tela;
- Quando finalizar, ele deverá clicar no botão `Save`;
- Essa ação fará com que a aplicação leia os dados digitados pelo usuário e exiba-os adequadamente nas caixas de texto abaixo do botão `Save`.

Não se preocupe se esse aplicativo não tem uma função prática. O objetivo é mostrar como criar um código em Kotlin capaz de interagir com a tela e as ações do usuário.

Para começar, vá até o arquivo `MainActivity`. Ele deve estar semelhante ao trecho de código a seguir:

```
class MainActivity : AppCompatActivity() {

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

Essa classe representa uma `Activity` no Android, que é responsável por exibir e definir os comportamentos de uma tela da aplicação. Porém, ela não é uma classe qualquer, pois ela estende de `AppCompatActivity`, como pode ser visto em sua declaração. Isso faz com que ela tenha métodos especiais que serão invocados pelo sistema Android.

Um dos métodos especiais dessa classe é o `onCreate`. Ele é chamado pelo sistema Android toda vez que a `Activity` deve ser criada, que pode acontecer em algumas situações como:

- Quando o usuário deseja executar a aplicação, clicando no seu ícone;
- Quando o usuário gira o aparelho, colocando-o em modo paisagem;
- Quando o usuário altera a configuração de idioma do aparelho e a tela precisa ser redesenhada.

As duas últimas condições são chamadas de **alteração de configuração da tela**. Quando isso acontece, a tela precisa ser redesenhada totalmente, pois houve uma configuração em que não é possível alterar sua aparência sem destruí-la e recriá-la novamente. O capítulo seguinte irá tratar esse assunto com um pouco mais de detalhe. O importante agora é saber que esse método é chamado quando a tela precisa ser criada.

Veja também que o método `onCreate` recebe um parâmetro chamado `savedInstanceState` do tipo `Bundle`. Ele também será detalhado no capitulo seguinte.

Dentro desse método existe a instrução `setContentView`, que recebe como parâmetro um recurso de layout para ser exibido, que é justamente a tela `activity_main` que estava sendo construída na seção anterior desse capítulo. Essa instrução é a responsável por, finalmente, exibir a tela dentro da `Activity` para o usuário.

6.5.1 - View binding

Agora que a tela já está sendo exibida para o usuário, pode-se começar a criar o código para interagir com ela, mas para isso é necessário criar as referências dos componentes gráficos dentro da classe `ActivityMain`. Dessa forma, o código em Kotlin poderá manipulá-los.

Existem algumas formas diferentes para a criação das referências dos componentes gráficos dentro do código Kotlin. Uma das formas mais modernas até o momento é a utilização de uma técnica chamada `View binding`. Para implementar essa técnica é necessário realizar os seguintes passos:

a) Indicar ao Android Studio que o projeto deve utilizar a técnica:

Para isso, abra o arquivo `build.gradle` , vá até a seção `android` e acrescente a subseção `dataBinding` no final, como no trecho a seguir:

```
    dataBinding {
        enabled = true
    }
}
```

O Android Studio solicitará que o projeto seja sincronizado novamente, através de uma mensagem no canto superior direito da tela. Execute esse passo para que o projeto seja preparado para se trabalhar com `View Binding`.

b) Envolver o componente `ScrollView` de `activity_main` em um componente do tipo `layout`:

Para isso, abra o arquivo `activity_main.xml` e altere seu início para ficar com o trecho a seguir:

```
<layout 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">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

Da mesma forma, altere seu fim para que as tags do XML fechem corretamente:

```
    </ScrollView>
</layout>
```

Isso fará com que o Android Studio gere as classes necessárias para representar todos os componentes dessa tela, facilitando muito o trabalho a ser desenvolvido na classe `MainActivity`.

c) Criar o atributo `ActivityMainBinding` na classe `MainActivity`:

Depois que os passos anteriores foram realizados, o Android Studio gera automaticamente uma classe do tipo `ActivityMainBinding` , que contém a referência para todos os componentes gráficos da tela definida em `activity_main.xml`. Os identificadores únicos que foram definidos nesse arquivo são utilizados para definir os nomes das referências dentro de `ActivityMainBinding`.

Agora, basta criar um atributo desse tipo dentro da classe `MainActivity`, como no trecho a seguir:

```
import br.com.siecola.androidproject01.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
```

A última linha do trecho anterior é a criação do atributo do tipo `ActivityMainBinding`, da classe `MainActivity`. Repare que ele está com o modificador `lateinit`, para garantir o Kotlin que ele será inicializado antes de ser utilizado. Sem isso, o compilador geraria um erro.

Finalizando, para que esse atributo seja inicializado, é necessário associá-lo com o recurso de layout de tela que será exibido. Isso deve ser feito dentro do método `onCreate`, logo após a instrução `super.onCreate(savedInstanceState)`, como no trecho a seguir:

```
binding = DataBindingUtil.setContentView(this, R.layout.activity_main)
```

Essa instrução deve substituir a que estava em seu lugar:

```
setContentView(R.layout.activity_main)
```

Ou seja, essa linha deve ser removida dessa função.

A partir desse ponto, o atributo `binding` possui todas as referências dos componentes criados em `activity_main.xml`, e podem ser utilizados para a definição do comportamento dessa tela.

6.5.2 - Criando o comportamento da tela

As seguintes ações podem ser seguidas para definir o comportamento da tela de uma aplicação Android:

- Reagir ao evento de clique em um botão;
- Ler uma caixa de texto editável com o valor digitado pelo usuário;
- Atualizar um conteúdo em uma caixa de texto;
- Alterar uma imagem em um `ImageView`.
- E vários outros.

Cada evento pode ser tratado e implementado de forma diferente, dependendo do componente e do evento, mas a ideia principal é indicar ao componente que existe uma função especial que deve ser invocada quando o evento acontecer.

Para exemplificar, pegue a referência do botão `btnSave` dentro do atributo `binding` da classe `ActivityMain` , criado em `activity_main.xml` e configure-o para executar uma função que imprime uma mensagem de log, como no trecho a seguir, que deve ser colocado no final da função `onCreate`:

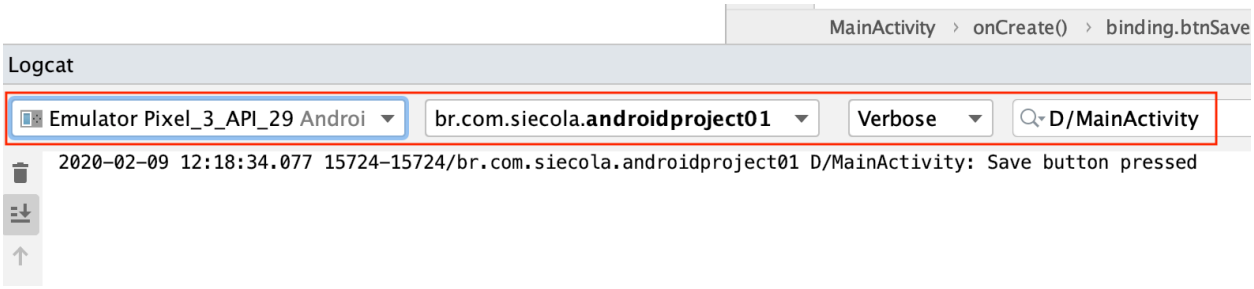
```
binding.btnSave.setOnClickListener {
        Log.d("MainActivity", "Save button pressed")
}
```

Basicamente, esse trecho executa os seguintes passos:

- Pega o atributo `binding`, que possui todas as referências dos componentes gráficos criados em `activity_main.xml`;
- Acessa o componente de identificação `btnSave`;
- Define o comportamento para o *listener* invocado quando o botão é clicado para executar a função anônima delimitada por `{ }`;
- Dentro dessa função anônima, a instrução `Log.d("MainActivity", "Save button pressed")` imprime uma mensagem de log.

A classe `Log` é a responsável por imprimir mensagens no `Logcat` do Android Studio. O primeiro parâmetro é chamado de **tag**, que é utilizado para diferenciar mensagens de contextos diferentes. Normalmente esse parâmetro leva o nome da classe em que está sendo executado. O segundo parâmetro é a mensagem em si.

Para testar essa implementação, execute a aplicação novamente. Assim que ela estiver rodando, clique no botão `Save` e veja que o log foi impresso no `Logcat`, localizado nessa aba na parte inferior do Android Studio:



Exibindo uma mensagem de log

Tenha certeza de selecionar o dispositivo correto, bem como o nome do aplicativo e configurar o filtro de mensagens para `D/MainActivity`, que é o nível de log em debug e o nome da classe que o gerou.

Agora que o evento de clique no botão `btnSave` está funcionando, é possível finalizar a implementação desse aplicativo, exibindo os detalhes do produto, que foram digitados pelo usuário, nas caixas de texto localizadas após o botão de salvar. Para isso, acrescente o seguinte trecho de código na função que trata o evento de clique do botão `btnSave`:

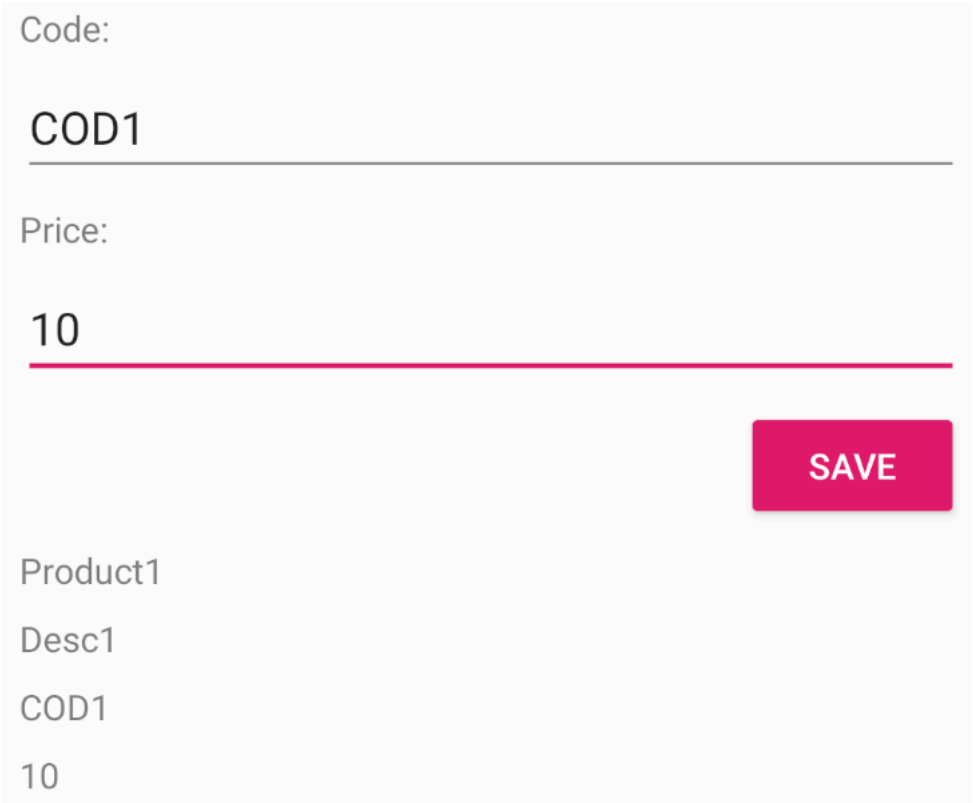
```
binding.txtName.text = binding.edtName.text
binding.txtDescription.text = binding.edtDescription.text
binding.txtCode.text = binding.edtCode.text
binding.txtPrice.text = binding.edtPrice.text
```

Como o atributo da classe `binding` possui a referência de todos os componentes da tela, basta acessar cada uma das caixas de texto editáveis, os `EditText`, buscando pela propriedade `text` e atribuir seu valor ao `TextView` correspondente. Isso fará com que a informação digitada pelo usuário seja exibida no final da tela.

Veja como deve ficar toda a classe `MainActivity`:

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import androidx.databinding.DataBindingUtil
import br.com.siecola.androidproject01.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(this, R.layout.activity_main)

        binding.btnSave.setOnClickListener {
            Log.d("MainActivity", "Save button pressed")

            binding.txtName.text = binding.edtName.text
            binding.txtDescription.text = binding.edtDescription.text
            binding.txtCode.text = binding.edtCode.text
            binding.txtPrice.text = binding.edtPrice.text
        }
    }
}
```

Para testar a implementação, execute novamente a aplicação, preencha os campos com os detalhes de um produto qualquer e clique no botão `Save`. Os campos de texto deverão ficar preenchidos com tais informações, como mostra a figura a seguir:



Testando a aplicação

Essa aplicação poderia ficar melhor. Pelo menos poderia tratar a condição quando o usuário não digitar um valor para o nome do produto, considerando que somente ele fosse necessário. Nesse caso, uma mensagem de erro poderia ser lançada na tela. Para isso, acrescente uma condição para validar esse teste:

```
if (!binding.edtName.text.isEmpty()) {
    binding.txtName.text = binding.edtName.text
    binding.txtDescription.text = binding.edtDescription.text
    binding.txtCode.text = binding.edtCode.text
    binding.txtPrice.text = binding.edtPrice.text
} else {
    Toast.makeText(this, "Please, enter the name.", Toast.LENGTH_SHORT).show()
}
```

Na primeira linha desse trecho, foi verificado se o atributo `text` do componente `edtName` não está vazio. Se ele não estiver, significa que o usuário digitou algo nesse campo e então os detalhes do produto podem ser lidos.

Caso não haja informação dentro do atributo `text` de `edtName`, uma mensagem é lançada na tela, através da instrução `Toast.makeText`. Essa instrução recebe os seguintes parâmetros:

- Uma referência da `Activity` na qual a mensagem deve aparecer;
- A mensagem de texto em si;
- A duração em que a mensagem deve ficar exibida na tela.

Por último, o método `.show()` é chamado para exibir a mensagem na tela ao usuário.

Por fim, para manter a estratégia coerente de possibilitar que esse aplicativo seja traduzido para outros idiomas, é necessário adicionar a string `Please, enter the name` como um recurso do projeto. Isso pode ser feito colocando o cursor sobre esse texto e acessando a opção `Extract string resource` da lâmpada amarela que aparecer do lado esquerdo. Para finalizar, basta digitar um nome para o recurso, por exemplo `str_enter_name`.

Com essa última modificação, o Android Studio se encarrega de alterar o código Kotlin para buscar a string do recurso definido, como pode ser visto no trecho a seguir:

```
Toast.makeText(this, getString(R.string.str_enter_name), Toast.LENGTH_SHORT).show()
```

Quando esse aplicativo for traduzido para outros idiomas, o Android se encarregará de pegar o texto no idioma correto para ser exibido.

É importante lembrar que o código do projeto está no GitHub e pode ser acessado aqui[16].

6.6 - Conclusão

Esse capítulo apresentou alguns conceitos iniciais de um projeto Android, bem como os passos para a construção de interfaces gráficas utilizando o `ConstraintLayout`, o que faz com que a tela fique adaptável a diversos tamanhos de dispositivos.

Também apresentou o conceito de `View binding`, uma técnica moderna que facilita a associação e utilização das referências dos componentes gráficos dentro do código em Kotlin.

Por fim, iniciou o leitor na construção de comportamentos da aplicação, com interação direta com a interface gráfica do usuário.

O próximo capítulo traz conceitos mais aprofundados sobre `Activity`, introduzindo conceitos de ciclo de vida e gerenciamento de seus estados.

[16] https://github.com/siecola/android_project01