COMPUTAÇÃO PARA DISPOSITIVOS MÓVEIS

CAPÍTULO 1 – VAMOS CRIAR UM APLICATIVO MÓVEL?

Evandro de Souza Lima Rocha

INICIAR

Introdução

Os dispositivos móveis estão presentes na vida pessoal e profissional das pessoas, mas você consegue mensurar isso? Segundo uma pesquisa do IBGE (2016), o uso de dispositivos móveis, para navegar na internet, já é maior que o de computador pessoal (PC). Em cerca de 97% dos domicílios com acesso à internet, o *smartphone* (dispositivo móvel mais popular) é utilizado. E cerca de 92,4% dos habitantes que acessam a internet no Brasil (116 milhões de habitantes), utilizaram aplicativos móveis.

A grande procura por aplicações móveis ocorre devido à crescente necessidade que as pessoas têm, de coleta e acesso às informações de forma rápida, correta, intuitiva e segura, a qualquer momento e em qualquer localização, o que torna cada vez mais importante a criação de aplicativos. É por isso que o desenvolvimento de aplicativos móveis é uma promissora habilidade para despontar no mercado de trabalho.

Considerando a importância dessas aplicações na vida das pessoas, como começo a desenvolver aplicativos móveis?

Neste capítulo, vamos compreender o universo da computação móvel e começar a desenvolver aplicativos. Para isso, vamos identificar e analisar os principais dispositivos, plataformas de *software*, ferramentas, padrões, técnicas e métodos voltados para a computação móvel, presentes no mercado. Assim, este conhecimento é utilizado para a construção e entendimento de nossa primeira aplicação.

Vamos entender como funciona o desenvolvimento de aplicações móveis e criar um aplicativo? Bons estudos!

1.1 Introdução ao desenvolvimento de aplicativos

Antes de começar a desenvolver aplicações, é importante entendermos o universo da computação e dos dispositivos móveis. Este tópico visa responder às seguintes perguntas: que características possuem os dispositivos móveis? Quais são as principais plataformas de *software* utilizadas neste meio e suas características? Quais padrões são utilizados neste ecossistema de aplicações? Acompanhe!

1.1.1 Dispositivos móveis

Desde o final dos anos 1990, a popularização e modernização das tecnologias, para comunicação em redes sem fio, proporcionaram maior conectividade aos dispositivos e possibilitaram o acesso a informações remotas, onde quer que se esteja, favorecendo a computação móvel.

A computação móvel é um paradigma computacional que possibilita, às pessoas, o acesso a serviços em qualquer localização, mesmo em movimento, ou seja, a função essencial da computação móvel é permitir às pessoas o acesso à informação em qualquer lugar e a qualquer momento.

Nessa linha, o dispositivo móvel é qualquer equipamento computacional que possua mobilidade, para se transportar, possibilitando a seus usuários, o acesso a serviços computacionais independentes de localização a qualquer momento. Segundo Lee (*et al.,* 2005), as principais características dos dispositivos móveis são:

- **portabilidade**: esta característica está ligada diretamente ao tamanho e peso do dispositivo, quanto mais leve e menor um dispositivo for, mais portável ele é, pois permite que seja facilmente deslocado;
- **usabilidade**: capacidade de ser utilizado pelas pessoas em diversos ambientes;
- **funcionalidade**: oferece serviços ao usuário em aplicações móveis;
- **conectividade**: comunicação com outros sistemas e dispositivos, transmitindo ou recebendo informações.

Analisando essas características, podemos perceber a enorme variedade de dispositivos móveis presentes em nossa sociedade como, por exemplo: telefones inteligentes (*smartphones*), *tablets,* computadores portáteis (*notebooks*), *videogames* portáteis (PS Vita, Nintendo 3DS), câmeras e filmadoras digitais, *players* de multimídia portáteis (Ipod, MP4, MP3), relógios inteligentes (*smartwatches*), óculos de realidade virtual, etc.

A grande variedade de dispositivos móveis está associada a plataformas e arquiteturas de desenvolvimento móvel distintas, sendo algumas, até mesmo, fechadas para o grande público de desenvolvedores. Então, a fim de abordar o desenvolvimento de aplicações de uma forma prática, para que consigamos desenvolver nossas próprias aplicações, vamos nos concentrar no desenvolvimento para *smartphones,* devido à enorme popularidade e versatilidade destes equipamentos.

Antes do *smartphone,* veio o telefone celular, que tinham poucas funções, como envio de voz (chamadas eletrônicas), e funções básicas de dados (acesso precário à internet por *wap* e envio e recebimento de mensagens por SMS e MMS). Considerados pela indústria como uma nova era na evolução dos telefones celulares, os *smartphones* são telefones celulares com um sistema operacional ou plataforma que permite a instalação, atualização e remoção de aplicativos móveis.

Quando surgiram os primeiros *smartphones,* não havia tantas funcionalidades e facilidade de acesso às aplicações móveis quanto hoje. A primeira geração destes aparelhos se iniciou no ano 2000, com o lançamento do Ericsson R380 que rodava a plataforma Symbian OS. Nesta época, os celulares em geral, não tinham tela *touch,* nem lojas de aplicativos centralizadas. As principais plataformas eram:

Symbian, Palm, Windows Mobile e Blackberry.

A partir de 2007, impulsionados pelo lançamento do iPhone, surgiu uma nova era de *smartphones* com tela *touch,* como principal característica.

VOCÊ O CONHECE?

Steven Paul Jobs (1955-2011) é considerado um gênio da área de tecnologia por ter inovado nos setores de computação, animação no cinema, música, telefonia celular, *tablets*, varejo e publicação digital. Com a criação do iPhone, revolucionou o mundo dos *smartphones*, mudando totalmente a forma como as pessoas interagiam com os dispositivos móveis.

Quando as lojas centrais de aplicativos, como a App Store, se popularizaram em um novo modelo, Fling (2009) definiu esta, como a era *touch* na evolução dos telefones celulares.

1.1.2 Principais plataformas móveis

Muitas plataformas surgiram na era *touch,* como Android, iOS, Windows Phone, Bada, *web* OS, Tizen, Firefox OS. Porém apenas os sistemas Android e iOS possuem quantidade relevante no mercado de *smartphones*. Segundo pesquisa da consultoria Gartner (2018), as plataformas Android e iOS dominam o mercado de *smartphones,* estando presentes em 99,9% dos dispositivos vendidos.

Plataforma	2017 (unidades)	2017 (Percentual do mercado)
Android	1.320.118,1	85,9%
iOS	214.914,4	14,0%
Outras plataformas	1.493,0	0,1%
Todas	1.536.535,5	99,9%

Figura 1 - Principais plataformas móveis no mercado. Fonte: Gartner (2018).

#PraCegoVer: Apresenta uma tabela composta por 5 linhas e 3 colunas. Na primeira linha temos os nomes das 3 colunas: primeira coluna: plataforma,

segunda coluna: unidades em 2017 e a terceira coluna: percentual do mercado em 2017. Na segunda linha a plataforma Android possui 1.320.118,1 unidades e um percentual do mercado de 85,9%. Na terceira linha a plataforma iOS possui 214.914,4 unidades e um percentual do mercado de 14,0%. Na quarta linha a plataforma Outras plataformas possui 1.493,0 unidades e um percentual do mercado de 0,1%. Na quinta linha o total de todas as plataformas possui 1.536.535,5 unidades e um percentual do mercado de 99,9%.

Mas, antes de desenvolver nossos aplicativos, devemos entender cada uma das plataformas relevantes no mercado. Nesta seção, é apresentada uma visão geral das plataformas Android e iOS.

Android

Segundo o Google (2018), a plataforma Android é uma pilha de *softwares* baseada no sistema operacional Linux, de código-fonte aberto, e foco em atender dispositivos móveis. A versão mais recente deste sistema, em 2019, é a 8.1 (Oreo). As versões ativas do Android em dispositivos móveis podem ser vistas na figura abaixo.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.5%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.3%
5.0	Lollipop	21	4.8%
5.1		22	17.6%
6.0	Marshmallow	23	25.5%
7.0	Nougat	24	22.9%
7.1		25	8.2%
8.0	Oreo	26	4.9%

Figura 2 - Fragmentação de versões da plataforma Android. Fonte: Google (2018).

#PraCegoVer: Apresenta uma tabela composta por 9 linhas e 4 colunas. A primeira linha indica os nomes das colunas; Primeira coluna: Version, segunda coluna: Codename, terceira coluna: API e quarta coluna: Distribuition; Segunda linha: Version: 2.3.3 – 2.3.7, Codename: Gingerbread, API: 10, Distribuition: 0.3%. Terceira linha: Version: 4.0.3 – 4.0.4, Codename: Ice Cream Sandwich, API: 15, Distribuition: 0.4%. Quarta linha: Version: 4.1.x, Codename: Jelly Bean, API: 16, Distribuition: 1.5%; Version: 4.2.x, Codename: Jelly Bean, API: 17, Distribuition: 2.2%; Version: 4.3, Codename: Jelly Bean, API: 18, Distribuition: 0.6%. Quinta linha: Version: 4.4, Codename: KitKat, API: 19, Distribuition: 10.3%. Sexta linha: Version: 5.0, Codename: Lollipop, API: 21, Distribuition: 4.8%; Version: 5.1, Codename: Lollipop, API: 22, Distribuition: 17.6%; Sétima linha: Version: 6.0, Codename: Marshmallow, API: 23, Distribuition: 25.5%. Oitava linha: Version: 7.0, Codename: Nougat, API: 24, Distribuition: 22.9%; Version: 7.1, Codename: Nougat, API: 25, Distribuition: 8.2%. Nona linha Version: 8.0, Codename: Oreo, API: 26, Distribuition: 4.9%;

Para acompanhar o percentual de mercado de cada versão da plataforma Android rodando em dispositivos móveis, você pode acessar o link <https://developer.Android.com/about/dashboards/?hl=pt-br (https://developer.Android.com/about/dashboards/?hl=pt-br)>.

VOCÊ QUER VER?

Quer conhecer a história do Android e visualizar um breve histórico das versões das plataformas Android? O vídeo *A história do* Android *– Tecmundo* (KLEINA, 2017), disponível no endereço <https://www.youtube.com/watch?v=5K4pEk19nhs (https://www.youtube.com/watch?v=5K4pEk19nhs)> apresenta esta plataforma da origem e da evolução das versões até chegar na Oreo.

O ambiente de desenvolvimento Android é baseado nas linguagens de programação Java e Kotlin. Este ambiente não requer um sistema operacional específico, podendo ser instalado nos sistemas Windows, MAC OS e Linux. O Android Studio é a ferramenta oficial recomendada pelo Google para desenvolvimento de aplicações Android.

Para a publicação na loja de aplicativos do Google, deve ser criada uma conta e realizado um pagamento de uma taxa única de inscrição ao programa de desenvolvedores Android. Antes de serem publicados, os aplicativos passam por um processo de revisão para assegurar que não violam as políticas estabelecidas pela loja.

iOS

iOS é a abreviatura para *Iphone Operation System,* sendo desenvolvido pela Apple. Esta plataforma é baseada no Sistema Operacional MAC OS X e foi projetada para atender às necessidades exclusivas de aparelhos móveis desenvolvidos pela Apple (iPhone, iPod, iPad).

O sistema operacional iOS é uma plataforma proprietária e sua versão mais recente, em 2018, é o iOS11. A figura, abaixo, apresenta a fragmentação das versões ativas da plataforma iOS.

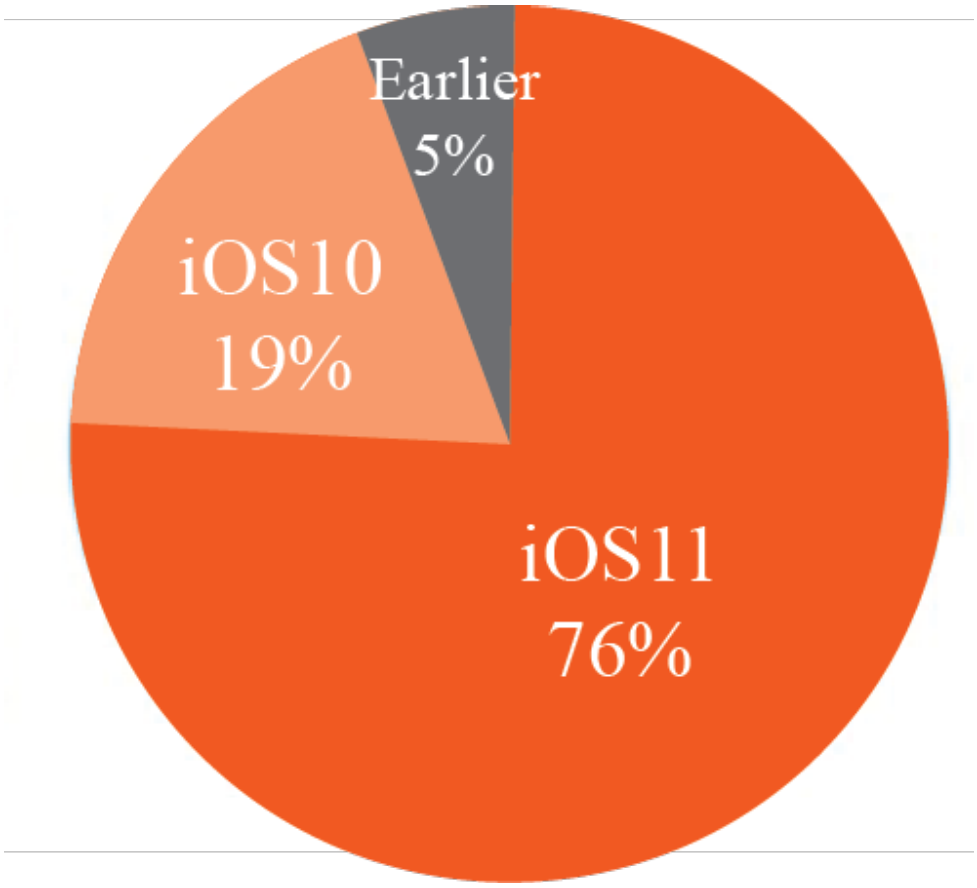


Figura 3 - Fragmentação de versões da plataforma iOS. Fonte: Owen (2018, [s./d.]).

#PraCegoVer: Apresenta um gráfico de pizza com 3 fatias, uma fatia é chamada de iOS11 e representa 76%, a outra fatia é chamada de iOS10 e representa 19% e a outra fatia é chamada de Earlier e representa 5%.

O desenvolvimento de aplicações para esta plataforma, requer um computador com a plataforma macOS e o ambiente de desenvolvimento Xcode, que permite que os desenvolvedores criem suas aplicações e as testem em emuladores baseados no sistema iOS. O desenvolvimento é atrelado a duas linguagens de programação: o Objective-C e o Swift.

VOCÊ SABIA?

A linguagem de programação Swift foi lançada pela Apple para suceder o Objective-C.S. Segundo o ranking “IEEE Spectrum ranking – IEEE” (DIAKOPOULOS; CASS, 2018), a linguagem Swift já é a décima mais popular do mundo. Saiba mais em: <https://spectrum.ieee.org/static/interactive-the-

top-programming-languages-2017 (https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017)>.

Para a utilização de recursos avançados do Xcode e a distribuição na *App Store*, loja de aplicativos da Apple, é exigida uma licença no *Apple Developer Program*.

Além da licença citada, é necessário que os aplicativos submetidos à loja de aplicativos, estejam de acordo com as diretrizes estabelecidas pela Apple, como as orientações de interface gráfica e as orientações de revisão da *App Store.*

Comparação entre as plataformas móveis apresentadas

O quadro a seguir, resume as principais diferenças entre as plataformas Android e iOS para desenvolvedores individuais de aplicativos móveis.

Características	Android	iOS
Diversidade de dispositivos	Alta. Inúmeros dispositivos de variadas fabricantes atingindo as várias camadas da sociedade	Baixa. Pouca variedade de dispositivos. Apenas iPhone, iPad e iPod *touch* rodam esta plataforma
Ambiente de desenvolvimento (SDK)	Aplicações nativas podem ser desenvolvidas em computadores com os sistemas Windows, Linux e macOS	Aplicações nativas devem ser desenvolvidas utilizando o sistema macOS X
Licença de desenvolvedor	Uma taxa de única inscrição no valor de U$ 25,00.	Taxa paga anualmente no valor de U$ 99,00
Publicação de aplicações nas lojas oficiais de aplicativos da plataforma	Termos mais simples de aprovação, favorecendo a uma aprovação rápida do aplicativo	Os termos da Apple para aprovação de um aplicativo são rigorosos, normalmente leva um tempo bem mais longo de aprovação que nas demais plataformas
Linguagens oficiais de programação	Java e Kotlin	Objective-C e Swift
Fragmentação de versões presentes no mercado	Alta. Devido a cada fabricante de dispositivo disponibilizar atualização do Android como queira	Baixa. Todos os dispositivos iPhone, a partir do iPhone 5S, permitem atualização para o iOS, que proporciona uma concentração maior de dispositivos nesta versão
Licença da plataforma	Código aberto	Sistema fechado e proprietário

Quadro 1 - Comparação entre as principais plataformas móveis. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Apresenta uma tabela composta por 8 linhas e 3 colunas. Primeira linha temos os nomes das 3 colunas: primeira coluna: Características, segunda coluna: Android e a terceira coluna: iOS. Segunda linha: Características: Diversidade de dispositivos, Android: Alta. Inúmeros dispositivos de variados fabricantes atingindo as várias camadas da sociedade. iOS: Baixa. Pouca variedade de dispositivos. Apenas iPhone, iPad e iPod touch rodam esta plataforma. Terceira linha: Características: Ambiente de desenvolvimento (SDK), Android: Aplicações nativas podem ser desenvolvidas em computadores com os sistemas Windows, Linux e macOS. iOS: Aplicações nativas devem ser desenvolvidas utilizando o sistema macOS X. Quarta linha: Características: Licença de desenvolvedor,

Android: Uma taxa de única inscrição no valor de U$ 25,00. iOS:Taxa paga anualmente no valor de U$ 99,00. Quinta linha: Caracaterísticas: Publicação de aplicações nas lojas oficiais de aplicativos da plataforma, Android: Termos mais simples de aprovação, favorecendo a uma aprovação rápida do aplicativo, iOS: Os termos da Apple para aprovação de um aplicativo são rigorosos, normalmente leva um tempo bem mais longo de aprovação que nas demais plataformas. Sexta linha: Características: Linguagens oficiais de programação, Android: Java e Kotlin, iOS: Objective-C e Swift. Sétima linha: Características: Fragmentação de versões presentes no mercado, Android: Alta. Devido a cada fabricante de dispositivo disponibilizar atualização do Android como queira, iOS: Baixa. Todos os dispositivos iPhone, a partir do iPhone 5S, permitem atualização para o iOS, que proporcionam uma concentração maior de dispositivos nesta versão. Oitava linha: Características: Licença da plataforma, Android: Código aberto, iOS: Sistema fechado e proprietário.

São vários fatores que diferenciam estas plataformas, o que dificulta a vida dos desenvolvedores, quando desejam criar aplicações que rodem nestas duas plataformas. Na próxima seção, entenderemos os métodos para trabalhar com aplicações multiplataforma.

1.1.3 Abordagens de desenvolvimento de aplicativos móveis

Cada plataforma é associada a padrões, ferramentas de desenvolvimento e linguagens de programação próprios, tornando custosa e trabalhosa, a criação de aplicações para mais de uma plataforma. Porém, um componente é comum em todas elas: a visualização de páginas *web,* que possibilita a execução de aplicações *web,* independente de plataforma.

Assim, existem três abordagens para a construção de aplicativos: abordagem nativa, abordagem *web* móvel e abordagem híbrida (mescla entre as abordagens nativa e *web* móvel).

Abordagem de desenvolvimento de aplicações nativas

As aplicações móveis, ditas nativas, são construídas com o uso dos SDK's, disponibilizados pelas plataformas, nas quais a aplicação deve ser executada. A grande desvantagem deste método de desenvolvimento é que existe baixa reutilização de código entre plataformas, fazendo com que o volume de trabalho e manutenção das aplicações aumente consideravelmente, na medida em que é

necessário suportar uma nova plataforma.

Existem *frameworks* no mercado que amenizam o retrabalho, ao desenvolver aplicativos nativos para múltiplas plataformas. Os principais presentes no mercado são o Xamarin e React Native. Eles proporcionam um ambiente único de desenvolvimento para ambas as plataformas, utilizando uma única linguagem de programação.

Os fatores que justificam o uso desta abordagem são (SANTOS, 2014):

- a experiência do usuário é fator crítico do projeto, que justifica o tempo e recursos despendidos em solução nativa para cada plataforma;
- necessidade de fazer uso massivo dos recursos nativos de cada plataforma suportada;
- alta necessidade de otimização no uso de recursos de *hardware*, como CPU, memória e processamento gráfico;
- o projeto tem como alvo apenas uma plataforma móvel;
- a aplicação deve ser instalada localmente e disponibilizada nas lojas de aplicativos oficiais.

Abordagem de desenvolvimento de aplicações *web* móveis

As plataformas atuais têm, por padrão, um navegador *web* para acessar a internet. A abordagem de desenvolvimento de aplicações *web* móveis foi criada com o propósito de diminuir o tempo e custo no desenvolvimento de uma aplicação. Por meio deste método, as aplicações são criadas, utilizando tecnologias *web* (HTML, CSS, Javascript, entre outros) e não são instaladas nos dispositivos móveis, ou seja, são hospedadas num servidor *web*. A única diferença deste tipo de aplicação para um *web site* comum é o foco nos dispositivos móveis, fazendo tratamentos específicos para as plataformas móveis e resoluções de tela. Os fatores que justificam esta abordagem são (SANTOS, 2014)

- a aplicação necessita operar em diferentes plataformas e existem restrições críticas de tempo e orçamento;
- a aplicação não tem requisitos para funcionar *offline*, ou seja, só deve

funcionar com acesso à internet;
- a performance e experiência do usuário não são itens críticos do projeto;
- não existe necessidade de fazer uso dos recursos nativos de cada plataforma;
- não existe necessidade de disponibilizar aplicação nas lojas de aplicativos oficiais.

Abordagem de desenvolvimento de aplicações híbridas

Com a combinação das abordagens de desenvolvimento nativo e *web*, surge a abordagem de aplicações híbridas. Da mesma forma que as aplicações *web* móveis, as aplicações híbridas fazem uso do motor *web* de cada plataforma.

A grande particularidade neste método de desenvolvimento é que os desenvolvedores utilizam um *framework* que encapsula, em código padronizado Javascript, as chamadas APIs nativas de cada plataforma disponibilizando um único ambiente de desenvolvimento baseado em tecnologias *web*.

As aplicações híbridas são empacotadas e armazenadas localmente no dispositivo depois de instaladas e podem ser distribuídas nas lojas oficiais de aplicativos das plataformas móveis, assim como as aplicações nativas.

Os fatores que justificam esta abordagem são (SANTOS, 2014):

- a performance e fluidez do aplicativo não são requisitos estritamente *necessários* . Embora instalada no dispositivo, a aplicação faz uso do motor *web* para renderização da interface de usuário que não é tão rápido quanto o acesso dos componentes de IUs nativos;
- a equipe de desenvolvimento detém conhecimento de tecnologias *web*;
- a aplicação necessita usar recursos nativos, mas o ciclo de desenvolvimento precisa ser curto;
- não há necessidade de otimização no uso de recursos de *hardware* como CPU, memória e processamento gráfico;
- a aplicação deve ser instalada localmente e disponibilizada nas lojas de aplicativos.

Comparação entre as abordagens

O quadro a seguir, apresenta um resumo de como relacionam fatores importantes no desenvolvimento de aplicações multiplataforma, com cada abordagem de desenvolvimento.

Característica	Aplicação nativa	Aplicação *web*	Aplicação híbrida
Custo de desenvolvimento e manutenção	Alto	Baixo	Baixo
Tempo de desenvolvimento e manutenção	Alto	Baixo	Baixo
Desempenho	Alto	Baixo	Médio
Experiência do utilizador da aplicação móvel	Ótima	Baixa	Média
Reutilização de código em diferentes plataformas	Não	Sim	Sim
Acesso aos recursos do dispositivo	Total	Baixo	Parcial
Necessidade de conexão à internet	Não	Sim	Não
Canais de Distribuição	Lojas de aplicativo	*Web site*	Lojas de aplicativo

Quadro 2 - Comparação entre as abordagens de desenvolvimento.

#PraCegoVer: Apresenta uma tabela composta por 9 linhas e 4 colunas. Primeira linha temos os nomes das 4 colunas: primeira coluna: Características, segunda coluna: Aplicação nativa e a terceira coluna: Aplicação web e quarta coluna: Aplicação híbrida. Segunda linha: Características: Custo de desenvolvimento de manutenção, Aplicação nativa: Alto, Aplicação web: Baixo, Aplicação híbrida: Baixo. Terceira linha: Características: Tempo de desenvolvimento e manutenção, Aplicação nativa: Alto, Aplicação web: Baixo, Aplicação híbrida: Baixo. Quarta

linha: Características: Desempenho, Aplicação nativa: Alto, Aplicação web: Baixo, Aplicação híbrida: Médio. Quinta linha: Características: Experiência do utilizador da aplicação móvel, Aplicação nativa: Ótima, Aplicação web: Baixa, Aplicação híbrida: Média. Sexta linha: Características: Reutilização de código em diferentes plataformas, Aplicação nativa: Não, Aplicação web: Sim, Aplicação híbrida: Sim. Sétima linha: Características: Acesso aos recursos do dispositivo, Aplicação nativa: Total, Aplicação web: Baixo, Aplicação híbrida: Parcial. Oitava linha: Características: Necessidade de conexão à Internet, Aplicação nativa: Não, Aplicação web: Sim, Aplicação híbrida: Não. Nona linha: Características: Canais de Distribuição, Aplicação nativa: Lojas de Aplicativos, Aplicação web: Web site, Aplicação híbrida: Lojas de aplicativo.

Desta forma, a escolha do método de desenvolvimento para utilizar em aplicações móveis, não é trivial. Cada abordagem possui vantagens e desvantagens, que vão depender da natureza e requisitos de cada aplicativo.

1.2 Projeto de aplicativo

Como projetar um aplicativo? Ao desenvolver uma solução móvel, existem diversos desafios e preocupações muito importantes no paradigma da computação móvel, mas que são pouco relevantes no paradigma de desenvolvimento para *desktop* ou puramente *web*.

Neste tópico são apresentados aspectos mais gerais que devem ser levados em conta, na concepção de uma nova aplicação. Dentre esses aspectos, é ressaltada a preocupação de projetar interfaces de usuário condizentes com os padrões de *design* de soluções móveis.

Por fim, são apresentadas técnicas para explorar um projeto de interface de usuário, sem que seja necessária ainda, sua codificação, tornando o processo de validar ideias e projetos, mais ágil e menos custoso.

1.2.1 Arquitetura de uma solução móvel

Ao iniciar um projeto de aplicativo para dispositivos móveis, devemos nos atentar a diversas complicações decorrentes da grande variedade disponível e ritmo acelerado de crescimento e evolução deste mercado (LEE *et al.*, 2005):

- **diversidade de recursos físicos**: os aplicativos móveis devem se adequar aos diversos tamanhos de tela e resoluções dos dispositivos e serem fluídos em uma enorme variedade de configurações de CPU e memória;
- **tratar conexões de rede sem fio**: os aplicativos móveis devem se adaptar às diversas conexões com rede sem fio (*Wi-fi*, 2G, 3G, 4G, *Bluetooth*, etc.), que costumam ser intermitentes, de baixa velocidade e com alta latência;
- **dificuldade de entrada de dados**: os aplicativos móveis devem possuir interfaces de entrada de dados simplificadas e rápidas devido à grande diversidade de contextos de uso e atenção limitada do usuário;
- **energia limitada**: os aplicativos móveis devem oferecer uma diversidade de recursos, consumindo o mínimo possível de energia do dispositivo;
- **manutenção e atualização dos aplicativos**: os aplicativos móveis demandam constante manutenção para se adequarem às rápidas evoluções dos sistemas operacionais móveis que, normalmente, trazem novos padrões de codificação e recursos a cada versão.

Quais são os elementos envolvidos num projeto de aplicativo? Lee *et al*. (2005) apresenta múltiplos fatores que devem ser analisados num projeto de aplicativo, representados na arquitetura de solução móvel apresentada na figura abaixo.

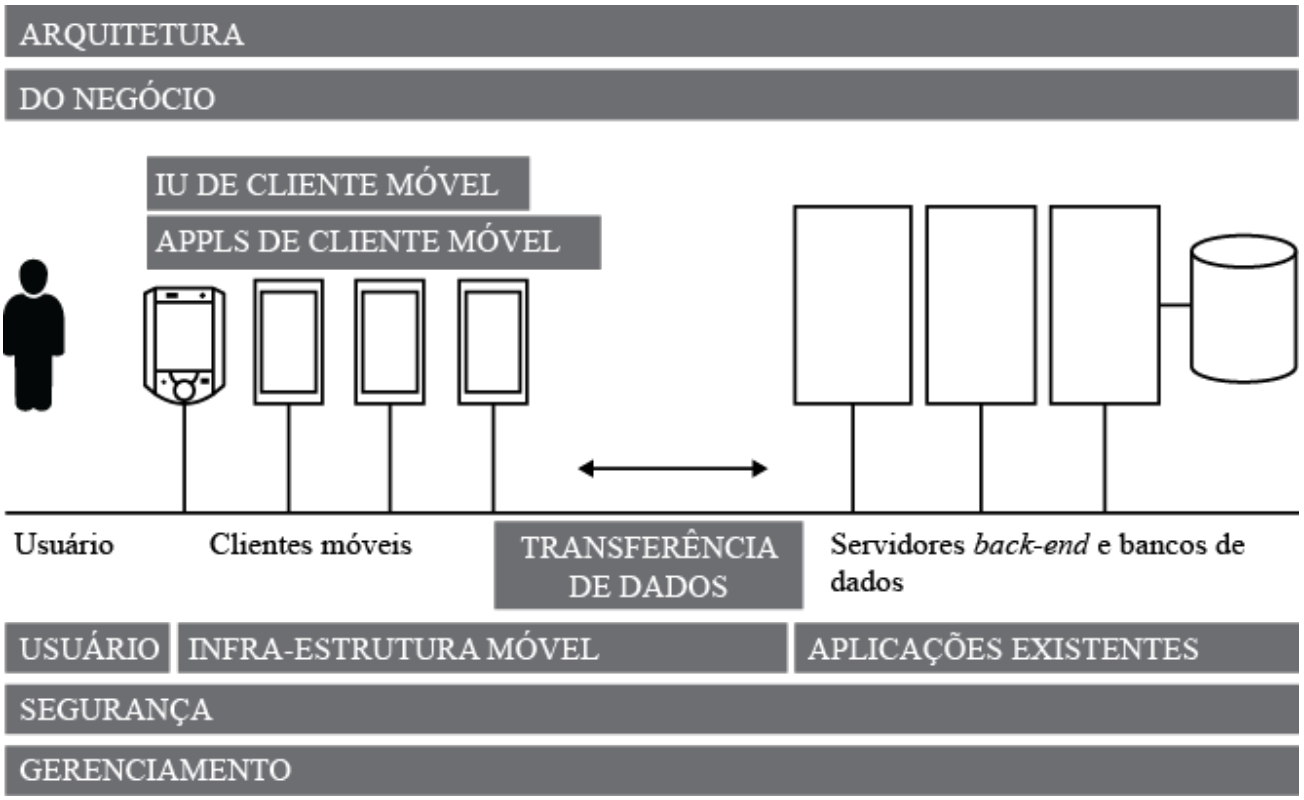


Figura 4 - Arquitetura de solução móvel. Fonte: LEE et al. (2005, p. 7).

#PraCegoVer: Apresenta a arquitetura do negócio. Um boneco pintado todo de preto representa o usuário, do lado direito tem se o IU de Cliente móvel – Appls de cliente móvel representado com o desenho de 4 celulares (clientes móveis), em seguida tem-se a transferência de dados em via dupla entre os clientes móveis com os servidores back-end (Representados por 3 retângulos na vertical) e bancos de dados (representado por um cilíndro). Abaixo de toda a figura tem-se 3 linhas: na primeira linha tem-se a palavra usuário abaixo do boneco preto, depois a infra-estrutura móvel que engloba os clientes móveis e a transferência de dados e por fim as aplicações existentes que engloba os servidores back-end e bancos de dados; na segunda linha tem-se a segurança e na terceira linha o gerenciamento, ambas englobam todo o processo desde o usuário, clientes móvies, transferência de dados e servidores back-end até o bancos de dados.

Estes fatores definidos por Lee *et al.* (2005), têm as seguintes funções:

- **contexto do negócio**: identificar as reais necessidades e objetivos dos usuários – alvo da aplicação – se será uma aplicação para eles se

comunicarem, como ferramenta de trabalho, para entretenimento, para fins educacionais, etc.;

- **arquiteturas de aplicação móvel**: analisar qual arquitetura de aplicativo adotar, ou seja, tomar decisões, como definir a abordagem de desenvolvimento (nativa, *web*, híbrida) e quais *frameworks* utilizar para desenvolver o aplicativo;
- **infraestrutura móvel**: definir quais dispositivos móveis (*tablets*, telefones celulares, relógios inteligentes, etc.) e plataformas, a aplicação deverá ser suportada;
- **interface de usuário** (IU de cliente móvel): definir projeto de interface de usuário visando que a aplicação seja intuitiva;
- **transferência de dados**: definir mecanismo de acesso a dados nos servidores *back-end* e banco de dados;
- **segurança**: atentar para a proteção dos dados para casos de perda e roubo do dispositivo. É muito importante manter a privacidade, buscando evitar vulnerabilidades e ameaças;
- **estudos de caso de aplicações móveis**: realizar estudos de caso e análises em aplicativos já existentes, identificando padrões e boas práticas para serem utilizados em novas aplicações.

Esta arquitetura nos mostra uma visão geral da solução móvel completa, que envolve mais que apenas o projeto do aplicativo móvel (apenas a solução cliente), como segurança e conexão com servidores *back-end*, por exemplo. Porém, nesta seção vamos focar apenas no projeto de interface de usuário de cliente móvel.

1.2.2 Projeto de interface de usuário

O projeto de interface do usuário é parte essencial do processo de projeto, para que o *software* possa atingir todo o seu potencial. Este projeto deve combinar as habilidades, experiências e expectativas dos usuários previstos, uma vez que interfaces inadequadas poderão limitar, em vez de ajudar a atingir o objetivo para o qual o sistema foi projetado (SOMMERVILLE, 2011).

Ao projetar aplicações móveis, é necessário entender que elas devem possuir interações rápidas, curtas e focadas. devido aos seguintes fatores:

- a atenção do usuário ser dividida com outras atividades;

- a interação com a aplicação móvel constituir uma tarefa secundária, que não deve interferir na tarefa principal do usuário;
- o usuário está sujeito a diversas interrupções durante a interação.

Assim, a interface projetada deve apresentar claramente a informação, de modo que favoreça a compreensão do usuário, de como ele deve realizar a interação, para propiciar melhor usabilidade. Fonseca (*et al.,* 2010) define princípios que devem existir na concepção de interfaces de usuário em aplicações para propiciar melhor usabilidade.

- **proximidade**: elementos relacionados entre si devem ser agrupados e aproximados uns dos outros, para que sejam vistos como um conjunto coeso e para apontar elementos que não possuam relação lógica, que não devem ser misturados. Isso permite identificar elementos relacionados de forma ágil;
- **alinhamento**: nenhum elemento deve ser colocado arbitrariamente na interface. Cada elemento deve ter uma conexão visual com outro. Assim como o princípio da proximidade, este princípio visa coesão na interface de usuário;
- **repetição**: sempre que possível e relevante, elementos e representações gráficas podem ser repetidos numa interface como espaçamentos, tipografia, espessura, imagens e cores. Seguir este princípio, propicia consistência na interface, visto que a repetição de elementos de forma uniforme ao longo da interface, facilita o reconhecimento por parte do usuário;
- **ordenação**: o princípio da ordenação deve ser aplicado para dispor os elementos numa ordem lógica que seja entendida pelo usuário, evitando confusões e facilitando a informação contida na interface. A lógica de ordenação é dependente da natureza dos elementos e objetivo. Por exemplo, a ordem alfabética é a forma mais simples e, normalmente, aplicada em listas de itens ou nomes.

Além destes princípios, os principais fabricantes de plataformas móveis disponibilizam guias para auxiliar desenvolvedores na construção de interfaces de

usuário. As principais guias são: Android *Style Guide* para sistemas Android e o iOS *Human Interface Guidelines* para sistemas iOS.

1.2.3 Protótipo de interface de usuário

São diagramas estáticos que representam o esqueleto do design de um projeto de aplicativo, para discutir e comunicar o projeto de interface de usuário. Um modelo de representa o de um aplicativo de forma bastante simplificada. Neste modelo não há muita preocupação com critérios estéticos (cor, tamanho da fonte, etc.). Normalmente, há grande predominância das cores preto e branco nestes desenhos, como vemos na figura abaixo.

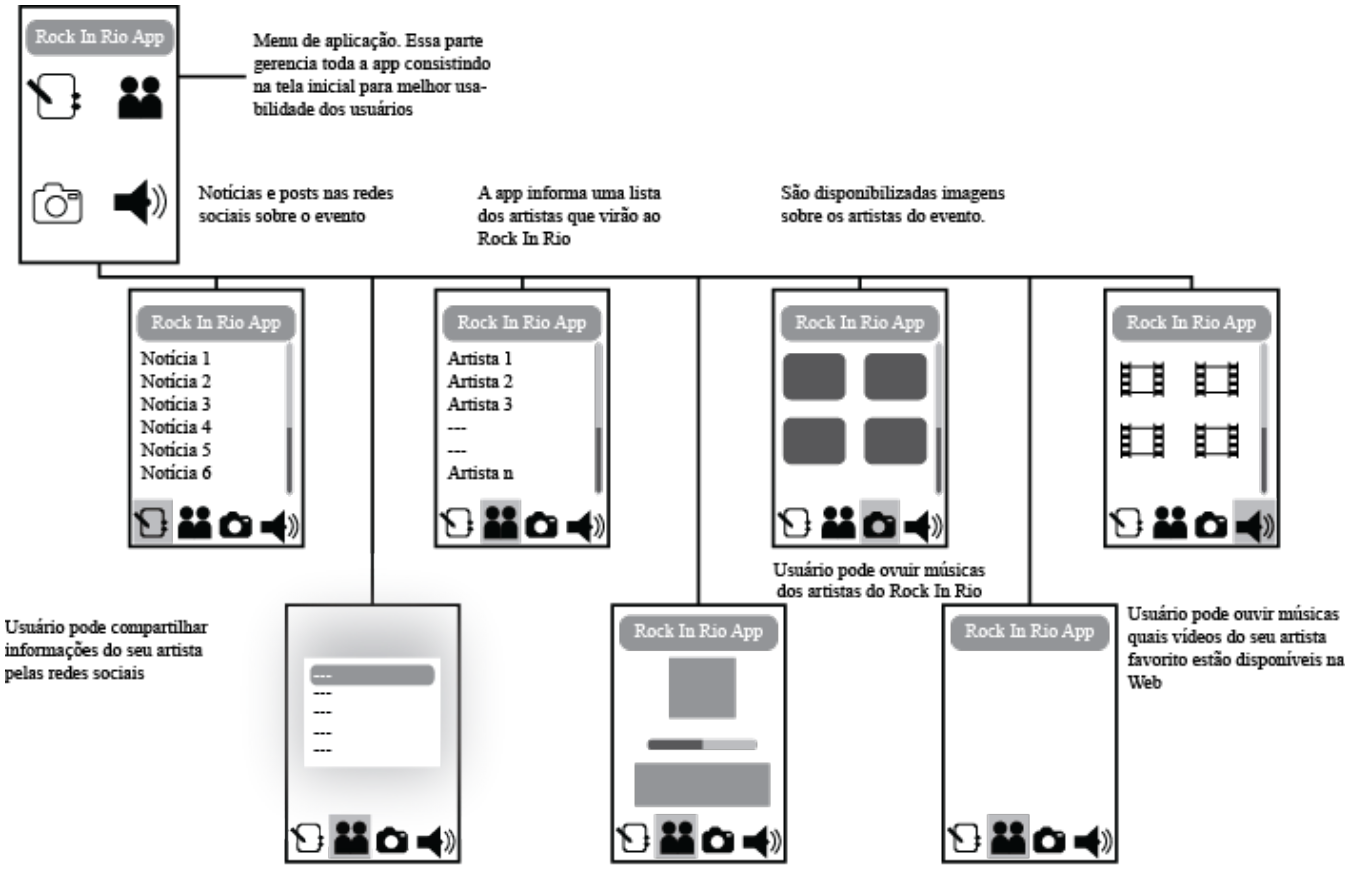


Figura 5 - Wireframe de um aplicativo. Fonte: FONTÃO et al. (2012).

#PraCegoVer: Apresenta o desenho de um celular com a interface do aplicativo Rock in Rio, mostrando o menu de aplicação. Essa parte gerencia toda a App consistindo na tela inicial para melhor usabilidade dos usuários. Conectado a essa interface tem-se outros desenhos de celulares cada um representando um item do menu principal: interface com as notícias e posts nas redes sociais sobre o evento; interface onde o usuário pode compartilhar informações de um artista pelas redes

sociais; interface com a App informando uma lista dos artistas que virão no Rock in Rio; interface onde os usuários podem ouvir músicas dos artistas do Rock in Rio; interface onde são disponibilizadas as imagens sobre os artistas do evento e interface onde o usuário pode ouvir músicas quais vídeos do seu artista favorito estão disponíveis na web.

A representação do projeto de interface de usuário em *wireframes* deve conter a representação de todas as partes importantes do projeto e poderão ser utilizados como parte da documentação.

Mockup

Um *mockup* é um diagrama estático que representa fielmente o *design* de aplicativo. Diferente do *wireframe*, um *mockup* é um modelo bem próximo do *design* final do produto, ou até o próprio *design* visual do produto. Já neste modelo, há uma preocupação bem maior com elementos estéticos, como cores, fontes e contraste, que não acontece nos modelos de *wireframe*.

Figura 6 - Mockup de um aplicativo. Fonte: FONTÃO et al. (2012).

#PraCegoVer: Apresenta o desenho de dois celulares, acima dos desenhos está escrito Dispositivos touch e do lado esquerdo dos desenhos: Resolução de tela: 640 x 360 px (modo paisagem), Profundidade da cor: 24 bits, Plataforma: S60 5ª edição e Browser: Web Runtime 1.1.

Os *mockups* (exemplo na figura acima) são muito utilizados com a função de vender uma ideia de um produto antes de ele estar pronto para um público estratégico (possíveis clientes, por exemplo).

Enfim, um projeto de um aplicativo cliente móvel pode ser construído seguindo as seguintes diretrizes:

- **definir contexto do negócio**: definir os requisitos, ou seja, levantar que

o aplicativo deve fazer;
- **infraestrutura móvel**: definir quais dispositivos e plataformas, o aplicativo deve atender;
- **projeto de interface de usuário**: definir interface de usuário, com a técnica de prototipação;
- **arquiteturas de aplicação móvel**: definir arquitetura de *software* do aplicativo. Sabendo dos requisitos e plataformas, definir qual abordagem de desenvolvimento utilizar ou quais *frameworks* utilizar.

1.3 Introdução ao desenvolvimento Android

Como implementar uma aplicação móvel? Agora que podemos projetar nossos aplicativos, é hora de botar a mão na massa. Pela plataforma Android, nossas aplicações móveis atingirão mais de 80% do mercado de dispositivos móveis (vide a tabela "*Principais plataformas móveis no mercado*"). Esta plataforma admite o uso de duas linguagens oficiais de programação para aplicações nativas: Java e Kotlin.

Além destas duas linguagens, é possível escrever código, utilizando as linguagens C e C++ em aplicações Android, utilizando o NDK (Android *Native Development Kit*), caso necessite (GOOGLE, 2018), para:

- conseguir mais desempenho de um dispositivo para aplicativos de computação intensiva, como jogos ou simulações de física;
- reutilizar bibliotecas C ou C++ ou as de outros desenvolvedores.

Neste tópico é apresentado como implementar e executar um aplicativo utilizando a ferramenta Android Studio.

Para fins didáticos, vamos utilizar apenas a linguagem Java para os exemplos de aplicativos apresentados por ser a linguagem mais difundida no ambiente Android.

O primeiro passo do processo de desenvolvimento é configurar o ambiente que requer o kit de desenvolvimento Java (JDK), cuja versão seja superior a 1.6.

Para identificar a versão da máquina virtual Java instalada, acesse o terminal ou o *prompt* de comando e digite o seguinte comando: java-version

Além disso, é necessária a instalação de um ambiente de desenvolvimento integrado (IDE) na máquina de desenvolvimento. Aqui, abordamos a IDE Android Studio 3.0, pois é a ferramenta oficial para desenvolvimento de aplicativos Android.

CASO

Vamos compreender, na prática, como desenvolver a partir do Android Studio.

Primeiro, faça o *download* do Android Studio, que pode ser realizado no endereço: <https://developer.android.com/studio/install (https://developer.android.com/studio/install)>. A instalação é simples e rápida, mas caso tenha alguma dúvida, pode seguir o manual de instalação do Android na página <https://developer.android.com/studio/install (https://developer.android.com/studio/install)>.

Ao instalar o Android Studio, por padrão, já é instalado o kit de desenvolvimento (SDK) mais atual e as ferramentas básicas (bibliotecas Android, emuladores, ferramentas de *debug*), para a criação de uma aplicação móvel.

Agora, você está pronto para criar um projeto no Android Studio, utilizando as opções. "*File*"/"*New Project*" (Arquivo/Novo projeto). Acompanhe o texto para saber o passo a passo.

Na primeira tela da GUI de criação de projeto (figura abaixo), são preenchidas informações básicas do projeto como nome, domínio e pasta do sistema, no qual ficará armazenado. As opções de adicionar suporte às linguagens de programação só devem ser marcadas, se houver necessidade de escrever partes da aplicação móvel em C/C++ ou em Kotlin.

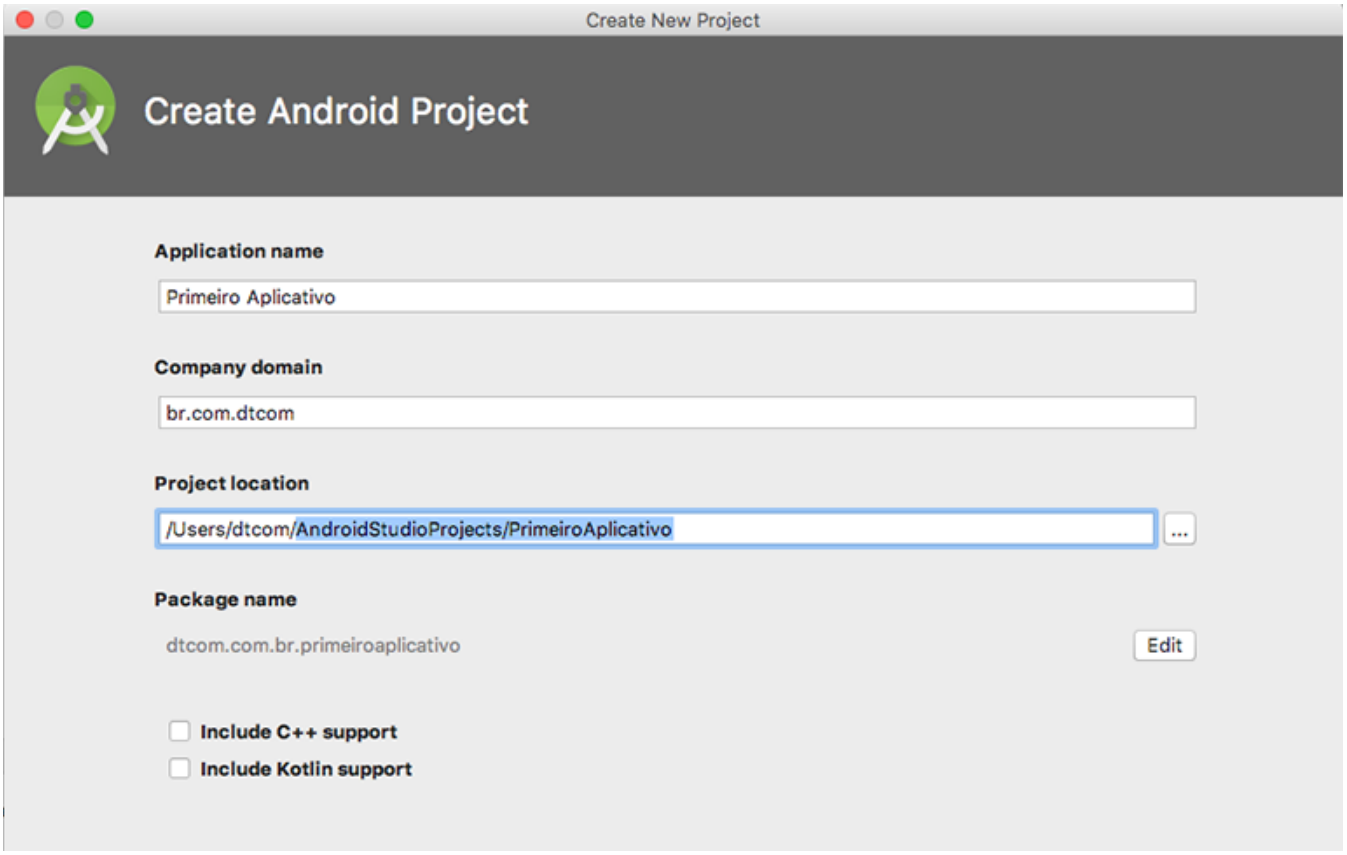


Figura 7 - Primeiro passo na criação de um projeto Android. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela da GUI com o nome de Create Android Project e os campos para preenchimento: Application name: Primeiro Aplicativo, Company domain: br.com.dtcom, Project location: /Users/dtcom/AndroidStudioProjects /PrimeiroAplicativo, Package name: dtcom.com.br.primeiroaplicativo com o botão Edit na frente, opções não selecionadas abaixo: Include C++ support e Include Kotlin suppor.t.

Como todos os códigos do material se focam na linguagem Java, todos os projetos de exemplo apresentados podem ser criados com estas opções desmarcadas.

Na segunda tela (figura abaixo), deverão ser selecionadas as versões mínimas do Android para cada categoria de dispositivo móvel. Por exemplo, se for selecionada a API 15 (referente ao Android 4.0.3) para *smartphones*/ *tablets*, a aplicação só irá executar em *smartphones*/ *tablets* com versão superior a 4.0.3. O Google recomenda esta versão, pois atinge mais de 99% dos *smartphones*/ *tablets* ativos no momento (vide figura “Fragmentação de versões da plataforma Android”).

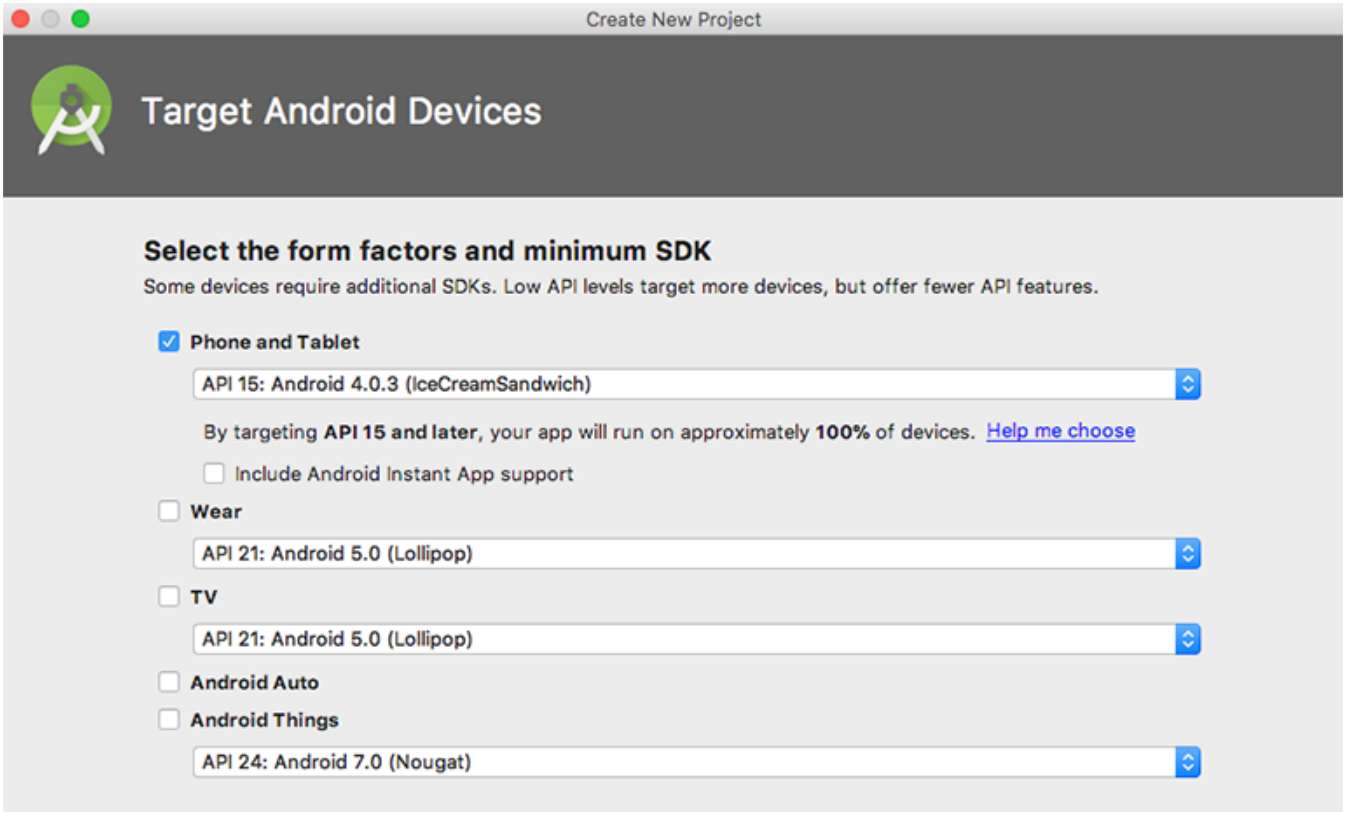


Figura 8 - Segundo passo na criação de um projeto Android. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: print da tela da próxima GUI com o nome de Target Android Devices, abaixo: Select the form factors and minimum SDK, Some devices require assitional SDKs. Low API levels target more devices, but offer fewer API features. Abaixo os campos para preenchimento: Phone and Tablet: API 15: referente ao Android 4.0.3 (IceCreamSandwich), Wear: API 21: Android 5.0 (Lollipop), TV: API 21: Android 5.0 (Lollipop) e Android Things: API 24: Android 7.0 (Nougat).

A opção de criar o aplicativo como *instant app* deve estar desmarcada. O conceito de *instant app* é compatível com versões superiores a 4.2 (API 17) e possibilita que aplicativos nativos sejam executados em dispositivos móveis sem que estejam instalados. Esta arquitetura de aplicativos é mais avançada e não será abordada, por enquanto. Como o foco do nosso projeto são os *smartphones*, vamos marcar apenas esta opção nesta etapa de criação do projeto.

Na terceira tela, é apresentada uma coleção de *templates* de *Activity* para selecionar como tela inicial do aplicativo. Uma *Activity* ou atividade pode ser entendida como uma tela da aplicação. Para cada tela é necessária a criação de, pelo menos, uma *Activity* que permitirá aos utilizadores manipular, adicionar ou

obter informação de uma aplicação. Ou seja, neste passo de inicialização do projeto, estamos definindo como será o *layout* da primeira tela do aplicativo, definida como *Activity* principal.

Neste exemplo (figura abaixo), vamos utilizar o *template empty Activity,* visto que nosso propósito neste exemplo é de entender o funcionamento dos componentes básicos do Android.

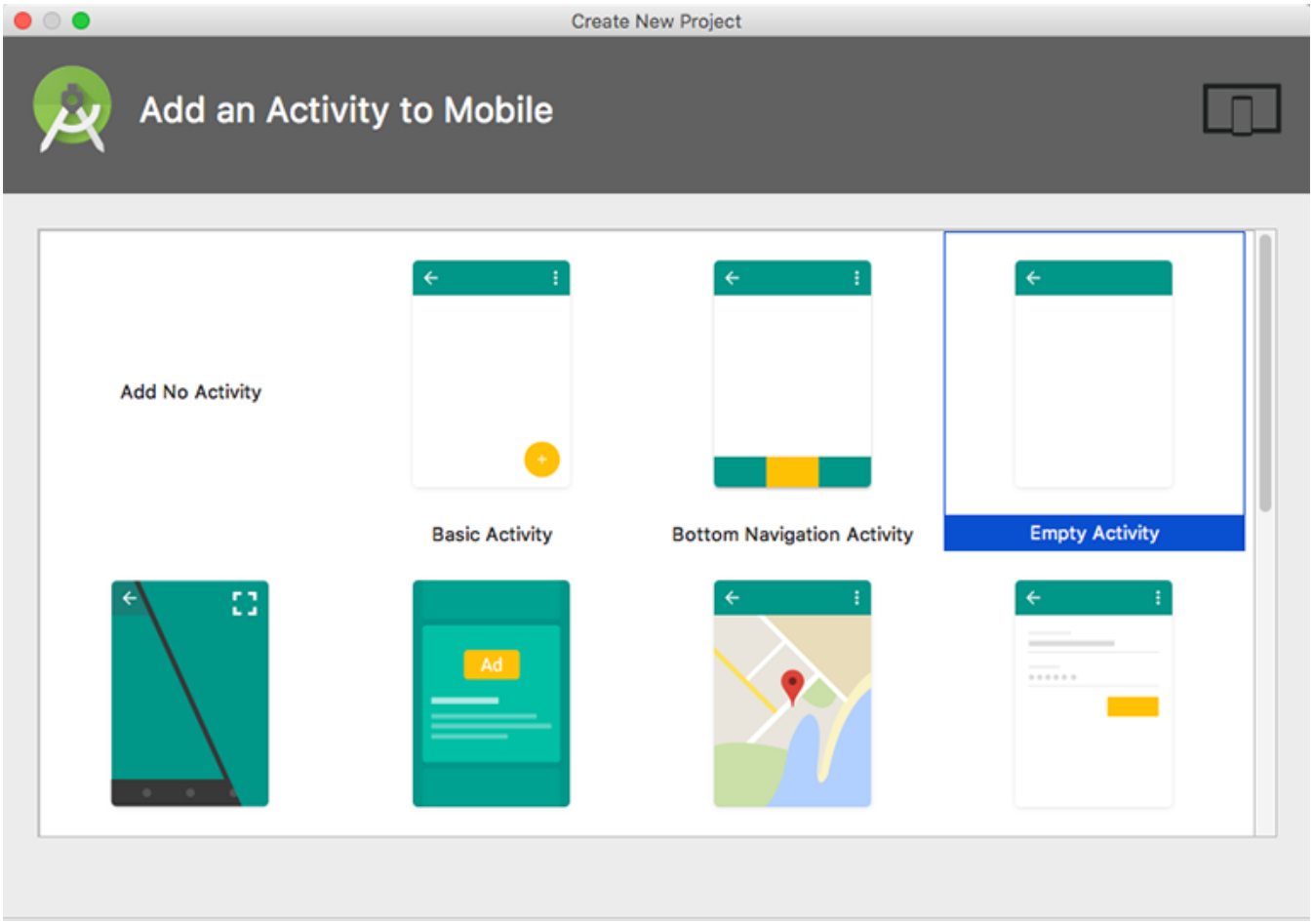


Figura 9 - Terceiro passo na criação de um projeto Android. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela da próxima GUI com o nome Add na Activity to Mobile, abaixo temos as opções: Add No Activity, Basic Activity, Bottom Navigation Activity e Empty Activity (opção selecionada).

Neste exemplo, utilizamos os nomes TelaPrincipalActivity e Activity_tela_principal.xml para os respectivos nomes da *Activity* e do arquivo XML de configuração de interface de usuário.

1.3.1 Entendendo a estrutura de um projeto Android

Depois dos primeiros passos de inicialização do projeto, é criado um projeto básico contendo o padrão de três pastas principais, como mostra a figura abaixo.

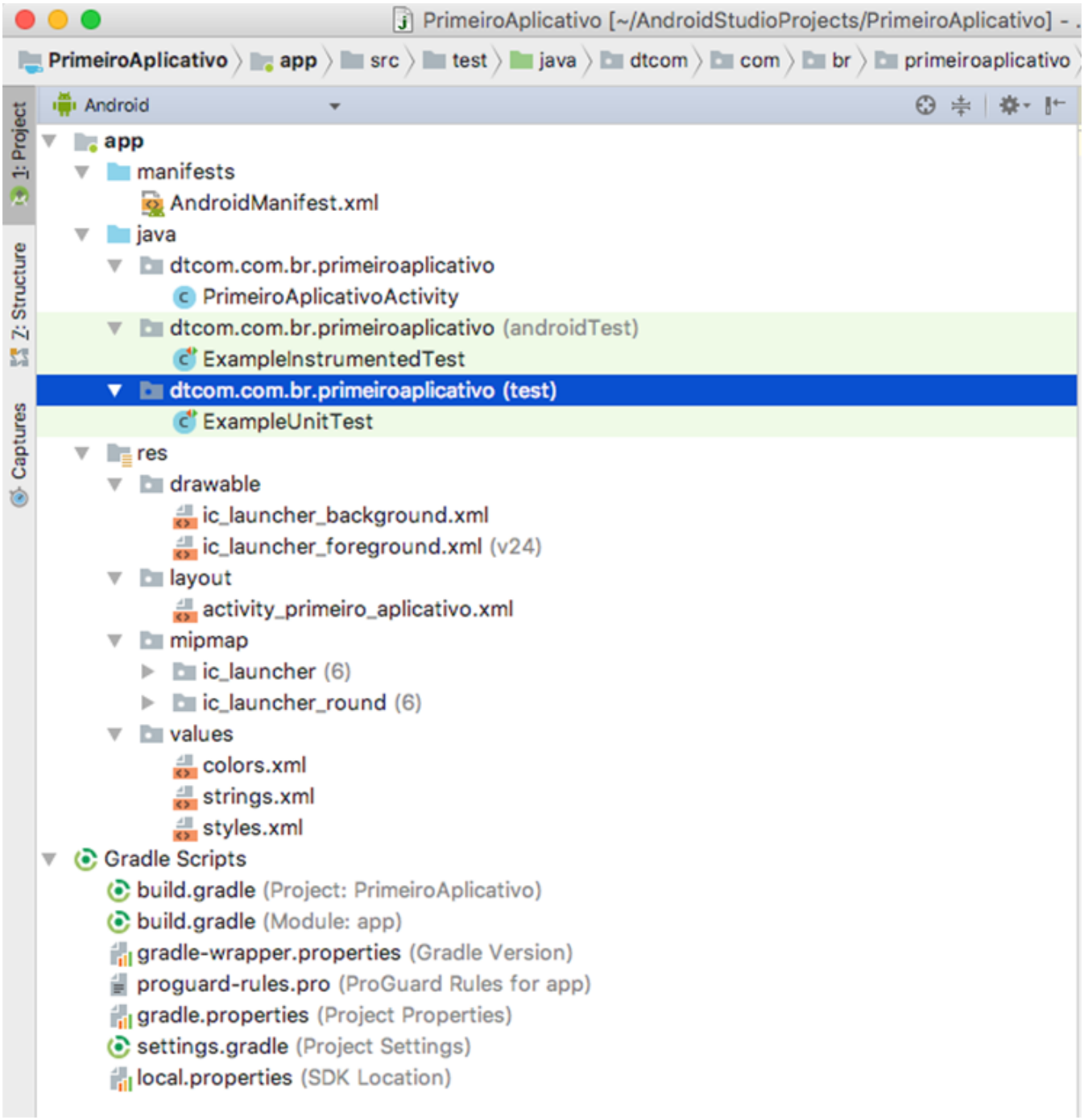


Figura 10 - Estrutura de um projeto Android. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela do projeto básico do Pimeiro Aplicativo com as pastas: manifests, java: dentro dessa pasta tem a opção dtcom.com.br.primeiroaplicativo (test) e a pasta res.

Estas pastas têm as seguintes funções (GOOGLE, 2018):

- *manifests*: pasta contendo diretrizes principais da aplicação. Nesta pasta devem conter arquivos de configurações com caráter global. O projeto é criado com o arquivo AndroidManifest.xml, que apresenta informações essenciais sobre o aplicativo ao sistema Android, necessárias para o sistema antes que ele possa executar o código do aplicativo. Entre outras coisas, o arquivo do *manifest* serve para: nomear o pacote Java (identificador) para o aplicativo, descrever os componentes do aplicativo, determinar os processos que se relacionam com o aplicativo, declarar as permissões que o aplicativo deve ter para acessar partes protegidas da API e interagir com outros aplicativos, declarar nível mínimo de API que o aplicativo deve suportar, listar bibliotecas às quais o aplicativo deve se vincular;
- java: nesta pasta devem estar todos os arquivos de código fonte Java da aplicação, separados por nome de pacote, inclusive o código de teste do JUnit;
- res: contém todos os recursos que não são código, como *layout*s XML, *strings* de IU e imagens em *bitmap*, divididos em subdiretórios correspondentes;
- *scripts Gradle*: Nesta pasta deverá estar o arquivo build.gradle que contém as diretrizes de compilação do projeto e demais *scripts Gradle*.

VOCÊ SABIA?

O Gradle é um sistema avançado de automatização de *builds* (compilação do código fonte em um apk). O processo de compilação de um aplicativo Android envolve muitas ferramentas e etapas que são integradas com o Gradle, tornando ágil e fácil a geração de um apk. Maiores detalhes sobre esta ferramenta podem ser encontradas no endereço: <https://docs.gradle.org/current /userguide/userguide.html (https://docs.gradle.org/current/userguide/userguide.html)>.

Desta forma, é estruturada uma aplicação padrão Android. Nas próximas seções deste tópico, iremos focar nos arquivos de definição de interface de usuário que ficam localizados na pasta res. As configurações globais realizadas no arquivo

AndroidManifest.xml e arquivos fontes java serão explicados mais adiante no tópico 1.4.

1.3.2 Construindo a interface de usuário

É na pasta /res/*layout* que, por padrão, ficam localizados os arquivos de definição de interface de usuário de cada tela da aplicação. Em nosso projeto de exemplo, o Android Studio criou o arquivo primeiro_aplicativo.xml. Ao acessar este arquivo na IDE (dois cliques no arquivo) são apresentadas duas visões: *design* e *Text* (abas localizadas no canto inferior esquerdo do Android Studio). Na visão *Text*, é carregado um editor de texto de código fonte (figura abaixo) junto ao *preview* da tela do aplicativo.

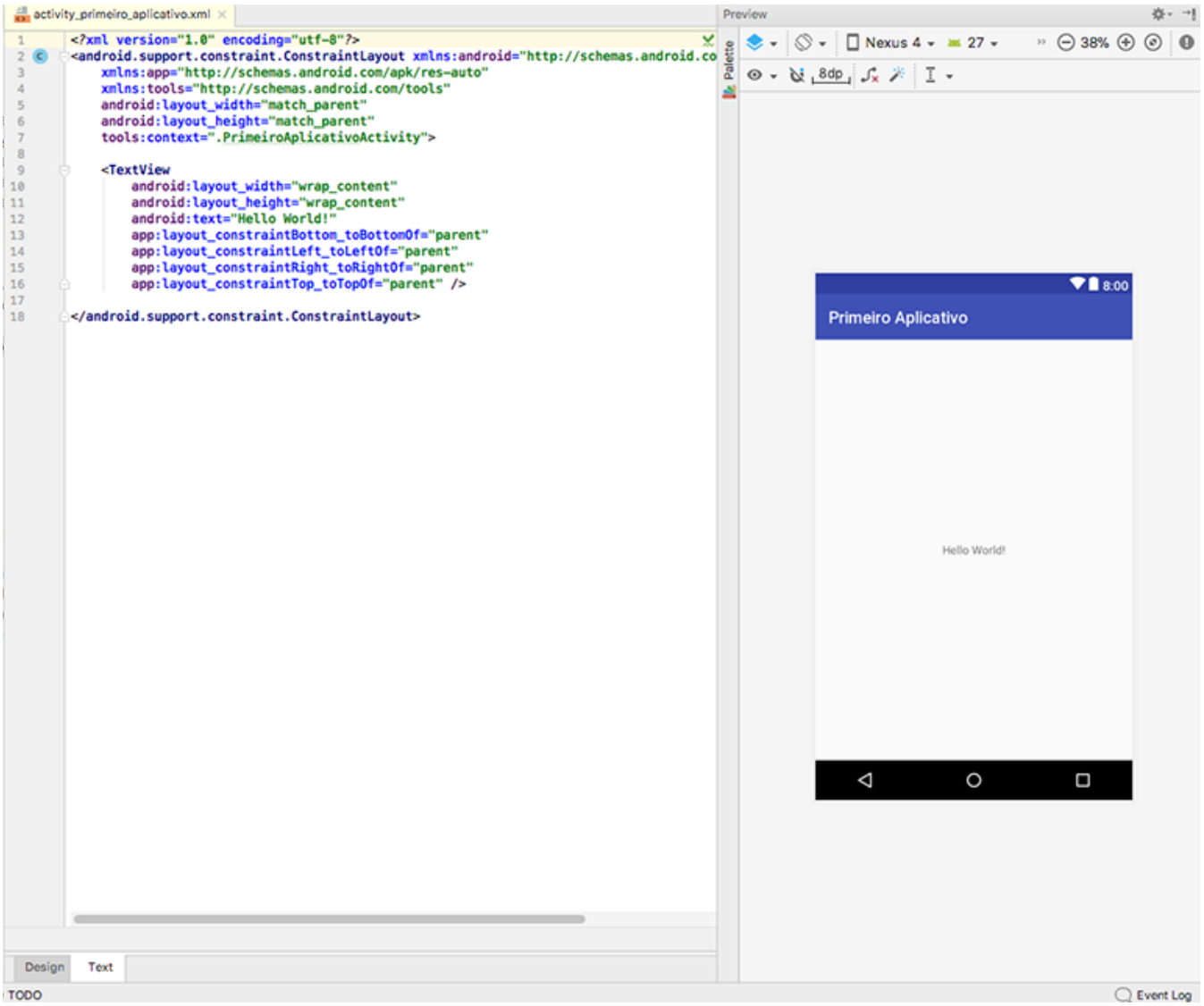


Figura 11 - Ferramenta de construção de interface de usuário. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Ferramenta de construção de interface de usuário. No projeto de

exemplo, o Android Studio criou o arquivo primeiro_aplicativo.xml. Ao acessar este arquivo na IDE (dois cliques no arquivo) são apresentadas duas visões: design e Text (abas localizadas no canto inferior esquerdo do Android Studio). Na visão Text, é carregado um editor de texto de código fonte junto ao preview da tela do aplicativo.

Já na visão *design*, são carregadas quatro ferramentas para montar o *layout* da tela visualmente (GOOGLE, 2018):

- *pallete* (paletas): contém uma lista de componentes de interface de usuário, como botões, caixas de texto, *layout*s, containers, etc. Cada componente pode ser facilmente arrastado para a área de *preview*, automaticamente o código fonte da tela será atualizado com o componente adicionado;
- *component tree* (árvores de componentes): exibe a hierarquia da interface de usuário construída;
- *preview* (previsão do aplicativo): uma previsão de como será exibido em um dispositivo, a interface de usuário que está sendo construída, é visualizada em inúmeras resoluções de tela e pode-se alternar entre as orientações *portrait* e *landscape*;
- *properties* (propriedades): exibe as propriedades de um elemento da árvore de componentes que podem ser alteradas.

Toda interface de usuário é definida pela combinação de componentes *View* e *Viewgroup*.

Os componentes classificados como *View* são componentes básicos para criação de interface de usuário. Os componentes *View* são listados na ferramenta de paleta na visão *design* do Android Studio, mencionada anteriormente. A plataforma Android possui como base componentes de *View* especiais, que são classificados como *widgets*.

Os *widgets* mais comuns em interface de usuário são: *TextView* (componente para exibir texto na tela), *ImageView* (componente para exibir imagem na tela), *Button* (componente para exibir um botão na tela), *EditText* (componente para exibir uma de caixa texto).

Além destes widgets, a plataforma Android ainda conta três componentes para organizar a disposição de dados na interface de usuário (GOOGLE, 2018):

- ***ListView***: organiza os dados passados em formato em uma lista vertical com rolagem, ou seja, o *listview* recebe dados através de um *array* e apresenta, na tela, os dados de cada elemento do *array* um embaixo do outro;
- ***GridView***: organiza os dados passados em formato de tabela com rolagem, ou seja, é possível configurar no componente como os dados de cada elemento do *array* passado serão dispostos na tela. É possível criar uma lista de elementos como no *listview* utilizando o *GridView* desde que o desenvolvedor o configure com uma coluna.
- ***RecyclerView***: é um novo widget surgido na API 21 do Android que vem substituindo tanto o *GridView* e *ListView* nas interfaces de usuário, visto que tem a flexibilidade de apresentar dados tanto em lista quanto em formato tabela, possui melhor performance e menor consumo de memória que os dois componentes mais antigos.

Os elementos *View* precisam ser agrupados para serem exibidos na tela. A plataforma Android fornece componentes *ViewGroup* para esta finalidade. Existem inúmeros componentes *ViewGroup* (também listados na ferramenta de paleta na visão *design* do Android studio), os principais componentes *ViewGroup* são (GOOGLE, 2018):

- *Linearlayout*: alinha componentes *View* ou *ViewGroup* em formato de linha ou de coluna na tela;
- *Gridlayout*: alinha componentes *View* ou *ViewGroup* em formato de tabela na tela;
- *Relativelayout*: mais complexo e mais poderoso que os *ViewGroup* mencionados anteriormente. Este componente alinha componentes *View* ou *ViewGroup* de acordo com referência a outros componentes posicionados na tela;
- *Constraintlayout*: é considerado uma evolução ao componente *Relativelayout*. Pois permite alinhar componentes *View* pelo posicionamento relativo, mas é mais flexível e fácil de utilizar que o

Relativelayout.

Sendo assim, a interface de usuário de um aplicativo na plataforma Android é formada por uma hierarquia de componentes *View* e *ViewGroup*. Na figura a seguir, é apresentado um exemplo de organização destes dois componentes.

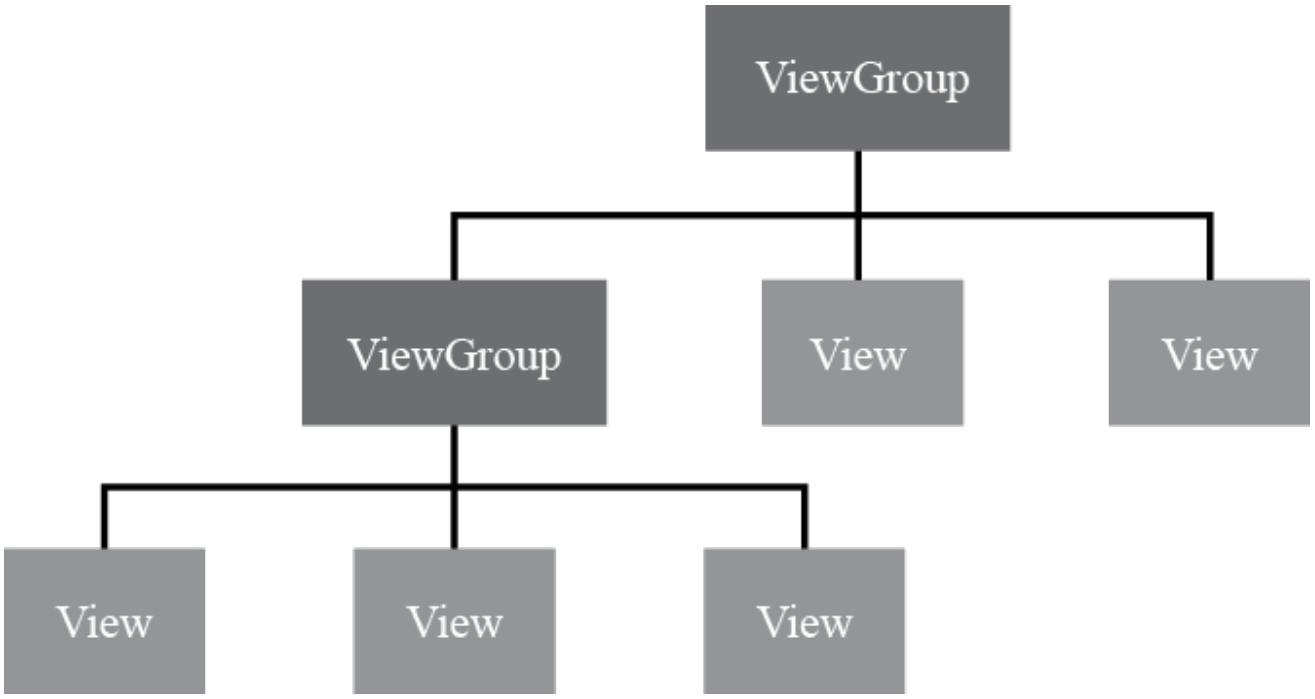


Figura 12 - Exemplo de Hierarquia de uma tela. Fonte: Google (2018).

#PraCegoVer: Gráfico com retângulos hierárquicos, no qual os retângulos distribuídos verticalmente e ligados por linhas representam a comunicação e a hierarquia dos itens. O ViewGroup ocupa o primeiro nível do organograma (primeira hierarquia). No segundo nível (segunda hierarquia) partindo do retângulo do ViewGroup, sai uma linha que será dividida para se ligar a um ViewGroup e a dois View. No terceiro nível do ViewGroup sai uma linha que será dividida para se ligar a 3 view.

Enfim, existem duas maneiras de construir a interface de usuário de uma aplicação e elas se complementam. Ao editar o modelo de interface na aba *design*, o código é alterado imediatamente e, ao atualizar o código XML na aba *Text*, o modelo é atualizado, de acordo com o que foi alterado.

1.3.3 Gerando e executando a aplicação

Vamos verificar se está tudo correto com o projeto, compilando um apk e instalando em um dispositivo. Acesse a opção *Run* (Executar) na barra de tarefas do Android Studio, selecionando o primeiro item (*run app*) que compila e executa o aplicativo desenvolvido.

Após o processo de compilação ocorrer com sucesso, o Android Studio solicitará a escolha de um destino de implantação: um dispositivo móvel real ou um emulador de dispositivo Android, conhecido como AVD (Android *Virtual Device*). Para a execução da aplicação em um dispositivo real é necessário:

- configurar o dispositivo para permitir depuração USB. O passo a passo de como realizar esta configuração está disponível em <https://developer.Android.com/studio/run/device?hl=pt-br (https://developer.Android.com/studio/run/device?hl=pt-br)>
- instalar *driver* USB de acordo com a fabricante do dispositivo. Todos os passos para identificação e instalação do *driver* adequado é acessível na página <https://developer.Android.com/studio/run/oem-usb?hl=pt-br (https://developer.android.com/studio/run/oem-usb?hl=pt-br)>.

VOCÊ SABIA?

Uma das mais importantes ferramentas disponíveis no SDK é o Android *Virtual Device Manager* (AVD Manager). Usando o AVD Manager, é possível criar dispositivos móveis emulados (AVD) com as mais variadas características (CPU, memória, resolução de tela, armazenamento, etc.) no ambiente de desenvolvimento. A documentação explicando como funciona o AVD *Manager* pode ser acessada em <https://developer.Android.com/studio/run/managing-avds?hl=pt-BR (https://developer.Android.com/studio/run/managing-avds?hl=pt-BR)>.

O AVD Manager é uma ótima ferramenta para incrementar a qualidade da aplicação. Pela ferramenta é possível emular o funcionamento da aplicação nos mais variados tamanhos de tela e versões Android.

1.3.4 Monitorando a aplicação em tempo de execução

Pelo Android Studio é simples monitorar a execução das aplicações Android

durante o processo de desenvolvimento, pela ferramenta Logcat. Esta ferramenta, por padrão, é localizada numa das abas do lado inferior esquerdo do Android Studio.

Na barra de ferramentas da aba *logcat* (figura abaixo), é possível configurar qual dispositivo a ser monitorado (no caso de estar rodando a aplicação em mais de um dispositivo), quais tipos de mensagem devem ser exibidas (todas, *info*, *error*, etc.), quais aplicações devem ser monitoradas.

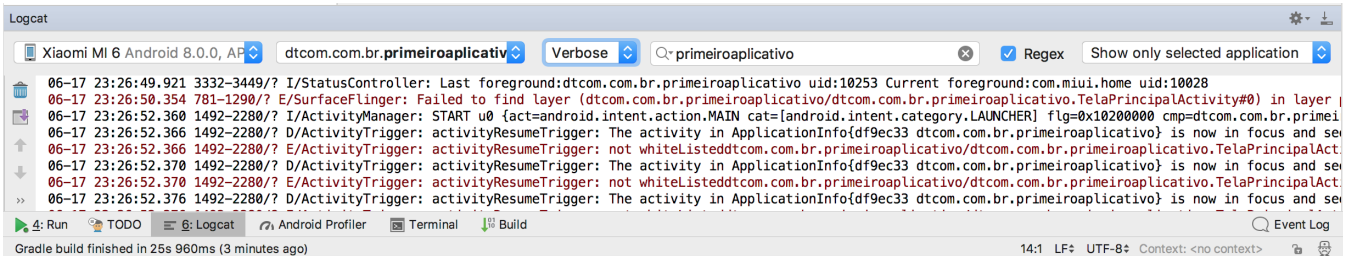


Figura 13 - Monitorando a aplicação PrimeiroAplicativo. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela de execução do Logcat, campos para serem selecionados: primeiro campo: Xiaomi MI 6 Android 8.0.0, segundo campo: dtcom.com.br.primeiroaplicativo, terceiro campo: Verbose, quarto campo: primeiroaplicativo, quarto campo: Regex, quinto campo: Show only select application.

O logcat é uma ferramenta essencial no desenvolvimento de qualquer aplicação Android nativa, visto que permite depurar a aplicação, permitindo que o desenvolvedor encontre as causas de defeitos (*bugs*), por meio das mensagens emitidas pela aplicação no console do logcat.

Finalmente, aprendemos a executar e monitorar nossa primeira aplicação Android! Por enquanto, ainda não escrevemos nenhuma linha de código Java. Entraremos em mais detalhes no próximo tópico.

1.4 Fundamentos do desenvolvimento Android

Neste tópico, são apresentados os conceitos básicos *necessários* para a construção

de um aplicativo Android. Para auxiliar neste entendimento, é utilizado o caso do projeto PrimeiroAplicativo, criado de forma automatizada no tópico anterior. Entenderemos os componentes básicos envolvidos em uma aplicação Android e como eles se relacionam por meio da análise de código fonte e arquivo de configuração gerados.

1.4.1 Intents

O Android possui um mecanismo de troca de mensagens conhecido como *Intents*. As *Intents* são mensagens assíncronas que são usadas para comunicação entre os diversos processos rodando no sistema operacional. Ou seja, os diversos aplicativos executados na plataforma Android, podem se comunicar e trocar dados com o sistema operacional por meio das *Intents*.

VOCÊ QUER LER?

Para obter mais informações sobre este mecanismo de troca de mensagens na plataforma Android denominada *Intents*, são apresentadas na página "*Intents* e Filtros de *Intents*" (GOOGLE, 2018).

Voltando ao projeto PrimeiroAplicativo de exemplo criado pelo Android Studio na seção passada, a classe TelaPrincipalActivity é definida como *Activity* principal do código destacado na figura anterior (Monitorando a aplicação PrimeiroAplicativo), do AndroidManifest.xml:

```
<activity android:name=".TelaPrincipalActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 14 - Definição da Activity principal. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela com as linhas de comando:

```
<activity android:name=“.TelaPrincipalActivity”>
<intent-filter>
<action android:name=“android.intent.action.MAIN” />
<category android:name=“ android.intent.category.LAUNCHER” />
<intent-filter>
<activity>
```

No trecho de código da figura acima, estamos registrando uma *Activity* com o nome TelaPrincipalActvity. Perceba que temos um elemento neste arquivo chamado de ***Intent*-filter**.

Pelos parâmetros **Android.Intent.category.LAUNCHER** e a ação **Android.Intent.action. MAIN**, definimos qual *Activity* será a primeira a ser carregada ao iniciar o aplicativo.

Logo, no trecho XML destacado, a função da *tag* Intent-filter é definir a TelaPrincipalActivity como *Activity* principal e primeira a ser carregada.

1.4.2 Ciclo de Vida das atividades (instâncias de classes Activity*)*

Diferentemente de outros paradigmas de programação em que os aplicativos são iniciados com um método “*main*”, os aplicativos Android iniciam o código em uma instância de uma classe *Activity* (definida como *Activity* principal no arquivo AndroidManifest.xml) chamando método “*onCreate*” que indica que a aplicação foi inicializada, ou seja, o *primeiro* método a ser executado de uma aplicação Android é o método *onCreate*() da *Activity* principal.

Uma *Activity* pode adquirir os seguintes estados (DEITEL; DEITEL; WALD, 2016):

- **ativa**: uma atividade ativa é visível na tela e “tem o foco”, isto é, está no primeiro plano. O usuário pode interagir com a atividade que está no primeiro plano;
- **pausada**: uma atividade pausada é visível na tela, mas não tem o foco – como quando uma caixa de diálogo de alerta é exibida. Não é possível interagir com a atividade pausada até que ela se torne ativa;
- **parada**: uma atividade parada não é visível na tela, ou seja, ela está em segundo plano.

Para cada mudança de estado da tela, existe um método da classe *Activity* que trata esta mudança. Os métodos são (GOOGLE,2018):

- onCreate: como já foi mencionado, é acionado apenas quando a *Activity* é criada;
- onStart: este método é executado logo após uma *Activity* ter sido criada ou após ela ter sido reiniciada;
- onResume: o método é executado acionado quando a *Activity* volta a ter foco;
- onPause: o método é executado acionado quando a *Activity* perde o foco, ou seja, a *Activity* será substituída por outra (pausada), ou existe um componente de interface assumindo o foco (mensagem de diálogo);
- onStop: o método é executado quando a *Activity* deixa de ser exibida porque outra *Activity* assumiu o foco;
- onDestroy: o método é executado sempre que a *Activity* esteja sendo destruída, ou seja, após a execução deste método a plataforma limpará da memória a *Activity.*

O diagrama da figura abaixo ilustra como estão interligados estes estados.

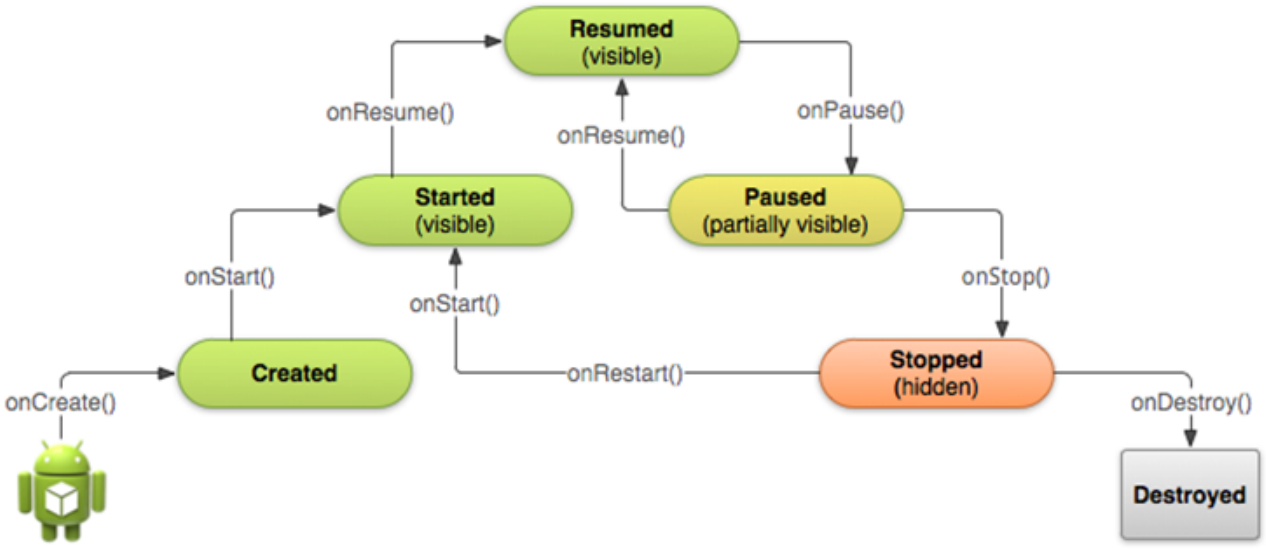


Figura 15 - Ciclo de Vida das classes Activity. Fonte: GOOGLE (2018).

#PraCegoVer: O esquema mostra o ciclo de vida em forma de degraus, subindo

até atingir o topo e então começa a descer os degraus. Começa com um boneco verde (símbolo do Android) que se conecta ao próximo passo (Created) através do onCreate(), este se conecta com o Started (Visible) através do onStart(), que se conecta com o Resumed (Visible) através do onResume(), aqui estamos no topo e os degraus começam a descer, sendo o primeiro degrau Paused (partially visibe) conectado pelo onPause() neste step pode-se voltar para o degrau anterior Resumed (Visible) através do onResume(), ou descer mais um degrau Stopped (hidden) atráves do onStop() e o último estágio do ciclo – terminando os degraus temos o Destroyed, através do onDestroy().

Desta forma, a aplicação do projeto PrimeiroAplicativo, ao iniciar, deverá executar o seguinte método definido na classe TelaPrincipalActivity destacado na figura abaixo.

```
package dtcom.com.br.primeiroaplicativo;

import ...

public class TelaPrincipalActivity extends AppCompatActivity {

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

Figura 16 - Implementação da classe TelaPrincipalActivity. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela com as linhas de comando:

package dtcom.com.br.primeiroaplicativo;
import ...
public class TelaPrincipalActivity extends AppCompatActivity {
@Override
protected void onCreate (Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

```
setContentView(R.layout.activity_tela_principal);
}
}
```

O método onCreate é acionado pelo sistema Android que passa um argumento Bundle contendo o estado salvo da atividade, ou seja, dados presentes na *Activity*. A classe Bundle é uma classe utilitária para armazenar ou transportar dados entre o sistema Android e as classes *Activity*. Utilizaremos bastante a classe Bundle em nossas aplicações para transmitir dados entre instâncias *Activity*.

VOCÊ QUER LER?

A principal função deste argumento Bundle no método onCreate, é permitir que a *Activity* seja recriada com o estado que ela tinha antes de ser destruída.

Utilizando o argumento *savedInstanceState*, é chamado o método *onCreate* da classe pai que é obrigatório em todas classes *Activity* que sobrescrevem este método.

A chamada de método *setContentView* apresentada é responsável por carregar a *Activity* de acordo com interface de usuário definido no arquivo tela_principal.xml. É importante atentar para a classe “R”. Cada arquivo na pasta res contém uma referência na classe R, que é automaticamente gerada pelo Android Studio.

O arquivo "R.java" não deve ser modificado manualmente, mas é de suma importância conhecê-lo. Pelas constantes, criadas automaticamente na classe R, é que é feita a ligação entre os arquivos *resources* (*layouts*, *drawables*, etc.) com o código Java.

1.4.3 Modelo de Permissões

Com a finalidade de propiciar privacidade aos usuários de aplicativos Android, uma aplicação desta plataforma deve requisitar acesso explícito aos dados do usuário (lista de contatos, SMS, *e-mail*, etc.), ou recursos do sistema operacional (acesso à rede, câmera, etc.). As permissões do sistema são divididas em duas

categorias (GOOGLE, 2018):

- **permissões normais**: não representam um risco direto à privacidade do usuário. Um aplicativo que lista uma permissão normal em seu *manifest*, o sistema a concede automaticamente;
- **permissões perigosas**: podem conceder ao aplicativo acesso aos dados confidenciais do usuário. Se seu aplicativo lista uma permissão normal em seu *manifest*, o sistema a concede automaticamente. Se você lista uma permissão perigosa, o usuário precisa aprovar seu aplicativo explicitamente.

O desenvolvedor deve requisitar permissões no AndroidManifest.xml utilizando o elemento <uses-permission>. Por exemplo, no projeto PrimeiroAplicativo, caso o aplicativo necessite de acesso ao cartão de memória do dispositivo, devemos adicionar o elemento <uses-permission> antes do elemento <application>. Podemos ver como fica o arquivo de *manifest*:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="dtcom.com.br.primeiroaplicativo" >
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="PrimeiroAplicativo"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".TelaPrincipalActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 17 - Configurando requisição de permissão para leitura no cartão de memória. Fonte: elaborado pelo autor, 2018.

#PraCegoVer: Linhas de comando com o elemento <uses-permission> antes do elemento <application>

```
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
package=”dtcom.com.br.primeiroaplicativo”>
<uses-permission
android:name=”android.permission.READ_EXTERNAL_STORAGE”></uses-
permission>
<application
android:icon=”@mipmap/ic_launcher”
android:label=”PrimeiroAplicativo”
android:roundIcon=”@mipmap/ic_launcher_round”
android:supportsRtl=”true”
android:theme=”@style/AppTheme”>
<activity android:name=”. TelaPrincipalActivity”>
<intente-filter>
<action android:name=”android.intent.action.MAIN” />
<category android:name=”android.intent.category.LAUNCHER” />
</intente-filter>
</activity>
</application>
</manifest>
```

A partir da API 23 (Android *Marshmallow*), a plataforma Android solicita permissão a um recurso em tempo de execução, ou seja, no momento que o usuário for utilizar pela primeira vez um determinado recurso. Assim, para aplicações com versões recentes do Android (versões posteriores a API 23), além de configurar a requisição de permissão no arquivo AndroidManifest.xml (como mostrado na figura anterior). Deve-se utilizar o método Java estático *requestPermission* da classe *ActivityCompat*, sempre que for verificado que o aplicativo não possui permissão para um determinado recurso.

VOCÊ QUER LER?

Mais detalhes sobre o modelo de permissões do Android e mais exemplos de requisições de permissão ao sistema Android, podem ser encontrados na página “Permissões do sistema” (GOOGLE, 2018), disponível em <https://developer.android.com/guide/topics/security/permissions (https://developer.android.com/guide/topics/security/permissions)>.

Enfim, compreendemos os elementos fundamentais de uma aplicação Android. Estes são elementos muito importantes e que estarão presentes na maioria das suas futuras aplicações.

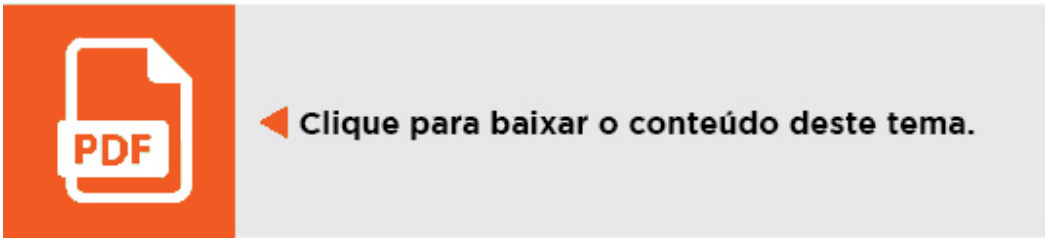
Síntese

Chegamos ao final do capítulo e já podemos criar e executar uma aplicação móvel. Vimos que o universo de dispositivos móveis é vasto e existem inúmeras possibilidades plataformas, ferramentas, técnicas, boas práticas e ferramentas envolvidas na criação de aplicativos móveis. Este capítulo, então, apresentou fundamentos práticos da programação para dispositivos móveis pela plataforma Android, utilizando as ferramentas oficiais de desenvolvimento.

Neste capítulo, você teve a oportunidade de:

- listar as principais características dos dispositivos móveis e quais são os principais desafios ao desenvolver aplicações neste universo;
- acompanhar o processo de evolução dos dispositivos móveis, com foco nos *smartphones*;
- identificar as principais plataformas móveis, presentes no mercado, e quais as suas principais características;
- refletir sobre as abordagens de desenvolvimento utilizadas para a implementação de aplicações móveis
- analisar os elementos de uma arquitetura de solução móvel;
- projetar a interface de usuário, usando padrões e princípios utilizados no mercado;
- compreender como expor um projeto de aplicativo sem necessidade de codificá-lo, pela técnica de prototipação;
- iniciar e executar uma implementação de um aplicativo móvel Android;
- compreender a estrutura de diretórios de aplicativos Android;
- entender a implementação de interface de usuário no Android Studio;
- compreender conceitos básicos do desenvolvimento de aplicativos,

como ciclo de vida das telas da aplicação baseadas em *Activity*, *Intent* (utilizada para troca de mensagens entre processos) e requisição de permissões.



Bibliografia

DEITEL, P. J.; DEITEL, H.; WALD, A. **Android 6 para Programadores**: uma abordagem baseada em aplicativos. 2. ed. Porto Alegre: Bookman, 2016. Disponível na Biblioteca Virtual Ânima: <https://Ânimabrasil.blackboard.com /webapps/blackboard/content/listContent.jsp?course_id=_198689_1& content_id=_4122211_1&mode=reset (https://Ânimabrasil.blackboard.com /webapps/blackboard/content/listContent.jsp?course_id=_198689_1& content_id=_4122211_1&mode=reset)>. Acesso em: 26/6/2018.

DIAKOPOULOS, D.; CASS, S. **Interactive**: the top programming languages. Portal IEEE Spectrum, publicado em 24 jul. 2017. Disponível em: <https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017 (https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017)>. Acesso em: 26/6/2018.

FONTÃO, A. L.; *et al*. Apps *frameworks*: Um processo centrado no usuário aplicado para o desenvolvimento de aplicativos móveis – um estudo de caso. In: **IADIS International Conference**, 2012, v. 1. p. 91-98. Disponível em: <https://www.researchgate.net/profile/Awdren_Fontao/publication /280041395_APPS_ (https://www.researchgate.net/profile/Awdren_Fontao /publication /280041395_APPS_framework_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links/55a50fea08aef604aa041cdb/APPS-framework-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf)*framework*

*(https://www.researchgate.net/profile/Awdren_Fontao/publication /280041395_APPS_framework_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links /55a50fea08aef604aa041cdb/APPS-framework-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf)*_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links /55a50fea08aef604aa041cdb/APPS- (https://www.researchgate.net/profile /Awdren_Fontao/publication /280041395_APPS_framework_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links/55a50fea08aef604aa041cdb/APPS-framework-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf)*framework (https://www.researchgate.net/profile/Awdren_Fontao/publication /280041395_APPS_framework_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links /55a50fea08aef604aa041cdb/APPS-framework-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf)*-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf (https://www.researchgate.net/profile/Awdren_Fontao/publication /280041395_APPS_framework_UM_PROCESSO_CENTRADO_NO_USUARIO_APLICADO_PARA_O_DESENVOLVIMENTO_DE_APLICATIVOS_MOVEIS_-_UM_ESTUDO_DE_CASO/links/55a50fea08aef604aa041cdb/APPS-framework-UM-PROCESSO-CENTRADO-NO-USUARIO-APLICADO-PARA-O-DESENVOLVIMENTO-DE-APLICATIVOS-MOVEIS-UM-ESTUDO-DE-CASO.pdf)>. Acesso em 26/06/2018.

FONSECA, M.; *et al*. **Introdução ao *design* de interfaces**. Lisboa: FCA, 2012

FLING, Brian. **Mobile *design* and development**. Sebastopol (CA): O'Reilly Media, 2009.

GARTNER. Gartner says worldwide sales of *smartphones* recorded first ever decline during the fourth quarter of 2017. **Portal Gartner**, Egham, UK, February 22, 2018. Disponível em < (https://www.gartner.com/newsroom/id/3859963)https: //www.gartner.com/newsroom/id/3859963 (https://www.gartner.com/newsroom

/id/3859963)>. Acesso em: 26/6/2018.

GOOGLE. **Android para desenvolvedores**. Portal Developers, 2018. Disponível em: < https://developer.android.com/guide/?hl=pt-br (https://developer.android.com/guide/?hl=pt-br)>. Acesso em: 26/6/2018.

IBGE. **Acesso à internet e a televisão e posse de telefone móvel celular para uso pessoal**. Instituto Brasileiro de Geografia e Estatística – IBGE. Rio de Janeiro: IBGE, 2016. Disponível em: < https://biblioteca.ibge.gov.br/visualizacao/livros/liv99054.pdf (https://biblioteca.ibge.gov.br/visualizacao/livros/liv99054.pdf)>. Acesso em: 26/6/2018.

KLEINA, N. **A história do Android – TecMundo**. Canal TecMundo, YouTube, publicado em 29 de agosto de 2017. Disponível em: < https://www.youtube.com/watch?v=5K4pEk19nhs (https://www.youtube.com/watch?v=5K4pEk19nhs)>. Acesso em: 28/06/2018.

LEE, V.; *et al*. **Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento**. Editora Pearson. 2005.

OWEN, M. **iOS 11 now installed on 76% of compatible devices, up 11% since January, 2018**. Disponível em <https://appleinsider.com/articles/18/04/25/iOS-11-now-installed-on-76-of-compatible-devices-up-11-since-january (https://appleinsider.com/articles/18/04/25/iOS-11-now-installed-on-76-of-compatible-devices-up-11-since-january)>. Acesso em: 26/06/2018.

SANTOS, L. D. J. **Desenvolvimento de soluções mobile para o banco de Portugal**. Repositório da Universidade de Lisboa Faculdade de Ciências (FC) Departamento de Informática / Department of Informatics (FC-DI) FC-DI - Master Thesis (projects). Lisboa: Universidade de Lisboa, 2014.

SOMMERVILLE, I. **Engenharia de *software***. Tradução: Kalinka Oliveira; Ivan Bosnic. 9. ed. São Paulo: Person Prentice Hall, 2011.