UNIVERSIDADE DO VALE DO TAQUARI – UNIVATES
CURSO DE ENGENHARIA DA COMPUTAÇÃO

APP BUNDLE E DYNAMIC DELIVERY: MODULARIZAÇÃO DE APPS PARA ANDROID

Leo Ely

Lajeado, dezembro de 2019

Leo Ely

APP BUNDLE E DYNAMIC DELIVERY: MODULARIZAÇÃO DE APPS PARA ANDROID

Monografia apresentada no Centro de Ciências Exatas e Tecnológicas, da Universidade do Vale do Taquari – Univates, como parte dos requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Prof. Me. Fabrício Pretto

Lajeado, dezembro de 2019

AGRADECIMENTOS

Um agradecimento especial ao meu orientador Fabrício Pretto, por todo o auxílio e pelas ideias apresentadas para a construção do presente trabalho, e pelo acompanhamento nas inúmeras disciplinas cursadas durante a graduação. Ainda, um agradecimento a todos meus professores e professoras, que proporcionaram inúmeros ensinamentos e contribuíram na evolução de meus conhecimentos.

Agradeço também aos meus familiares, que sempre me apoiaram no decorrer deste projeto e me estimularam a seguir em frente, sem desistir dos meus planos e dos meus ideais.

Um agradecimento final a todos os colegas pela troca de experiências e pelos momentos compartilhados durante todo o período da graduação, especialmente Alexandre Eger Marques, João Pedro Becchi e Raul Silvério Mittelstädt.

RESUMO

Aplicativos móveis são cada vez mais importantes na sociedade. Com a evolução tecnológica e o surgimento de smartphones, tablets e outros dispositivos com inúmeros novos recursos, os sistemas operacionais portáteis impulsionaram o desenvolvimento de aplicações para os mais diversos fins. Paralelamente, os processos de desenvolvimento evoluíram consideravelmente, devido aos desafios originados com os avanços da tecnologia. Decorrente de tal progresso, originaram-se problemáticas como o controle do tamanho dos aplicativos, a criação de novos meios de distribuição de apps e a viabilidade de modularização dos mesmos. Sendo assim, o presente trabalho realiza um estudo na documentação do novo formato de envio e desenvolvimento de aplicações para o sistema operacional Android, enfatizando a modularização dos recursos. Além disso, são apresentados os processos de criação de uma aplicação para o referido sistema operacional, utilizando os recursos de modularização. Testes de recursos técnicos foram realizados, verificando dados como o tamanho do aplicativo e dos módulos e o gerenciamento da entrega dos mesmos. Por fim, apresentam-se as conclusões baseadas nos resultados obtidos, as quais exibem os prós e contras da tecnologia examinada pelo presente trabalho.

Palavras-chave: Aplicação móvel. Android. Modularização. Tamanhos de apps.

ABSTRACT

Mobile apps are increasingly important in society. With the technologic evolution and the emergence of smartphones, tablets and other devices with countless new features, portable operating systems boosted the development of applications for the most diverse purposes. In parallel, development processes have evolved considerably due to the challenges originated from the advances in technology. Due to such progress, problems such as control of application sizes, creation of new means of distributing apps and the feasibility of modularizing them were originated. Thus, the present work carries out a study in the documentation of the new format for sending and developing applications for the Android operating system, emphasizing the resources' modularization. In addition, the processes for creating an application for said operating system is presented, using the modularization features. Technical resource tests were performed, verifying data such as application and modules' size and their delivery management. Finally, the conclusions based on the results obtained are presented, which exhibits the pros and cons of the technology examined by the present work.

Keywords: Mobile application. Android. Modularization. App sizes.

LISTA DE FIGURAS

Figura 1 – relações entre codinome, versão de sistema e nível de API 20

Figura 2 – exemplos de atividades disponíveis para uma aplicação 23

Figura 3 – diagrama de relações de um provedor de conteúdo 24

Figura 4 – estrutura de um arquivo de manifesto 26

Figura 5 – árvore de dependência de um APK dividido 28

Figura 6 – conteúdo de um Android App Bundle, com um módulo base e dois módulos de recursos dinâmicos 29

Figura 7 – estrutura de diretórios, pastas e arquivos de um projeto 31

Figura 8 – aplicativo publicado na Play Store 40

Figura 9 – telas principais da aplicação 41

Figura 10 – quantidade de armazenamento utilizada pelo app 42

Figura 11 – log de erro do módulo de recurso dinâmico 43

Figura 12 – diagrama ER da aplicação 45

Figura 13 – tela de configuração de novo projeto 47

Figura 14 – criação do módulo de recurso dinâmico 48

Figura 15 – configuração do módulo de recurso dinâmico 49

Figura 16 – configurações finais do módulo 50

Figura 17 – tela principal do aplicativo 51
Figura 18 – telas dos módulos do aplicativo 52
Figura 19 – método de instalação de um módulo.....53
Figura 20 – classe para verificação de status das ações 54
Figura 21 – classe para leitura de códigos QR.....56
Figura 22 – tela de criação da chave de upload.....57
Figura 23 – seleção do keystore para criação do arquivo assinado.....58

LISTA DE ABREVIATURAS E SIGLAS

AOT	Ahead-of-time
API	Application Programming Interface – Interface de programação de aplicações
APK	Android Package Kit
ART	Android Runtime
CTS	Compatibility Test Suite – Suíte de teste de compatibilidade
ER	Entidade-Relacionamento
ERP	Enterprise Resource Planner – Sistema Integrado de Gestão Empresarial
HAXM	Hardware Accelerated Execution Manager
HTTP	Hypertext Transfer Protocol – Protocolo de Transferência de Hipertexto
IDE	Integrated Development Environment – Ambiente integrado de desenvolvimento
IL2CPP	Intermediate Language to C++ – Linguagem Intermediária para C++
JDK	Java Development Kit – Kit de desenvolvimento Java
JSON	JavaScript Object Notation – Notação de Objetos JavaScript
ML	Machine Learning – Aprendizado de Máquina
NDK	Native Development Kit – Kit de desenvolvimento nativo
OHA	Open Handset Alliance
REST	Representational State Transfer – Transferência Representacional de Estado
SD	Secure Digital

SDK	Software Development Kit – Kit de desenvolvimento de software
SQL	Structured Query Language – Linguagem de Consulta Estruturada
UI	User Interface – Interface de Usuário
URI	Uniform Resource Identifier – Identificador de Recurso Uniforme
URL	Uniform Resource Locator – Localizador Uniforme de Recursos
VM	Virtual Machine – Máquina virtual
XML	Extensible Markup Language – Linguagem de marcação extensível

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Tema 13
1.1.1 Delimitação do tema 13
1.2 Problema 13
1.3 Objetivos 13
1.3.1 Objetivo geral 13
1.3.2 Objetivos específicos 14
1.4 Estrutura do trabalho 14

2 REFERENCIAL TEÓRICO 15
2.1 Computação móvel 15
2.1.1 O surgimento do Android 16
2.1.2 As bases do Android 16
2.1.3 Características do sistema e dos aplicativos 17
2.1.4 A distribuição de aplicativos 18
2.2 O desenvolvimento de aplicações 19
2.2.1 Seleção de plataforma 19
2.2.2 Um panorama da compatibilidade 20
2.3 Fundamentos de apps 21
2.3.1 Componentes 22
2.3.2 Intents e ContentResolver 25
2.3.3 Manifestos e recursos 25
2.4 Android App Bundle e Dynamic Delivery 27
2.4.1 Formato de um App Bundle 29
2.4.2 Solicitação e instalação de módulos 31
2.4.3 Propriedades de módulos dinâmicos 32

3 PROCEDIMENTOS METODOLÓGICOS 34
3.1 Métodos de pesquisa 34

3.2 Modo de abordagem 34
3.3 Objetivos da pesquisa 35
3.4 Procedimentos técnicos usados na pesquisa 35
3.5 Ferramentas e dispositivos 35

4 TRABALHOS RELACIONADOS 37

5 DESENVOLVIMENTO DA APLICAÇÃO 39
5.1 Proposta inicial 39
5.1.1 Validação da proposta 40
5.1.2 Resultados obtidos 41
5.2 Visão geral da nova proposta 44
5.2.1 Módulos do aplicativo 44
5.2.2 API REST 45
5.2.3 Aplicação Android 46
5.2.4 Classes utilizadas no aplicativo 53
5.2.5 Bibliotecas para API REST 55
5.2.6 APIs de acesso à câmera 55
5.2.7 Envio do app à Play Store 56
5.2.8 Resultados obtidos 58

6 CONSIDERAÇÕES FINAIS 61

REFERÊNCIAS BIBLIOGRÁFICAS 63

1 INTRODUÇÃO

Os aplicativos móveis tornaram-se cada vez mais importantes nos tempos atuais, sendo eles promovidos majoritariamente pelo advento do smartphone. Hoehle e Venkatesh (2015) ressaltam que na maioria dos países desenvolvidos, as pessoas frequentemente possuem mais de um aparelho móvel, e que os dispositivos e os sistemas operacionais estão se tornando cada vez mais refinados.

A história dos apps móveis começou com os primeiros telefones móveis, os quais necessitavam do software mais básico para enviar e receber chamadas de áudio, segundo Siuhi e Mwakalonge (2016). Basicamente, os primeiros apps eram editores de toques, calculadoras, calendários e jogos rudimentares, e no início dos anos 2000 os sistemas operacionais para smartphones (Windows Mobile, Symbian, Android e iOS) permitiram o desenvolvimento de softwares por terceiros, devido ao avanço tecnológico e à expansão do mercado. Por sua vez, cada fabricante introduziu e simplificou seus processos de desenvolvimento, permitindo aos usuários a personalização de seus aparelhos. Dessa forma, cada empresa poderia elaborar produtos atrativos, intuitivos e de fácil utilização, além de aumentar seus rendimentos.

Duas empresas foram responsáveis pela inovação do mercado onde estão inseridas: a Apple e a Google, conforme Novac et al. (2017). Ao lançar o iPhone, a Apple estabeleceu um novo paradigma, onde os conceitos-chave são interfaces de usuário controláveis e lojas de aplicativos. Em seguida, a Google deu continuidade

ao exemplo e lançou o Android, alterando permanentemente a visão do que se considera um sistema operacional.

Além de ser um sistema operacional, o Android também é considerado uma plataforma (NOVAC et al., 2017). Ele é projetado principalmente para telas sensíveis ao toque, bem como para televisores (Android TV), carros (Android Auto) e relógios (Wear OS), sendo capaz de transformar qualquer dispositivo móvel em um computador portátil. Suplementar a isso, o Google Play é um serviço de lojas online que oferece músicas, filmes, livros, aplicativos e jogos ao Android.

Por ser um software de código aberto, os utilizadores do Android podem receber benefícios por meio de desenvolvedores da Google ou por terceiros, os quais são capazes de implementar melhorias ao sistema. Ainda, com sua adoção de mercado em ritmo constante e o surgimento de novas tecnologias para desenvolvimento e difusão de aplicativos, o Android é considerado um dos sistemas mais evoluídos da atualidade.

Devido ao amplo ecossistema onde o Android está instalado, criar um aplicativo que opere desde o menor smartphone até o maior televisor é um problema, já que existem inúmeras arquiteturas de processadores, vários tamanhos e densidades de telas e inúmeros idiomas. Com isso, um app pode ter arquivos de grandes dimensões, fazendo com que ele se torne indisponível para uma série de dispositivos. Visando reduzir este inconveniente, desenvolvedores criam vários aplicativos com recursos específicos para cada plataforma, que são então entregues aos dispositivos.

Tal estratégia de desenvolvimento torna a construção de aplicações móveis complexa, e com isso em vista, foi criado o Android App Bundle. Ele se trata de um novo formato de envio de aplicações, onde os códigos e recursos do aplicativo são divididos em módulos e então enviados à Play Store, responsável por gerar aplicativos otimizados para cada dispositivo. Assim, desenvolvedores não precisam mais preocupar-se em criar apps específicos para determinadas plataformas (como diferentes versões de um mesmo app para televisores, smartphones e dispositivos vestíveis), o tamanho do app é diminuído e recursos podem ser adicionados sob demanda, por meio de módulos dinâmicos.

1.1 Tema

O presente projeto tem como foco realizar a exploração da modularização de conteúdos e funções de uma aplicação para dispositivos móveis na plataforma Android, de forma a compreender sua concepção, construção e execução relacionados ao processo de implementação, publicação e entrega.

1.1.1 Delimitação do tema

Analisar o funcionamento e a estruturação de um aplicativo, restringindo-se à criação de um app para Android que contenha módulos de recursos dinâmicos, por meio do Android App Bundle.

1.2 Problema

É eficiente criar um aplicativo para Android que seja modular, adicionando recursos a ele sob demanda?

1.3 Objetivos

Nas seções a seguir são apresentados os objetivos gerais e específicos do estudo.

1.3.1 Objetivo geral

Implementar uma aplicação utilizando a tecnologia Android App Bundle, verificando o funcionamento e os benefícios oferecidos pela mesma. Para fins exploratórios, um aplicativo para a plataforma Android foi desenvolvido com a utilização da nova tecnologia, contendo componentes modulares a serem instalados sob demanda por intermédio dos novos módulos de recursos dinâmicos.

1.3.2 Objetivos específicos

São designados como objetivos específicos:

- Compreender e documentar as tecnologias e os recursos envolvidos no processo App Bundle do Android;
- Desenvolver uma aplicação para Android que contenha módulos de recursos dinâmicos, os quais podem ser instalados sob demanda;
- Incorporar recursos de um sistema ERP (sistema integrado de gestão) aos módulos citados anteriormente;
- Validar o aplicativo desenvolvido via testes de recursos técnicos, verificando dados como o tamanho do aplicativo, dos módulos e o gerenciamento da entrega dos mesmos, apresentando posteriormente os resultados dessa avaliação.

1.4 Estrutura do trabalho

O trabalho está estruturado em 6 capítulos. O primeiro capítulo contém a introdução sobre a modularização de aplicativos para dispositivos móveis. O segundo capítulo compreende a revisão da literatura sobre as tecnologias de desenvolvimento de aplicações móveis. O terceiro capítulo detalha a metodologia de desenvolvimento e as ferramentas utilizadas para a realização das atividades apresentadas. O quarto capítulo apresenta os trabalhos relacionados ao presente projeto. Enquanto isso, o quinto capítulo toma a descrição do desenvolvimento da aplicação do aplicativo móvel. Por fim, o sexto capítulo apresenta as considerações finais do referido projeto.

2 REFERENCIAL TEÓRICO

Neste capítulo, é realizada uma abordagem dos conteúdos de computação móvel e modularização de aplicações, os quais servem de embasamento teórico acerca do tema proposto, além de situarem o leitor frente aos objetivos referidos no projeto.

2.1 Computação móvel

Segundo Nudelman (2013), a população está vivendo em uma era em que informações são capazes de alcançar qualquer um em questão de minutos, onde a computação móvel é considerada um divisor de águas no desenvolvimento da história humana. Para o autor, isso acontece devido à abrangência e capacidade da computação, que envolve as pessoas e dá a elas maior flexibilidade, liberdade e poder sobre o que são capazes de fazer. Além disso, Monteiro (2012) ressalta que os telefones celulares receberam inúmeros novos recursos ao longo dos anos e tornaram-se indispensáveis na vida cotidiana.

Complementar a isso, Nudelman (2013) diz que o surgimento dos smartphones deu origem à uma certa simbiose entre a tecnologia e o ser humano, alterando completamente a forma com que ocorre a interação entre homem e máquina. A tecnologia tornou-se um assistente digital intuitivo, quase como um órgão adicional. Lecheta (2016) adiciona também que atualmente nenhuma pessoa é capaz de manter-se afastada de um smartphone ou um tablet, sendo eles

basicamente indivisíveis da maioria dos indivíduos. Siuhi e Mwakalonge (2016) lembram ainda que os aplicativos móveis se tornaram uma das principais ferramentas na vida pessoal e profissional das pessoas, contribuindo em aplicações essenciais nas áreas de comunicação, educação, finanças, negócios, viagens e transporte, além das áreas médicas, utilitárias, sociais e de entretenimento.

2.1.1 O surgimento do Android

Conforme Lecheta (2016), os usuários tradicionais buscam no mercado tecnológico aparelhos com visual refinado, moderno e de fácil usabilidade, além da grande infinidade de aplicativos e recursos, enquanto as grandes corporações e os desenvolvedores buscam uma plataforma ágil e contemporânea para criar aplicativos. Fabricantes como Samsung, Sony, HTC, LG e Motorola necessitam de uma base robusta e rica em funções para criar seus produtos, e segundo o autor, nestes requisitos o Android se encaixa por ser o ideal em todos os casos citados.

Monteiro (2012) diz que o sistema operacional Android começou a ser desenvolvido em 2003 pela empresa Android Inc., que foi comprada pelo Google em 2005 e detém o desenvolvimento do sistema até hoje. Nudelman (2013) ressalta que o Android sofreu complicações em sua expansão, devido à fragmentação do mercado, à falta de foco e de padrões e às constantes atualizações; todavia, os problemas foram enfrentados e resolvidos, e o Android rapidamente evoluiu e aumentou sua fatia de mercado.

2.1.2 As bases do Android

Segundo Lecheta (2016), o sistema operacional do Android tem como base o *kernel* (núcleo) Linux, que é responsável por gerir os processos, as *threads*, a memória, a segurança dos arquivos e diretórios, além das redes e dos *drivers*. Adicional a isso, Monteiro (2012) ressalta que o Android é basicamente uma plataforma composta do sistema operacional, dos *middlewares* e do conjunto de aplicativos essenciais (como o próprio Telefone, Contatos e Navegador). Ademais, há também o Android SDK, que se trata de um conjunto de ferramentas e APIs

(interfaces de programação de aplicações) para o desenvolvimento de aplicativos, por meio da linguagem Java.

Lecheta (2016) salienta também que o Android é um projeto de código aberto, onde qualquer desenvolvedor é capaz de deixar sua contribuição no código-fonte, seja na correção de erros ou integração de novos recursos. Além disso, sua licença é gratuita, e os fabricantes não necessitam desembolsar qualquer quantia para usufruir da plataforma. Junto a isso, a Open Handset Alliance (OHA) trata-se de um grupo liderado pelo Google e formado por integrantes como Intel, HTC, LG, ASUS, Acer, Dell, Samsung, Sony, Toshiba, Huawei, entre outros nomes consolidados. Segundo o autor, o propósito do grupo é de firmar uma plataforma única e aberta para celulares, satisfazendo assim os consumidores com o produto, além de originar uma plataforma adaptável e próspera para aplicações corporativas.

2.1.3 Características do sistema e dos aplicativos

Segundo Monteiro (2012), a linguagem de programação Java atua como base para as aplicações do Android; entretanto, o sistema operacional não possui uma máquina virtual Java para executar tais aplicações, mas sim a Dalvik, que é aperfeiçoada para execução em equipamentos móveis. Quando se desenvolve uma aplicação, é utilizada a linguagem Java e seus recursos disponíveis, porém ao efetuar a compilação do *bytecode* ele é transformado para a extensão *dex*, que é um executável para a máquina Dalvik. Após a compilação, os executáveis e outros recursos da aplicação são condensados em um arquivo com a extensão *apk*, que está pronto para difusão e instalação. Enquanto isso, o sistema de construção (*build*) utilizado é o Gradle, e todo o processo de compilação e geração do arquivo é gerenciado pelo ambiente integrado de desenvolvimento (IDE).

Adicional a isso, a partir da versão 4.4 do Android desenvolveu-se a nova máquina virtual ART (Android Runtime), cuja ativação era opcional, conforme Lecheta (2016). Então, no lançamento da versão 5.0, o ART passou a ser a máquina virtual padrão, sucedendo a Dalvik. O ART possui diversos avanços, como a performance superior do *garbage collector*, evoluções no suporte ao debug de aplicativos e surgimento da compilação AOT (*ahead-of-time*), que otimiza o código

para melhorar o desempenho do aplicativo. O autor lembra ainda que ambos os consumidores e desenvolvedores não são afetados pela execução de uma ou outra máquina virtual; ainda assim, o desempenho do ART é superior, como afirma o Google.

Holzer e Ondrus (2011) explicam que o Google assumiu a ideologia de plataforma aberta ao lançar o Android, onde não existe somente um arquiteto responsável, mas um conjunto de desenvolvedores que são encorajados a se unir pelos provedores da plataforma. Com isso, os programadores possuem acesso a todos, ou pelo menos parte, do kit de desenvolvimento de software (SDK) e códigos-fonte do sistema operacional.

2.1.4 A distribuição de aplicativos

A disseminação das aplicações para o Android é normalmente feita pelo Google Play, inicialmente conhecido como Android Market, conforme Lecheta (2016). Tal ambiente foi criado para facilitar a distribuição de tais aplicações, bem como prover aos desenvolvedores um local universal para publicar suas criações. Conforme Android Developers (2009), para a publicação de aplicações, são realizadas duas tarefas: a preparação do app para lançamento, onde compila-se uma versão do aplicativo que pode ser instalada pelos utilizadores, e a liberação do app aos usuários, onde é realizada a divulgação, venda e distribuição da versão de lançamento do mesmo. Geralmente, as aplicações são lançadas via Google Play, porém a difusão inicial pode ser feita também por envios diretos aos usuários ou por um site.

Segundo Holzer e Ondrus (2011), possuir um único portal para publicação de aplicações permite ao provedor uma vantagem competitiva, já que oferece aos consumidores uma maneira fácil de localizar e efetuar download de aplicativos, além de beneficiar-se com os efeitos de *network* ao angariar cada vez mais novos usuários e desenvolvedores para a plataforma. SG (2016) ressalta que o Google primeiramente realiza uma revisão automatizada do app recebido, onde são feitas buscas por *malwares*, verificações de infração de diretrizes de conteúdo e reconhecimento de imagens (para averiguação de conteúdos sexualmente explícitos

ou imagens sob direitos autorais). Em seguida, o app é inspecionado manualmente, e a resposta da inspeção é entregue em menos de três horas, retornando um dos três seguintes avisos: publicação pendente (onde o aplicativo é enviado para revisão), publicado ou rejeitado.

2.2 O desenvolvimento de aplicações

De acordo com Lecheta (2016), o Android Studio é a IDE oficial para desenvolver aplicativos para o Android. Ela utiliza como base o IntelliJ IDEA, desenvolvido pela JetBrains, e contém o SDK com o emulador e todos os principais recursos para a criação de aplicações. Além disso, é necessário também possuir o JDK (kit de desenvolvimento Java) para que o ambiente de desenvolvimento funcione corretamente.

Lecheta (2016) salienta que o emulador do Android é conhecido por ser vagaroso, e que muitos programadores optam por elaborar e testar aplicativos diretamente em um aparelho físico, ou até mesmo instalam emuladores de terceiros devido à lentidão. Visando corrigir tal inconveniente, a Intel criou um acelerador para o emulador oficial, fazendo uso do Intel HAXM (gerenciador de execução de aceleração por hardware). Ele utiliza a tecnologia de virtualização presente nos processadores Intel para que o emulador execute instruções a aproximadamente 80% da velocidade nativa do processador.

2.2.1 Seleção de plataforma

Segundo Lecheta (2016), a existência de inúmeros dispositivos que executam o Android faz com que cada um deles possua uma versão diferente do sistema operacional. Dessa forma, torna-se necessário verificar o código identificador, conhecido como *API Level*, ao criar uma aplicação para o Android. Cada plataforma possui um identificador, e ele determinará a compatibilidade de um projeto às versões do sistema operacional. A Figura 1 exibe a relação entre o API Level e o codinome do sistema, bem como o código de sua versão e o NDK (kit de desenvolvimento nativo) por conveniência.

Figura 1 – relações entre codinome, versão de sistema e nível de API

Codename	Version	API level/NDK release
Android10	10	API level 29
Pie	9	API level 28
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19
Jelly Bean	4.3.x	API level 18
Jelly Bean	4.2.x	API level 17
Jelly Bean	4.1.x	API level 16
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7

Fonte: Adaptado pelo autor com base em Android Open Source Project. Disponível em: <https://source.android.com/setup/start/build-numbers>. Acesso em 23 abr. 2019.

2.2.2 Um panorama da compatibilidade

Conforme Android Developers (2011), o Android é desenhado para operar nos mais diversos dispositivos, e para que um aplicativo seja sucedido em todos eles são importantes a existência da variabilidade de recursos e uma interface de usuário que se adeque aos diferentes layouts existentes. Juntamente a isso, *frameworks* dinâmicos oferecem configurações específicas para que elas sejam carregadas com base nas especificações do aparelho onde se localizam; assim, desenvolvedores podem publicar um APK (Android Package Kit) que ofereça uma experiência de usuário aperfeiçoada para cada dispositivo.

Com os pontos listados anteriormente, existem dois tipos de compatibilidade: a compatibilidade de dispositivo e a compatibilidade de aplicativo (ANDROID DEVELOPERS, 2011). Dispositivos só são compatíveis com o Android caso possam

executar corretamente aplicativos escritos para o Ambiente de Execução Android, cujos detalhes são especificados pelo Programa de Compatibilidade Android. Além disso, cada dispositivo deve completar o teste de suíte de compatibilidade (CTS) para ser considerado compatível. Enquanto isso, a preocupação dos desenvolvedores sobre a compatibilidade de aplicativo é inexistente, pois se o app é instalado via Google Play Store, tem-se a garantia de que o usuário está utilizando um dispositivo Android compatível já que todos os aparelhos Android incluem a Play Store.

2.3 Fundamentos de apps

De acordo com Android Developers (2011), os aplicativos para o Android são escritos na linguagem de programação Java e as ferramentas do Android SDK fazem sua compilação para um APK, que contém todo o conteúdo de um app e os arquivos utilizados para instalação em um dispositivo. Após a instalação, a aplicação opera em uma *sandbox* própria (uma área que separa os dados e execução do aplicativo do sistema ou de outros apps), com as seguintes características:

- O Android é um sistema multiusuário, onde cada app é um usuário diferente;
- O sistema emprega um ID de usuário exclusivo a cada app (onde somente o sistema utiliza tal ID), além de definir permissões únicas para cada ID de usuário em todos os arquivos;
- Cada processo possui uma máquina virtual (VM) particular, executando separadamente;
- Cada app opera em seu próprio processo Linux, sendo iniciado quando qualquer componente do aplicativo requer execução e então desligado quando não é mais utilizado, ou quando o sistema necessita recuperar memória.

Consoante Android Developers (2011), com as características listadas anteriormente, nota-se que o Android empreende o conceito do privilégio mínimo,

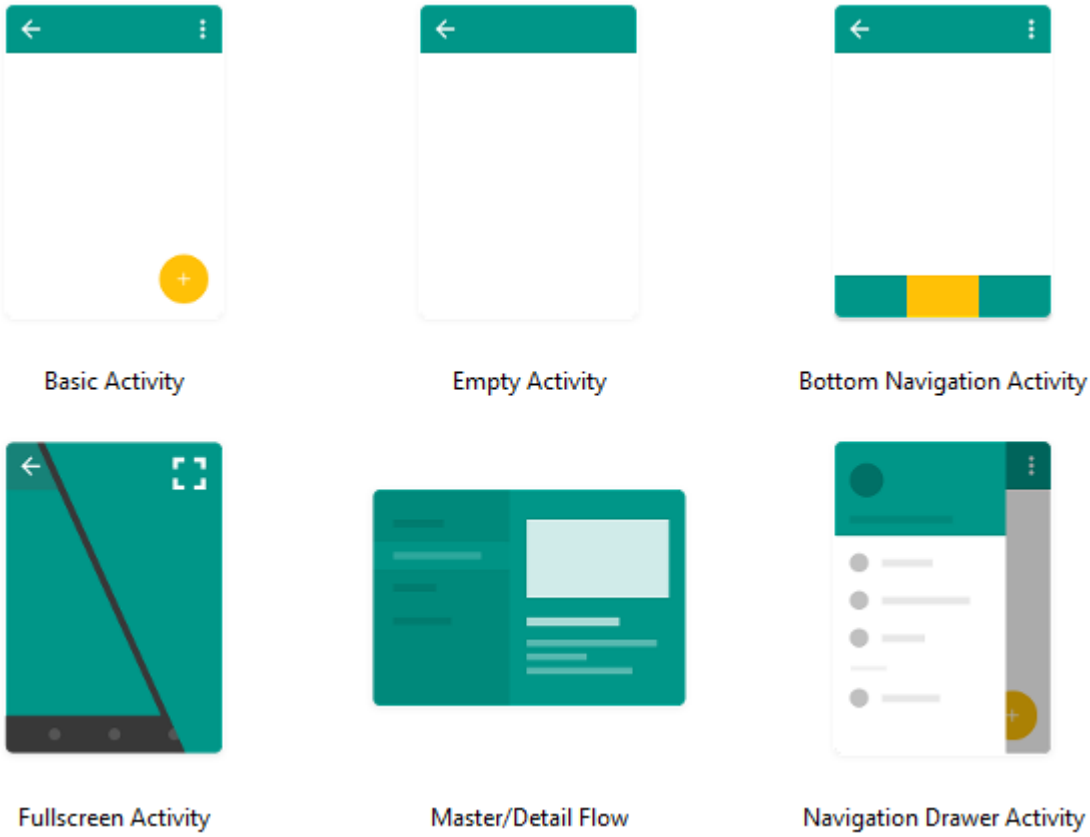
onde uma aplicação só possui acesso aos componentes essenciais para sua operação, e tal ação propicia um ambiente seguro onde um app não é capaz de aceder determinados segmentos do sistema sem as devidas permissões. Todavia, é possível fazer com que dois aplicativos compartilhem de um mesmo ID de usuário e assim acessem os arquivos entre um e outro, e há também a disponibilidade de um app requisitar permissões (as quais devem ser conferidas explicitamente pelo utilizador) para acessar dados de dispositivo, como contatos, mensagens, sistemas de arquivos externos (como cartões SD), câmeras, sensores e rádios (Bluetooth e Wi-Fi), por exemplo.

2.3.1 Componentes

Os componentes de aplicativos são tratados como os blocos estruturais de uma aplicação Android (ANDROID DEVELOPERS, 2011). Cada componente atua como um local de entrada para o app, porém nem todos os locais são pontos de entrada reais para o usuário e alguns podem ser ainda interdependentes. Ainda assim, um componente existe como uma entidade autônoma e desempenha uma tarefa particular, e cada bloco contribui nos procedimentos gerais do aplicativo. Eles são divididos em quatro tipos, cada um com um propósito específico e um ciclo de vida determinado:

A) Atividades (*Activities*) – retratam uma tela única com uma UI (interface de usuário), sendo possível ter várias atividades em um mesmo app. Ainda que elas operem em conjunto para criar uma experiência coesa, elas são independentes entre si. A Figura 2 exibe algumas das atividades disponíveis para utilização em um aplicativo;

Figura 2 – exemplos de atividades disponíveis para uma aplicação

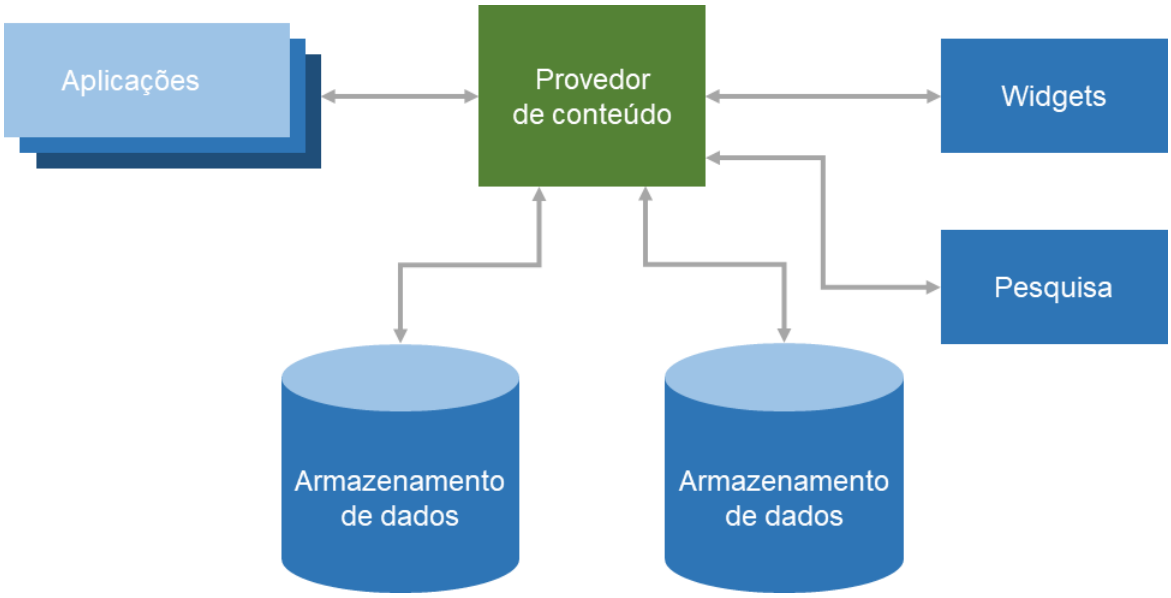


Fonte: Adaptado pelo autor com base na IDE Android Studio (2019).

B) Serviços (*Services*) – são elementos executados em segundo plano para exercer operações longas ou trabalhos para processos remotos, e não apresentam uma UI;

C) Provedores de conteúdo (*Content Providers*) – gerenciam conjuntos compartilhados de dados, armazenando-os em um sistema de arquivos, em um banco de dados SQL (SQLite) ou qualquer outro local de armazenamento persistente. Eles são benéficos para ler e gravar dados privados (e não compartilhados) em um app. A Figura 3 mostra as relações entre os componentes de um aplicativo e o provedor de conteúdo;

Figura 3 – diagrama de relações de um provedor de conteúdo



Fonte: Adaptado pelo autor com base em Android Developers. Disponível em: <https://developer.android.com/guide/topics/providers/content-provider-basics>. Acesso em 24 abr. 2019.

D) Receptores de transmissão (*Broadcast receivers*) – são componentes que respondem a anúncios de transmissão pelo sistema. Ainda que não apresentem uma UI, eles podem criar notificações para alertar o usuário da ocorrência de uma transmissão; entretanto, eles são geralmente um "link" para outros componentes e desempenham uma ínfima quantidade de trabalho.

Segundo Android Developers (2011), qualquer aplicação criada para o sistema Android é capaz de inicializar um componente de outra aplicação. Ao ativar um componente, o sistema executa o processo de tal aplicativo e gera as instâncias das classes essenciais a ele. Com isso, diferentemente de outros sistemas, os aplicativos não possuem nenhum ponto único de entrada. Devido ao sistema executar cada app em um processo individual com restrição de acesso a outros aplicativos, o Android efetua a ativação de um componente pertencente a outro aplicativo, por meio de uma mensagem de sistema explicitando o *intent* de um componente em particular a ser instaurado.

2.3.2 Intents e ContentResolver

As atividades, os serviços e os receptores de transmissão são acionados pelos *intents*, sendo eles tratados como mensagens assíncronas que podem ser explícitas ou implícitas (ANDROID DEVELOPERS, 2011). Eles relacionam componentes individuais entre si, sejam eles pertencentes à aplicação ou não, e estabelecem uma mensagem para ativar tais componentes ou um deles em específico.

Conforme Android Developers (2011), os intents definem o ato de realizar algo (enviar ou exibir, por exemplo) nas atividades e serviços, e podem ainda indicar o identificador de recurso uniforme (URI) dos dados usados na ação. Em determinadas situações, pode-se ainda ativar uma atividade para externar um resultado, retornando-o em um objeto Intent (podendo assim abrigar uma URI mostrando o resultado). Enquanto isso, nos receptores de transmissão, os intents apenas indicam os anúncios que estão sendo transmitidos.

Já os transmissores de conteúdo são ativados por meio do *ContentResolver*, onde o mesmo gerencia todas as transações do provedor de conteúdo para que o componente que esteja realizando tais operações com o provedor não necessite fazê-lo, invocando assim métodos no objeto ContentResolver. Dessa forma, cria-se uma camada de abstração entre o provedor de conteúdo e o elemento que requesta informações.

2.3.3 Manifestos e recursos

Android Developers (2011) relata que um aplicativo deve ter todos os seus componentes declarados em um arquivo de manifesto, localizado no diretório raiz do projeto e chamado de *AndroidManifest.xml*. Tal arquivo XML (linguagem de marcação extensível) identifica ainda as permissões do usuário necessárias à aplicação, especifica o nível mínimo de API necessário exigido (a partir das APIs utilizadas no app), define os requerimentos de hardware e software a serem utilizados, lista as bibliotecas de API a serem vinculadas (sendo outras além das estruturais), além de declarar os componentes do app. A Figura 4 mostra a estrutura

de um arquivo de manifesto de um aplicativo, com as declarações de componentes de uma atividade.

Figura 4 – estrutura de um arquivo de manifesto

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Fonte: Do autor (2019).

Ainda, os aplicativos para Android são compostos por recursos separados do código-fonte, podendo conter imagens, arquivos sonoros, entre outros que se conectam à apresentação do app (ANDROID DEVELOPERS, 2011). Animações, menus, estilos, cores e layouts das interfaces são determinados por arquivos XML, e os recursos de aplicativo permitem a modificação de suas propriedades sem necessidade de alterações de código, além de aperfeiçoá-lo para múltiplos tipos de dispositivos.

Qualquer novo recurso adicionado a um projeto tem um ID exclusivo que pode ser utilizado para referenciá-lo no código da aplicação, gerado automaticamente pelas ferramentas do SDK em um código de recurso chamado “R”, conforme Android Developers (2011). A oferta de recursos dissociados do código-fonte possibilita a entrega de recursos alternativos para diferentes configurações de aparelhos; um exemplo seriam as *strings* de UI, que poderiam ser salvas em diferentes idiomas e aplicadas com base na configuração de idioma do usuário,

utilizando qualificadores de idioma. Outro exemplo são os possíveis diferentes layouts para telas, contendo disposições únicas dos itens nos modos retrato e paisagem e que são aplicados dinamicamente conforme a orientação do aparelho, utilizando os qualificadores apropriados para cada layout.

2.4 Android App Bundle e Dynamic Delivery

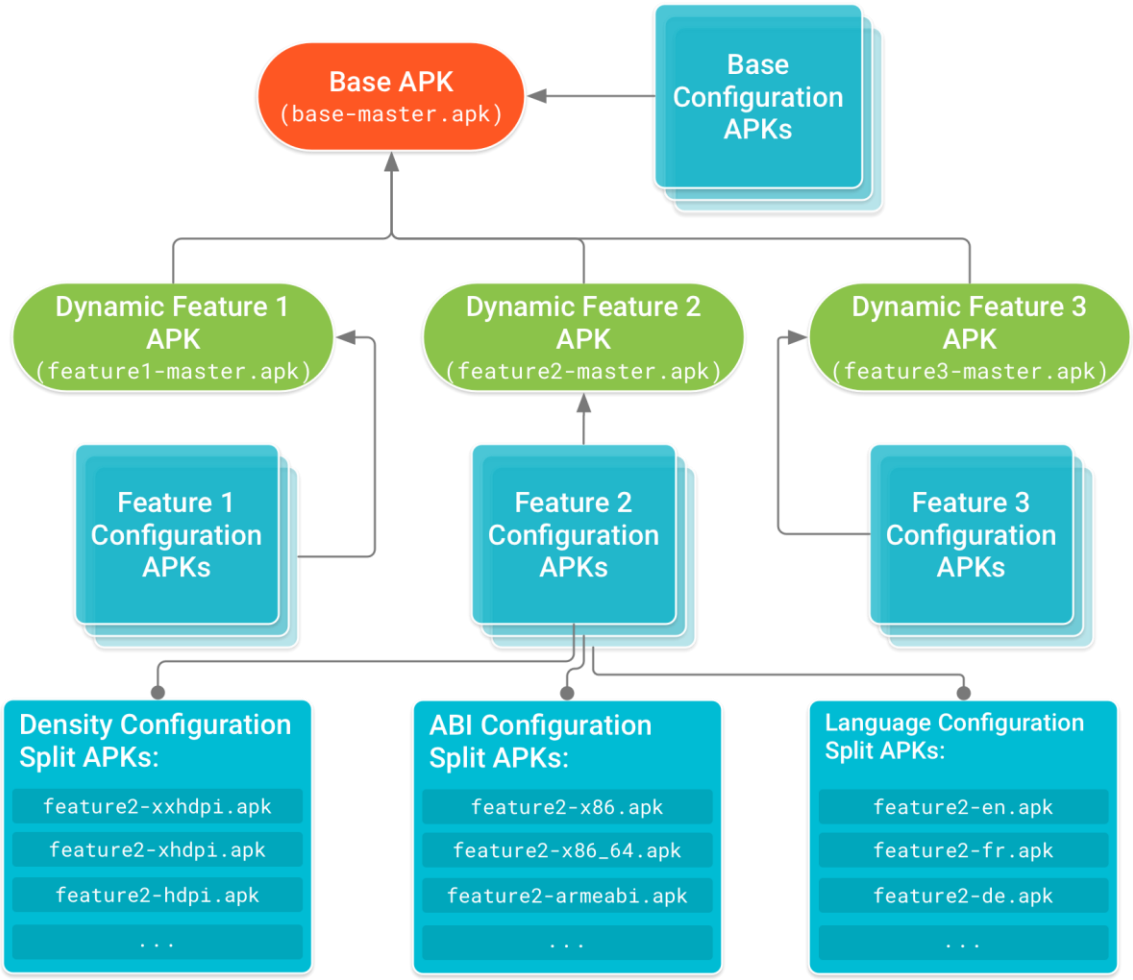
Consoante Android Developers (2018), há um novo formato de envio de aplicações, chamado de Android App Bundle. Ele inclui somente o código compilado e os recursos do projeto, e a Google Play passa a ser responsável pela geração e assinatura dos arquivos APK, enquanto a entrega de tais arquivos é feita pelo novo modelo de fornecimento da Google Play, chamado de Dynamic Delivery. Com ele, os desenvolvedores não precisam mais gerenciar múltiplos APKs para os diferentes dispositivos em que operariam, e os usuários recebem seus downloads em menor tamanho.

Além disso, é possível adicionar módulos de recursos dinâmicos a um projeto e incluí-los ao App Bundle, os quais podem ser requeridos em outro momento por meio da biblioteca Play Core e entregues pela Google Play por meio do Dynamic Delivery (ANDROID DEVELOPERS, 2018). Módulos contendo recursos e ativos podem ser configurados para não serem instalados no primeiro download da aplicação; então, por meio do Dynamic Delivery, transferem-se sob demanda os códigos e recursos de um determinado módulo a um aparelho em qualquer outro momento. Tais recursos estão disponíveis aos desenvolvedores a partir da versão 3.2 do Android Studio.

Para Android Developers (2018), o componente essencial do Dynamic Delivery é o mecanismo de *split APK* (APK dividido), disponível a partir do nível de API 21. APKs divididos são quase idênticos a APKs comuns; no entanto, o Android considera múltiplos APKs divididos que estão instalados como uma única aplicação. O benefício de tal mecanismo é ter a possibilidade de segmentar um APK monolítico em pacotes menores, que podem ser instalados sob demanda. A Figura 5 exibe as dependências em forma de árvore de um aplicativo disponibilizado por meio de APKs divididos, com o APK base atuando como topo da mesma e todos os outros

APKs vinculados à base. Eles não necessitam de construção por parte do desenvolvedor, com tal atividade ficando a cargo da Google Play.

Figura 5 – árvore de dependência de um APK dividido



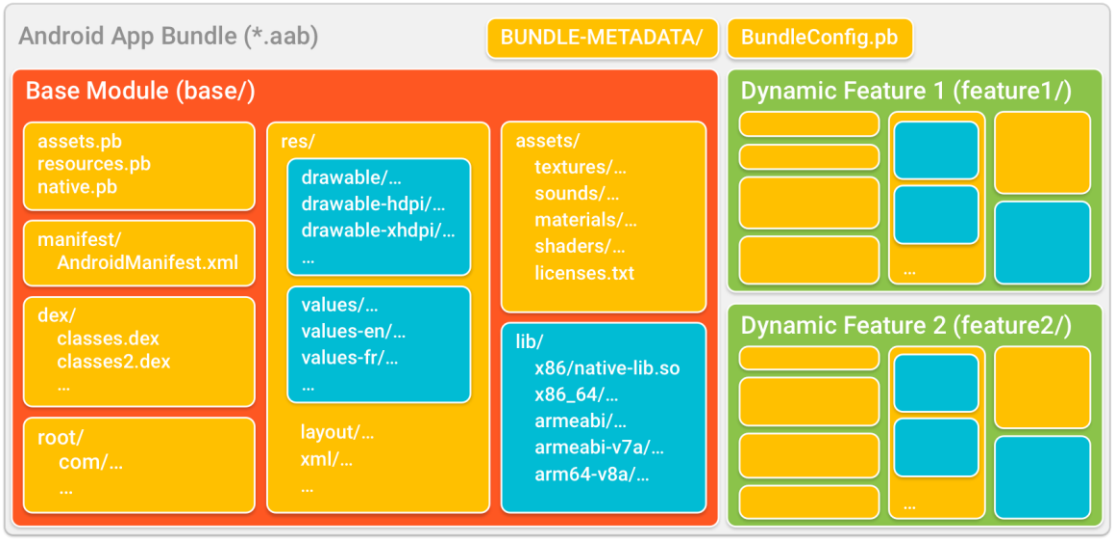
Fonte: Android Developers. Disponível em: <https://developer.android.com/guide/app-bundle#dynamic_delivery>. Acesso em 12 abr. 2019.

Devido à inexistência do suporte ao recurso de APKs divididos no nível de API 19 (e anteriores), o Google Play realiza a transferência dos chamados *multi-APK*, que são otimizados às configurações dos dispositivos a serem instalados e não incluem códigos e recursos supérfluos a eles (ANDROID DEVELOPERS, 2018). Entretanto, módulos de recursos dinâmicos não podem ser instalados em multi-APKs sob demanda, sendo necessário ativar a opção “Fusing” no manifesto do módulo dinâmico ou desativar tal módulo por completo.

2.4.1 Formato de um App Bundle

Segundo Android Developers (2018), um App Bundle contém binários assinados que arranjam os códigos e recursos de um projeto em módulos, onde tais códigos e recursos são organizados homogeneamente para cada um dos mesmos. Então, a Google Play utiliza o App Bundle para criar os vários APKs a serem entregues aos usuários, sejam eles o APK base, os módulos dinâmicos, o APK de configurações ou os multi-APK. A Figura 6 ilustra a organização dos módulos de um arquivo do Android App Bundle (que contém a extensão *aab*), sendo os blocos azuis códigos e recursos que são empregados pela Google Play na criação dos APKs de configurações para cada módulo. Eles contêm informações como as diferentes resoluções de tela para as atividades (bloco "*drawable*"), os idiomas da interface (bloco "*values*") e as bibliotecas a serem utilizadas pelo app (bloco "*lib*"), onde são entregues apenas os arquivos necessários ao aparelho destino.

Figura 6 – conteúdo de um Android App Bundle, com um módulo base e dois módulos de recursos dinâmicos



Fonte: Android Developers. Disponível em: <https://developer.android.com/guide/app-bundle#aab_format>. Acesso em 12 abr. 2019.

Os blocos em cada módulo da Figura 6 são detalhados da seguinte forma:

A) base/, feature1/ e feature2/ – são os diretórios localizados no topo da aplicação, sendo o módulo base sempre contido em um diretório base do App Bundle. Porém, os diretórios para cada módulo dinâmico são especificados no atributo "split" do manifesto de cada módulo;

B) BUNDLE-METADATA/ – diretório com arquivos de metadados utilizados por ferramentas ou lojas de aplicativos. Eles incluem uma lista completa dos arquivos *dex* de um app, e não são inclusos em um APK;

C) Arquivos .pb – são arquivos que descrevem o conteúdo de cada módulo para lojas de aplicativos, oferecendo dados sobre o mesmo (arquivo "*BundleConfig.pb*"), ou descrições do código e recursos dos módulos (arquivos "*native.pb*" e "*resources.pb*") que são então utilizados para otimizar um APK para diferentes configurações;

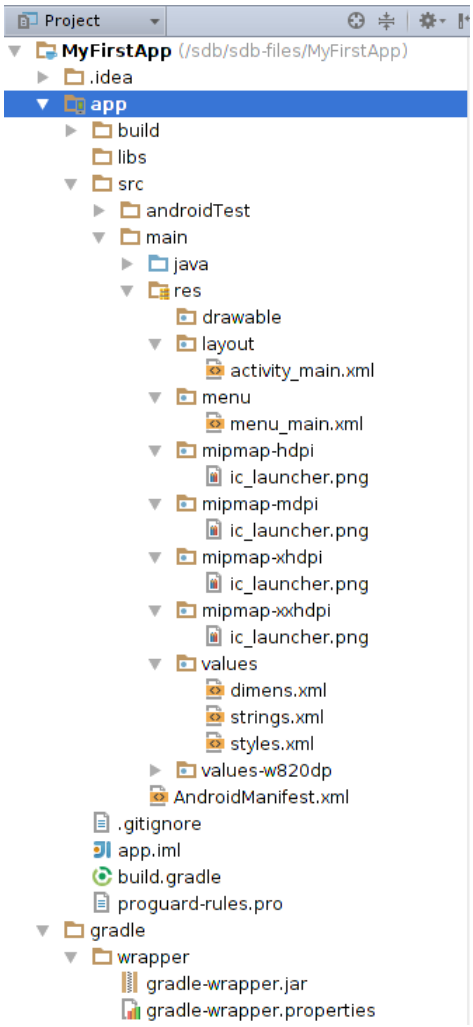
D) manifest/ – diretório que armazena os arquivos de manifesto de cada módulo;

E) dex/ – diretório que armazena os arquivos *.dex* de cada módulo;

F) res/, lib/ e assets/ – diretórios idênticos a um APK comum, onde a Google Play os examina e seleciona apenas os arquivos que atendem às configurações de determinado dispositivo;

G) root/ – diretório que guarda arquivos que são relocados para a raiz de qualquer APK que inclui o módulo localizado nesse diretório. A Figura 7 mostra a estrutura dos diretórios detalhados em D, E, F e G.

Figura 7 – estrutura de diretórios, pastas e arquivos de um projeto



Fonte: Android Developers. Disponível em: <https://developer.android.com/studio/projects>. Acesso em 26 mai. 2019.

Um App Bundle diferencia-se de um APK pelo fato de não ser possível implantá-lo em um dispositivo, sendo na verdade um formato de upload que inclui os códigos e recursos compilados em um único artefato (sendo ele o arquivo com extensão *aab*), onde a Play Store encarrega-se de construir, assinar e conferir os APKs por meio do Dynamic Delivery (ANDROID DEVELOPERS, 2018).

2.4.2 Solicitação e instalação de módulos

Conforme Android Developers (2018), caso uma aplicação faça uso de módulos de recursos dinâmicos, ela deve fazer uso da biblioteca Play Core para solicitar, monitorar e gerenciar os downloads de tais módulos. Com isso, dispositivos

que possuem o nível de API 21 e superiores são capazes de instalar módulos dinâmicos sob demanda, sendo necessário apenas a realização das chamadas para as APIs da biblioteca para baixar e instalar os módulos solicitados, cujos códigos e recursos são então enviados pela Google Play Store ao dispositivo-alvo.

Um aplicativo pode requisitar um módulo dinâmico enquanto estiver em primeiro plano por meio da classe *SplitInstallManager* (ANDROID DEVELOPERS, 2018). Ao realizar a requisição, o nome do módulo deve ser especificado identicamente ao elemento *split* no manifesto do módulo-alvo. Além disso, é possível postergar a instalação de um módulo para o momento em que o aplicativo esteja em segundo plano, disponibilizando-o na próxima execução do app.

Segundo Android Developers (2018), as solicitações de módulos de recursos dinâmicos são gerenciadas pela biblioteca citada anteriormente em uma estratégia "*fire-and-forget*": ela realiza o envio da solicitação feita ao dispositivo, porém não há o controle do sucesso ou falha da instalação do módulo. É possível inspecionar o progresso da requisição e realizar o manejo de alertas ao usuário em caso de sucesso ou falha da requisição ou implantação do módulo; no entanto, não é possível perscrutar módulos com sua instalação adiada, sendo necessário então a verificação prévia de sua instalação.

2.4.3 Propriedades de módulos dinâmicos

Módulos de recursos dinâmicos permitem a divisão de componentes e recursos do módulo base de uma aplicação, sendo possível adicioná-los a um App Bundle (ANDROID DEVELOPERS, 2018). Tal modularização pode exigir refatoração do código e maior esforço para desenvolvimento, logo é necessário avaliar minuciosamente quais recursos da aplicação seriam os mais proveitosos em estar disponíveis para entrega sob demanda.

De acordo com Android Developers (2018), módulos dinâmicos são compostos da mesma forma que módulos de aplicativos, provendo códigos, recursos e ativos nos diretórios já conhecidos. Entretanto, atributos extras no arquivo

de manifesto e propriedades adicionais na configuração do construtor permitem que os módulos sejam baixados sob demanda.

Android Developers (2018) explica que no momento da criação de um módulo dinâmico em um projeto, o Android Studio aplica o plugin do Gradle “*dynamic-feature*” no arquivo de configuração do módulo (chamado de *build.gradle*), permitindo que o sistema de build origine o App Bundle que inclua o módulo dinâmico. Porém, configurações de assinatura, propriedades de encolhimento de código, código de versão e nome de versão devem ser subtraídos do arquivo build.gradle, pois um módulo dinâmico sempre está vinculado ao módulo base e herda determinadas configurações do mesmo.

3 PROCEDIMENTOS METODOLÓGICOS

Apresentam-se neste capítulo os métodos utilizados no desenvolvimento do projeto, tendo como embasamento os conhecimentos teóricos.

Esta seção está subdividida em quatro grupos: método de pesquisa, modo de abordagem, objetivos e procedimentos técnicos da pesquisa.

3.1 Métodos de pesquisa

O método de pesquisa adotado foi o dedutivo, pois de acordo com Lakatos e Marconi (2003) no método dedutivo a conclusão é verdadeira se todas as premissas são verdadeiras também. O método dedutivo foi empregado ao projeto devido à natureza do raciocínio sucessivo do tema exposto, onde há a análise geral até a particular para a verificação da hipótese e a conclusão dos objetivos propostos.

3.2 Modo de abordagem

O modo de abordagem da pesquisa foi o qualitativo. Segundo Neves (2015), a pesquisa qualitativa se preocupa com a análise subjetiva dos dados pesquisados e não com sua quantidade, sendo assim possível entender qual o propósito do objeto de pesquisa. Tal modo de abordagem aplicou-se ao projeto em virtude da necessidade de validação dos objetivos específicos apresentados, a qual ocorreu

por meio de validações e coleta de informações via testes de recursos técnicos. Tais testes contêm dados como o tamanho do aplicativo, dos módulos e o gerenciamento da entrega dos mesmos. Dessa forma, os dados podem variar de acordo com o cenário em que a aplicação está inserida.

3.3 Objetivos da pesquisa

A metodologia quanto ao objetivo foi a exploratória, pois para Cervo, Bervian e da Silva (2007), a pesquisa exploratória é comumente o primeiro passo na pesquisa por experiência. Consoante a isso, Castro (2006) ressalta que na abordagem exploratória não há revisão sistemática prévia e nem discussões de teorias alternativas. A pesquisa foi exploratória devido à baixa exploração do assunto exposto e ao objetivo de contextualizar o pesquisador empiricamente acerca do tema, tomando como base o problema e os objetivos apresentados no primeiro capítulo.

3.4 Procedimentos técnicos usados na pesquisa

A metodologia quanto ao procedimento técnico foi a pesquisa bibliográfica, pois para Gil (2002) a pesquisa bibliográfica é composta por livros e artigos científicos, fundamentada em materiais já criados, e de acordo com Silveira e Gerhardt (2008) a pesquisa bibliográfica surge a partir de referências analisadas. O projeto vai de encontro ao procedimento bibliográfico devido ao estudo de materiais e registros existentes e reconhecidos para apresentação da proposta e do desenvolvimento da referida aplicação.

3.5 Ferramentas e dispositivos

O trabalho foi desenvolvido por meio da IDE Android Studio em linguagem Java, utilizando o formato App Bundle, e a criação e entrega dos módulos dinâmicos ocorreram por meio do Dynamic Delivery, uma tecnologia pertencente ao formato

citado anteriormente. O aplicativo foi hospedado na Google Play Store, devido ao uso de APIs necessárias ao projeto existentes somente na loja do Google. Para a realização dos testes e a coleta de dados, o app foi instalado em dois smartphones distintos, fabricados pela Samsung e Xiaomi. Cada um possui uma versão específica do sistema operacional Android, dessa forma sendo possível avaliar a compatibilidade do novo formato de desenvolvimento em diversas versões do sistema.

4 TRABALHOS RELACIONADOS

Nesta seção, apresenta-se um trabalho relacionado, o qual foi o principal motivador para o desenvolvimento do presente projeto.

Balbinot (2018) descreveu em sua monografia o desenvolvimento de uma aplicação voltada ao sistema operacional Android, com o intuito de construir uma plataforma de distribuição de conteúdo. Por essa plataforma, usuários poderiam disponibilizar seus conteúdos ou ainda obter conteúdo de terceiros.

Os conteúdos estariam hospedados em um serviço de infraestrutura de armazenamento, pela qual seria possível as distribuições aos dispositivos-alvo. Então, a partir de uma aplicação móvel básica, usuários realizariam o download dos conteúdos desejados, os quais eram basicamente aplicativos para o sistema Android. O app base continha uma lista com os apps localizados no serviço, e ao selecionar uma aplicação, um arquivo APK era baixado, o qual precisava ser instalado manualmente em um momento posterior pelo usuário.

O principal objetivo do projeto proposto na monografia de Balbinot (2018) era a criação de um serviço híbrido, já que o principal problema do seu aplicativo era a necessidade de atualização do mesmo a cada conteúdo inserido ou removido do serviço de armazenamento. Com o surgimento do Android App Bundle e visando a solução do problema citado anteriormente, sucedeu-se a ideia de desenvolver um aplicativo no novo formato de desenvolvimento, dando origem ao presente trabalho.

O presente projeto difere-se do serviço proposto por Balbinot (2018) por utilizar outro formato de desenvolvimento das aplicações, bem como possuir um funcionamento diferente na entrega dos arquivos de instalação dos aplicativos aos dispositivos. Enquanto o serviço citado anteriormente realizava a entrega de arquivos APK aos aparelhos, a aplicação descrita pelo trabalho atual tem os arquivos de instalação gerados automaticamente pela Play Store e então enviados aos dispositivos. Além disso, o aplicativo faz uso da própria loja para hospedar seu conteúdo, diferentemente do serviço que recorre à uma infraestrutura de armazenamento.

5 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo, é apresentada a descrição do desenvolvimento realizado para a execução do projeto. O presente trabalho propôs o desenvolvimento de um aplicativo para smartphones, com módulos dinâmicos a serem instalados sob demanda.

5.1 Proposta inicial

Inicialmente, a proposta do presente trabalho apresentou o desenvolvimento de um app no novo formato Android App Bundle, contendo dois módulos de recursos dinâmicos que contêm cenários tridimensionais distintos, para fins educacionais e/ou recreativos. Um dos módulos compreende um cenário aberto com visualização em 360 graus, e o outro uma visualização tridimensional com a análise de um esqueleto humano.

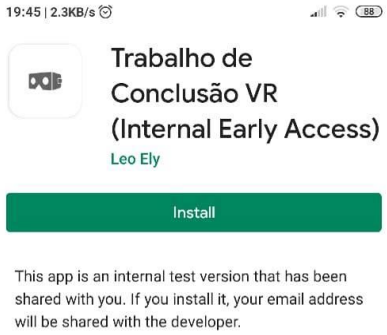
Então, criou-se o aplicativo, e um cenário aberto foi projetado pelo motor de jogo Unity, sendo posteriormente incorporado a um módulo de recurso dinâmico. Em seguida, o app foi enviado à Play Store para publicação e realização das avaliações e dos testes expostos nos objetivos do trabalho. Após três dias de avaliação, a aplicação foi publicada, e em seguida criou-se uma faixa de teste interno para validação do app. A faixa de teste interno, que é um recurso disponibilizado e gerenciado pela loja, possibilita o acesso prévio de um aplicativo a uma lista restrita

de usuários, onde os mesmos podem realizar testes e retornar logs de erro encontrados ao desenvolvedor.

5.1.1 Validação da proposta

Para validação do app submetido, realizaram-se testes em dois dispositivos físicos: um Samsung Galaxy S8, e um Xiaomi Redmi Note 7. Ambos foram escolhidos devido a um requisito de hardware específico dos cenários de Realidade Virtual, que era a presença do giroscópio. A Figura 8 mostra o aplicativo disponível para download na loja, extraída do aparelho da Xiaomi.

Figura 8 – aplicativo publicado na Play Store

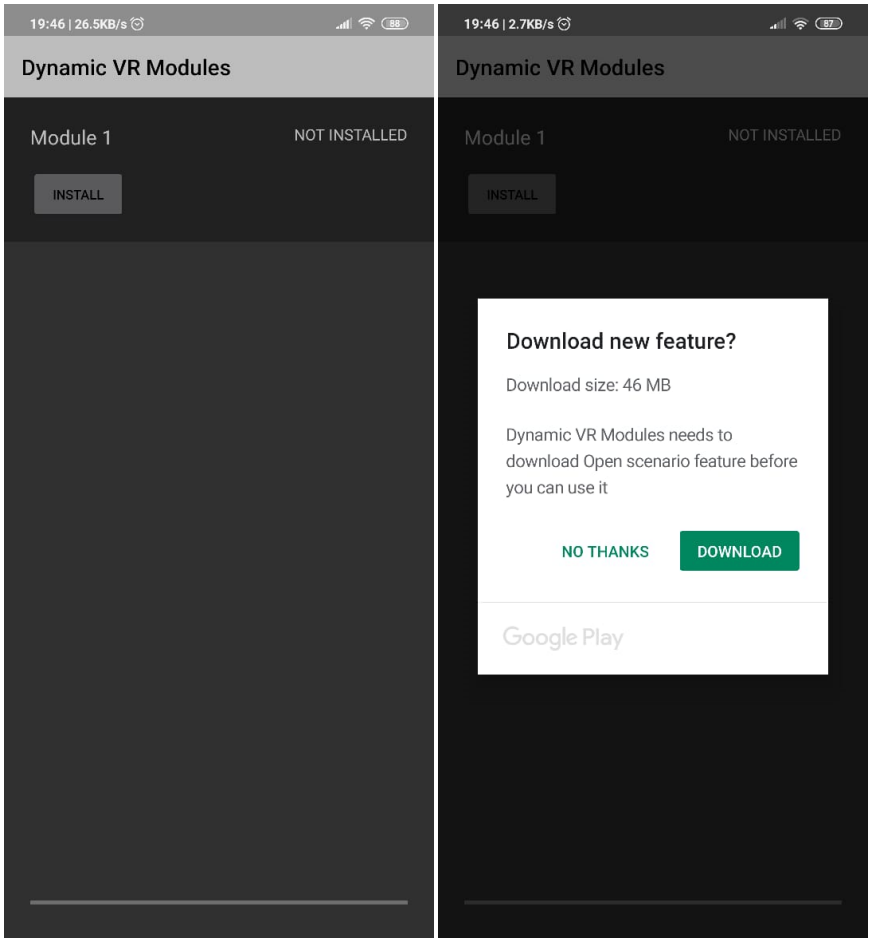


Fonte: Do autor (2019).

Após o download e instalação do app, o mesmo foi inicializado, para comprovação de seu funcionamento. Em sua abertura, verificou-se que o módulo de recurso dinâmico não estava instalado, já que ele estava configurado para ser instalado sob demanda. Ao solicitar a instalação do módulo, um pop-up foi exibido solicitando ao usuário a confirmação do download, um comportamento padrão para módulos de recursos dinâmicos que ultrapassem os 10 MB de tamanho. Seguinte à confirmação, uma barra de progresso permitia o acompanhamento do processo de download e instalação, e o aplicativo era recarregado após a instalação do módulo para permitir sua utilização. A reinicialização do app é um procedimento padrão a ser adotado após o download de um módulo, pois o módulo base não possui controle sobre o status da instalação de um recurso sob demanda. A Figura 9 exibe

à esquerda a tela principal do app, e à direita o pop-up de confirmação do download do módulo.

Figura 9 – telas principais da aplicação



Fonte: Do autor (2019).

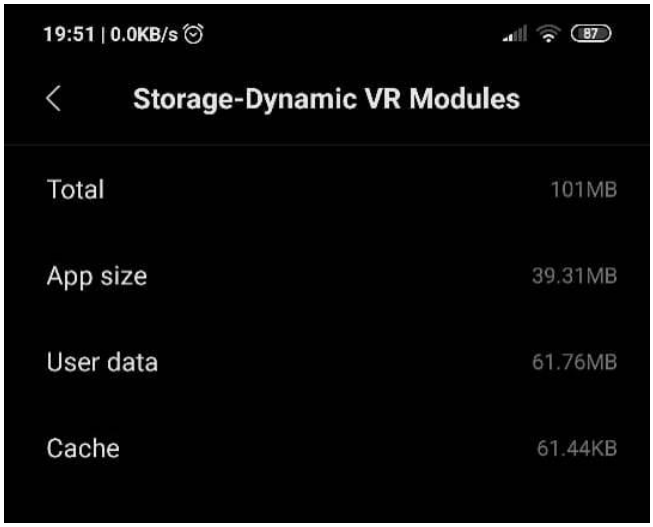
5.1.2 Resultados obtidos

Com os processos realizados anteriormente, pode-se perceber que o download e instalação de um módulo de recurso dinâmico ocorre de forma transparente ao usuário. O último apenas realiza as solicitações de instalação e as confirma quando necessário, e todo o resto das ações ficam a cargo do aplicativo e da Google Play.

Para avaliação do espaço utilizado pela aplicação, verificou-se na seção de configurações do dispositivo a quantidade de armazenamento utilizada. Após a

instalação do módulo, foi possível constatar que o tamanho ocupado pelo app é apresentado de forma segmentada, com o módulo base sendo exibido como “tamanho do app” e o módulo de recurso dinâmico como “dados do usuário”. As quantidades utilizadas para o dispositivo da Xiaomi foram de 39,31 MB e 61,76 MB, respectivamente. A Figura 10 mostra as quantidades de armazenamento utilizadas pela aplicação.

Figura 10 – quantidade de armazenamento utilizada pelo app



Fonte: Do autor (2019).

A fim de obter dados sobre o espaço ocupado pelo app no formato antigo, criou-se um APK único para instalação em um dispositivo, tendo como base a mesma aplicação. O mesmo utilizava cerca de 150 MB de armazenamento, enquanto o APK gerado pela Play Store a partir de um App Bundle utilizava 101 MB. Com isso, é possível notar que a geração dos aplicativos otimizados reduz o espaço de armazenamento utilizado, o qual varia para cada dispositivo.

Porém, ao realizar a abertura do módulo, era apresentado um erro referente às bibliotecas utilizadas pela Unity em cenários tridimensionais com animações de objetos. Tentativas de pré-carregamento das bibliotecas, bem como da transferência das mesmas à base do aplicativo foram realizadas; no entanto, nenhuma delas foi bem-sucedida. Para comprovação do funcionamento do cenário, o módulo foi desvinculado da aplicação, e então criou-se outra aplicação a partir do módulo, utilizando o mesmo como base. A nova aplicação foi enviada à loja e instalada em um dispositivo, onde funcionou sem nenhum retorno de erro. A Figura 11 mostra o

log de erro retornado na aplicação, referente às bibliotecas utilizadas quando utilizado o recurso da Unity como módulo.

Figura 11 – log de erro do módulo de recurso dinâmico

Fonte: Do autor (2019).

Conforme a documentação oficial da Unity (2019), ao exportar um projeto para a plataforma Android, o sistema de build faz a preparação dos recursos, a compilação de scripts, processamento de plugins e as construções da biblioteca IL2CPP (que converte códigos intermediários para scripts C++) em sequência. Uma pesquisa sobre o motivo do não-funcionamento das bibliotecas foi conduzida, e com ela notou-se que os apontamentos das bibliotecas pelos códigos compilados tornam-se incorretos ao importar um projeto gerado pela Unity a um módulo de recurso

dinâmico, o que causa os erros de execução observados nos logs de erro obtidos pelos testes.

Com tais resultados, conclui-se que a incorporação de recursos de cenários tridimensionais gerados pela Unity a módulos de recursos dinâmicos ainda não é possível, por ser uma tecnologia em desenvolvimento. Devido aos resultados obtidos, houve a necessidade de alteração da proposta de desenvolvimento do projeto, a qual está detalhada nos parágrafos a seguir.

5.2 Visão geral da nova proposta

Em uma nova proposta, o projeto realizou a estruturação de uma nova aplicação que inclui módulos de recursos dinâmicos, cada um com um módulo específico de um sistema ERP (sistema integrado de gestão). É possível realizar o download de qualquer um dos módulos a qualquer momento, uma vez que a instalação do app em um dispositivo contará somente com a base da aplicação, de onde é possível instalar os módulos.

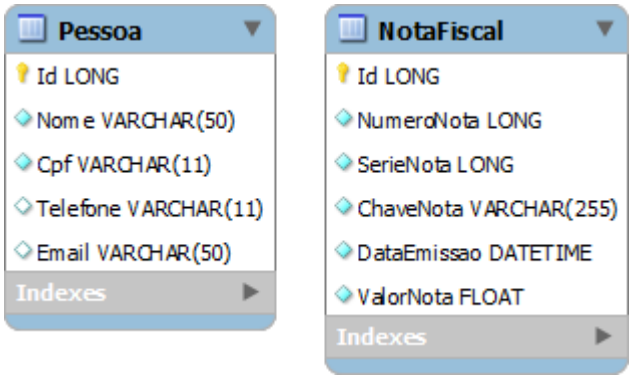
O propósito de modularização de um sistema ERP surge do fato de que os utilizadores do aplicativo não necessitam do sistema em sua totalidade, visto que funcionários de um determinado setor não precisam de módulos os quais eles não farão uso.

5.2.1 Módulos do aplicativo

Visando a modularização de um sistema empresarial qualquer (sendo a empresa pertencente ao ramo de transportes, vendas ou negócios), dois módulos foram desenvolvidos, um voltado ao controle de Pessoas e outro ao controle de Notas. Ambos os módulos realizam a leitura e armazenamento dos dados em um banco de dados, que está hospedado em uma máquina virtual em um servidor cloud.

Foram criadas duas entidades para serem utilizadas na aplicação. A Figura 12 apresenta o diagrama ER (Entidade-Relacionamento) das entidades, contendo os campos utilizados por cada um dos módulos. Cada entidade é um modelo no diagrama, e possui sua própria tabela no banco de dados.

Figura 12 – diagrama ER da aplicação



Fonte: Do autor (2019).

5.2.2 API REST

Para apoiar o processo de consumo de dados utilizados pelos módulos, foi desenvolvida uma API RESTful, que se baseia nos princípios do REST (Transferência de Estado Representacional). Ele consiste em regras e restrições que possibilitam a criação de sistemas com interfaces definidas, permitindo assim que a API citada anteriormente seja capaz de manipular e enviar dados a um banco de dados hospedado em um servidor remoto. Além disso, devido à utilização da Internet para o consumo dos dados, utilizou-se o formato JSON (notação de objetos JavaScript) para troca de informações, por ser um padrão leve e de ampla utilização.

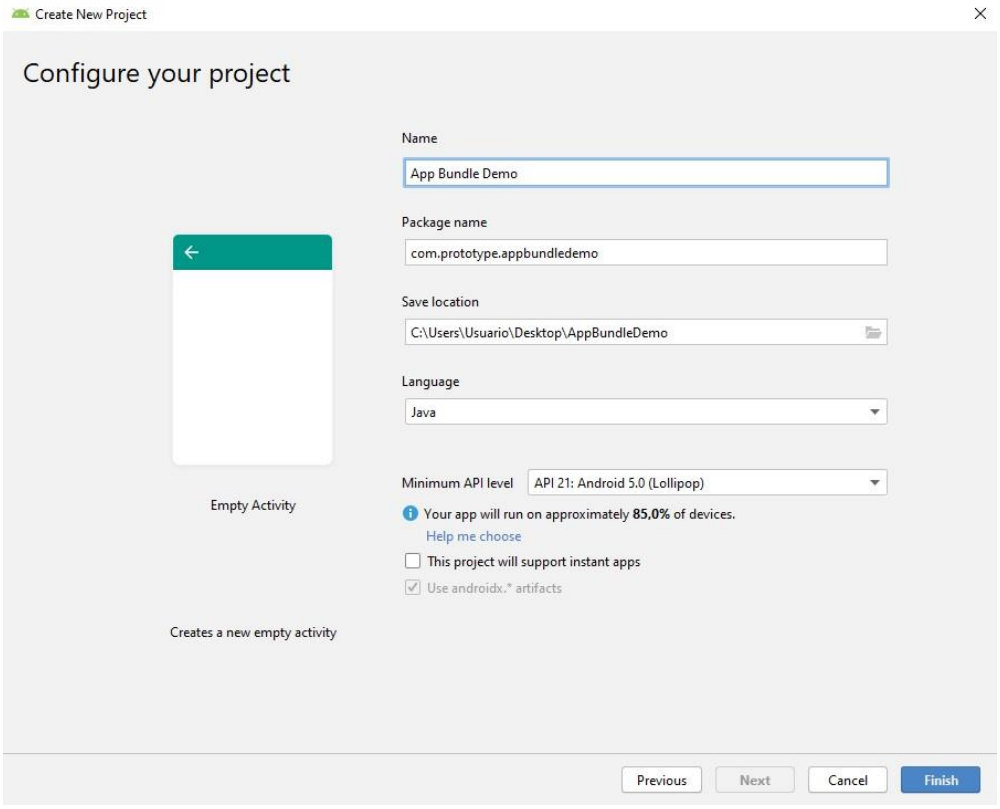
A API destinada ao projeto utiliza o *framework* ASP.NET Core, escrito com a linguagem C#. Tal escolha ocorreu devido à facilidade de uso e à robustez da tecnologia, além da possibilidade de configuração do banco de dados diretamente pela API. Para o desenvolvimento da mesma, utilizou-se a IDE Visual Studio Community, padrão para o desenvolvimento de projetos destinados à plataforma .NET.

Após a criação da API, a mesma foi implantada em uma máquina virtual, cuja máquina foi criada com o sistema operacional Ubuntu Server na plataforma Azure, da Microsoft. Essa plataforma foi escolhida pela estabilidade e por oferecer uma URL pública, pela qual o aplicativo pode conectar-se e consumir os dados armazenados no banco de dados da máquina virtual.

5.2.3 Aplicação Android

O aplicativo foi desenvolvido por meio da IDE Android Studio, sendo ela recomendada para a criação de apps que contenham módulos de recursos dinâmicos. O primeiro passo foi iniciar um novo projeto contendo somente uma atividade vazia, a qual abriga a lista de módulos disponíveis no aplicativo. Em seguida, a IDE solicita o nome do aplicativo, a linguagem na qual ele será desenvolvido e o nível mínimo da API a ser utilizada. Dados adicionais são solicitados (como o nome do pacote, o diretório do computador em que será armazenado o projeto, entre outros dados), os quais variam para cada cenário e não são determinantes no desenvolvimento. No caso do projeto descrito pelo presente trabalho, os dados relevantes são a linguagem Java e o nível mínimo de API 21 (para o correto funcionamento da modularização do app). A Figura 13 mostra o processo de criação do projeto na IDE.

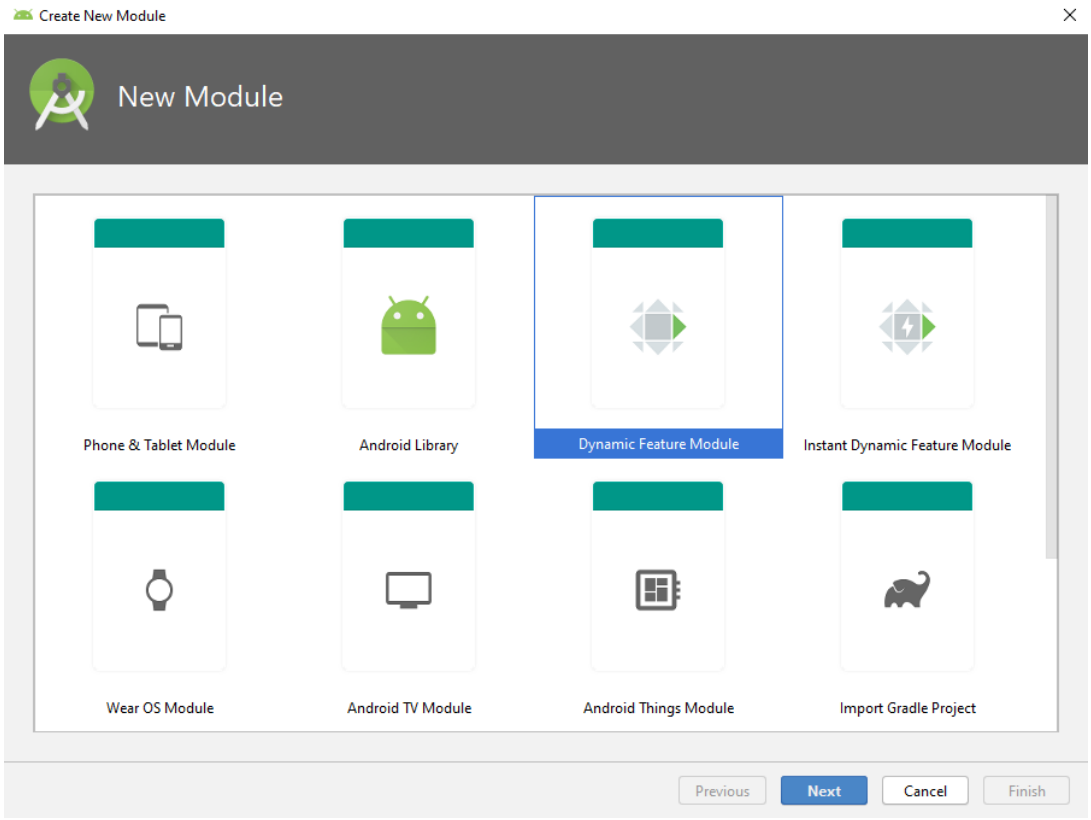
Figura 13 – tela de configuração de novo projeto



Fonte: Do autor (2019).

Após a criação do novo projeto, foram adicionados os módulos de recursos dinâmicos à aplicação. O processo é acessível tanto pelo menu superior, quanto pela raiz do projeto, ao clicar com o botão direito do mouse no topo da árvore do projeto. Em seguida, escolhe-se a opção “New → New Module”, e então uma janela é aberta, solicitando ao usuário qual módulo deve ser criado. Seleciona-se então a opção “Dynamic Feature Module”, conforme exposto na Figura 14.

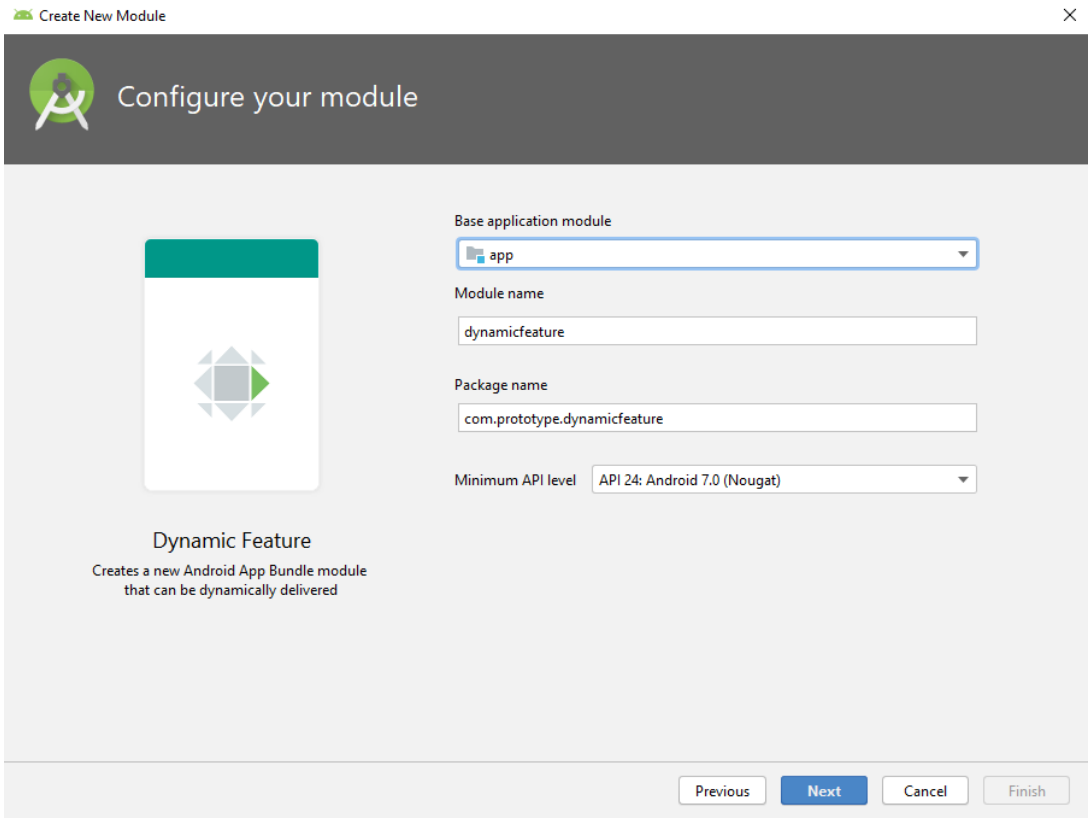
Figura 14 – criação do módulo de recurso dinâmico



Fonte: Do autor (2019).

Então, a IDE solicita qual é o módulo base da aplicação, o nome do novo módulo, o nome do pacote (automaticamente preenchido ao selecionar o módulo base) e o nível mínimo de API. Da mesma forma, o nível de API mínimo recomendado é o 21 (alinhado com o nível de API do módulo base). Para o projeto, o nível escolhido foi 24 devido ao uso de funções e métodos existentes somente na versão 8 da linguagem Java, a qual é compatível nativamente a partir do nível de API mencionado anteriormente. A Figura 15 mostra o processo de configuração dos dados citados.

Figura 15 – configuração do módulo de recurso dinâmico



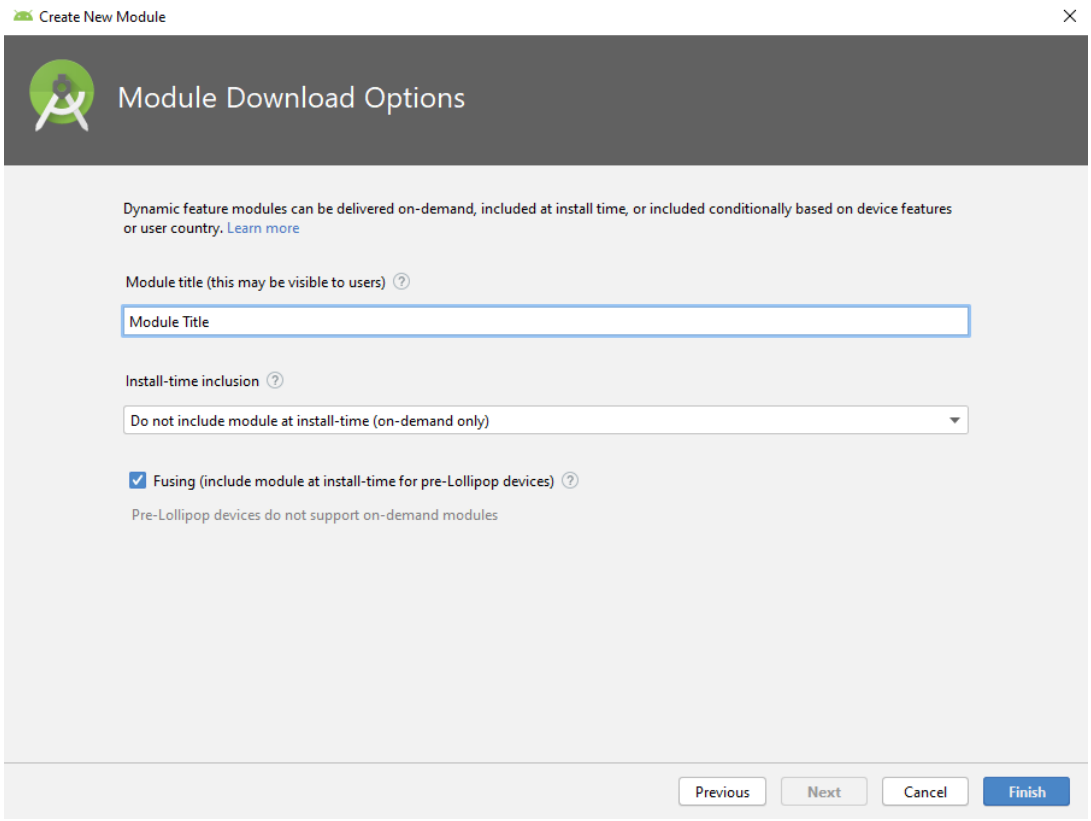
Fonte: Do autor (2019).

Na próxima etapa, escolhe-se o título do novo módulo, sua inclusão no projeto (o ponto central da presente pesquisa) e a inclusão do atributo “Fusing” (para disponibilizar o módulo a dispositivos com versões anteriores a API de nível 21, ativada por padrão). A inclusão pode ocorrer nas seguintes condições:

- Não incluir em tempo de instalação (módulo somente on-demand) – a instalação do módulo é feita posteriormente, quando solicitada;
- Incluir módulo em tempo de instalação – o módulo é incluso ao aplicativo quando o mesmo for instalado;
- Incluir módulo em tempo de instalação para dispositivos com recursos específicos – o módulo é incluído ao aplicativo quando o mesmo for instalado, porém somente em dispositivos que atendam os recursos especificados (como determinados requisitos de hardware, por exemplo).

Para o projeto, foi escolhida a opção somente on-demand, seguindo os objetivos referidos no Capítulo 1, onde a Figura 16 exibe a configuração final para o módulo de recurso dinâmico. Devido à proposta de criação de dois módulos, o processo é o mesmo para ambos, apenas alterando o nome e título de cada módulo (“Pessoas” para o primeiro módulo, e “Notas” para o segundo módulo).

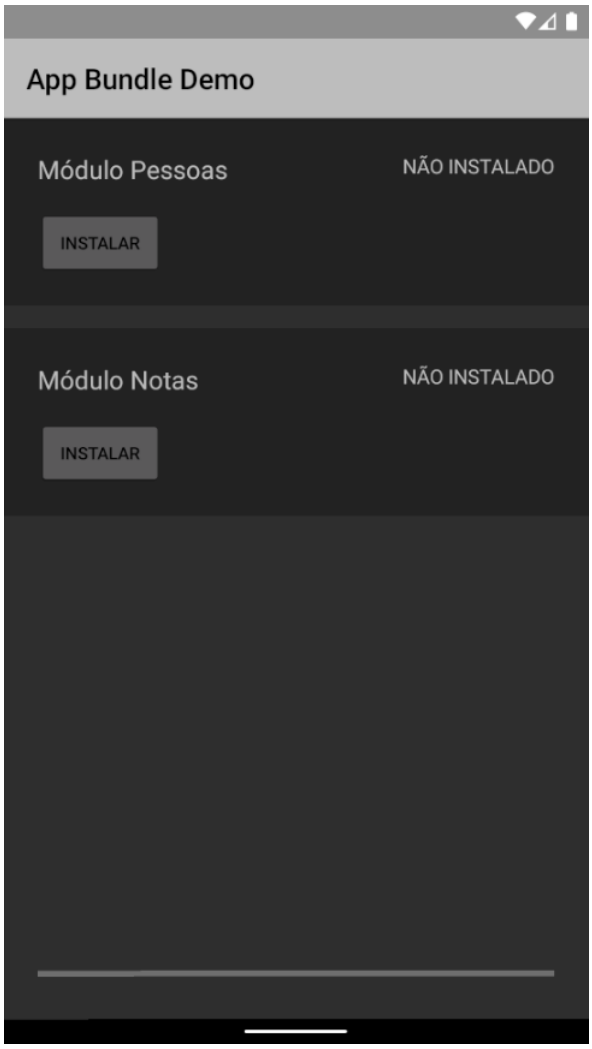
Figura 16 – configurações finais do módulo



Fonte: Do autor (2019).

Em seguida, foi criado o layout da tela principal do app, contendo a lista de módulos disponíveis. A Figura 17 mostra a tela desenvolvida, contendo o título do módulo, seu status de instalação e um botão de ação. Para cada caso, o botão realiza a ação apropriada (instalar ou abrir o módulo). Caso o módulo não esteja instalado, ao tocar em “Instalar”, é mostrado um alerta da ação em curso (bem como possíveis erros na decorrência do processo), e a barra de progresso inferior permite o acompanhamento do download e instalação do módulo.

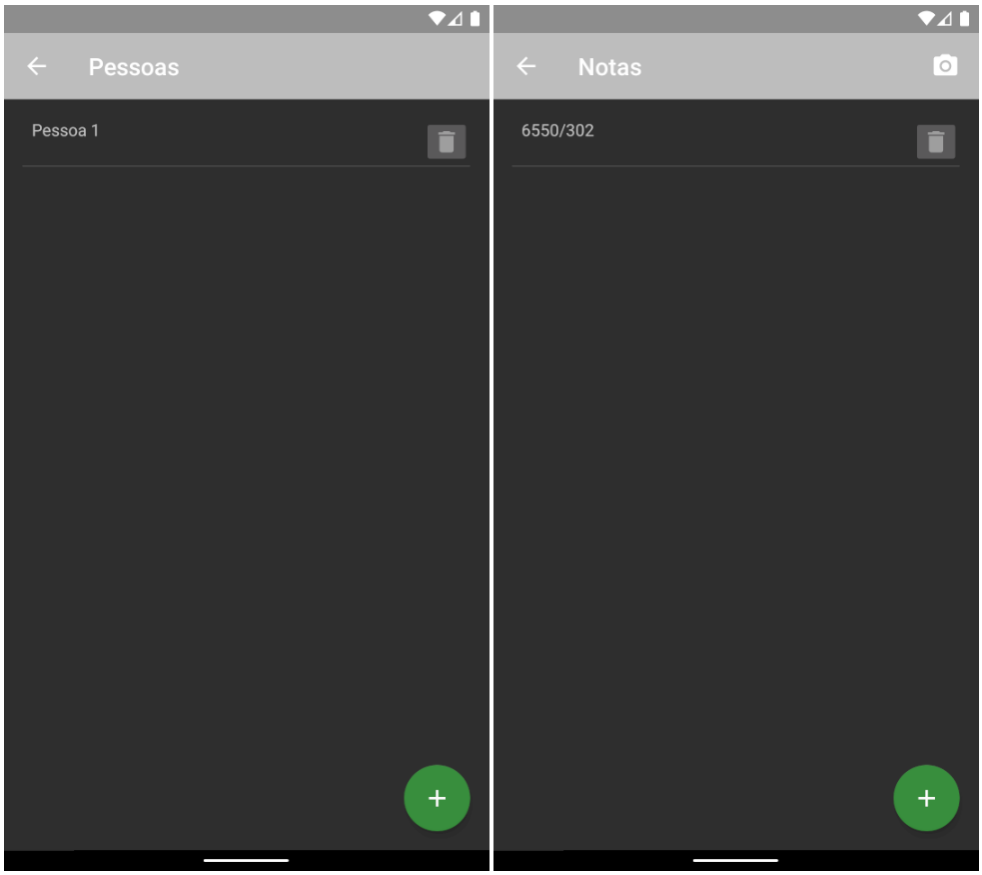
Figura 17 – tela principal do aplicativo



Fonte: Do autor (2019).

Após a finalização da tela principal, foram criadas as telas dos módulos, contendo a lista de registros e um botão para inserir um novo dado. Para editar um registro, é necessário apenas tocar no mesmo na lista. Uma caixa de diálogo para inserir ou alterar um registro foi criada e é exibida ao realizar alguma das ações citadas anteriormente (inserção ou edição). Conforme a Figura 18, as telas são basicamente iguais entre si, diferenciando-se pelo título e pela inclusão de um botão à direita na barra superior do módulo de Notas referente ao acesso à câmera do dispositivo.

Figura 18 – telas dos módulos do aplicativo



Fonte: Do autor (2019).

Um adicional do módulo de Notas é um leitor de códigos QR. O usuário é então transferido a uma tela, de onde é possível fazer a leitura de um código QR existente em uma nota. Caso a leitura seja feita com sucesso, é realizada a inclusão de uma nota no módulo automaticamente, de forma transparente ao usuário.

O processo de inclusão da nota é realizado da seguinte maneira: a API de acesso à câmera realiza a leitura de um código QR, o qual retorna um link para a Secretaria da Fazenda do estado que emitiu a nota. Em seguida, é realizado um consumo de um serviço HTTP através do link obtido, que devolve todos os dados contidos na nota. Por fim, a classe de controle do módulo de notas insere um novo registro, com os dados retornados do serviço HTTP.

5.2.4 Classes utilizadas no aplicativo

Para que os módulos de recursos dinâmicos sejam baixados e instalados corretamente, alguns métodos são necessários nas classes da aplicação. Para o presente projeto, os métodos necessários foram criados na classe principal do app. Resumidamente, recomendam-se métodos para as seguintes ações:

- Verificar o status de instalação dos módulos presentes na aplicação;
- Gerenciar a instalação do módulo de recurso dinâmico. Conforme apresentado na Figura 19, o método realiza a requisição de instalação baseado no nome do módulo (primeira linha do método). Em seguida, é iniciada a instalação do mesmo (segunda linha do método), onde também é registrado o *listener*;

Figura 19 – método de instalação de um módulo

```
private void installModule(String moduleName) {
    // Build module request for installation
    request = SplitInstallRequest.newBuilder().addModule(moduleName).build();
    manager.startInstall(request)
            .addOnSuccessListener(session -> {
                sessionId = session;
            })
            .addOnFailureListener(exception -> {
                switch (((SplitInstallException) exception).getErrorCode()) {
                    case SplitInstallErrorCode.NETWORK_ERROR:
                        // Network error (connection)
                        Toast.makeText(getApplicationContext(),
                                getString(R.string.toast_network_error),
                                Toast.LENGTH_LONG).show();
                        break;
                    case SplitInstallErrorCode.MODULE_UNAVAILABLE:
                        // Module unavailable
                        Toast.makeText(getApplicationContext(),
                                getString(R.string.toast_module_unavailable),
                                Toast.LENGTH_LONG).show();
                        break;
                    case SplitInstallErrorCode.INSUFFICIENT_STORAGE:
                        // Insufficient storage space
                        Toast.makeText(getApplicationContext(),
                                getString(R.string.toast_insufficient_storage),
                                Toast.LENGTH_LONG).show();
                        break;
                    case SplitInstallErrorCode.ACTIVE_SESSIONS_LIMIT_EXCEEDED:
                        // Active sessions exceeded (another download is in progress)
                        Toast.makeText(getApplicationContext(),
                                getString(R.string.toast_active_sessions_exceeded),
                                Toast.LENGTH_LONG).show();
                        break;
                }
                manager.cancelInstall(sessionId);
            });
}
```

Fonte: Do autor (2019).

- *Listener* para observar as ações de download e instalação do módulo, e executar eventos em caso de sucesso ou falha. Na Figura 20, a classe “Listener” sobrescreve o método existente, permitindo assim a verificação do status de download e instalação em caso de falha (primeiro bloco de execução) ou sucesso (bloco de execução *switch case*).

Figura 20 – classe para verificação de status das ações

```
class Listener implements SplitInstallStateUpdatedListener {
    @Override
    public void onStateUpdate(SplitInstallSessionState state) {
        for (String moduleName : state.moduleNames()) {
            if (state.status() == SplitInstallSessionStatus.FAILED
                    && state.errorCode() == SplitInstallErrorCode.SERVICE_DIED) {
                createConfirmationDialog(moduleName);
                return;
            }
            switch (state.status()) {
                case SplitInstallSessionStatus.REQUIRES_USER_CONFIRMATION:
                    // Ask for user confirmation (module exceeding 10MB in size)
                    try {
                        startIntentSender(state.resolutionIntent().getIntentSender(),
                                null, 0, 0, 0);
                        Toast.makeText(getApplicationContext(),
                                getString(R.string.toast_module_installing),
                                Toast.LENGTH_SHORT).show();
                    } catch (IntentSender.SendIntentException e) {
                        Log.e(moduleName, e.toString());
                    }
                    break;
                case SplitInstallSessionStatus.DOWNLOADING:
                    // Display download status
                    displayDownloadStatus(state);
                    break;
                case SplitInstallSessionStatus.INSTALLING:
                    // Display installation status
                    displayDownloadStatus(state);
                    break;
                case SplitInstallSessionStatus.INSTALLED:
                    // Module successfully installed
                    showProgressVisibility(false);
                    Toast.makeText(getApplicationContext(),
                            getString(R.string.toast_module_installed),
                            Toast.LENGTH_SHORT).show();
                    sessionId = 0;
                    finish();
                    startActivity(getIntent());
                    break;
            }
        }
    }
}
```

Fonte: Do autor (2019).

5.2.5 Bibliotecas para API REST

Devido à utilização de uma API RESTful para consumo de dados, tornou-se necessário utilizar bibliotecas de terceiros para realizar tais consumos. Para a aplicação, utilizaram-se as bibliotecas Retrofit 2 (para conexão com a API no servidor cloud) e o conversor Jackson para o Retrofit (para conversão de objetos de e para JSON). Ambos possuem fácil utilização e são populares entre os desenvolvedores de webservices, e são mantidos pela Square Inc. Para sua correta utilização, basta a realização das seguintes configurações (descritas na documentação oficial da desenvolvedora):

- Classes de modelos, para utilização nos modelos JSON;
- Implementação das interfaces que definem as operações HTTP;
- Classe *Builder*, sendo a instância que utiliza a interface e a API construtora para definir a URL utilizada pela API REST.

5.2.6 APIs de acesso à câmera

Em virtude da utilização da câmera para leitura de códigos QR no módulo de Notas, utilizou-se uma API chamada de *Mobile Vision*. Por ser uma parte do ML Kit (Machine Learning Kit), mantido pela Google Inc., ela possui extensa documentação oficial online para utilização, disponível na página Google Developers (2019).

Realizaram-se três procedimentos para incorporação da API ao projeto: a criação de uma atividade para utilização da câmera e leitura dos códigos, a inclusão da dependência ao arquivo de build (permitindo a utilização das bibliotecas pertencentes à API), e a configuração de uma classe para realizar as leituras. A Figura 21 mostra a classe utilizada para a leitura dos códigos QR, onde o primeiro bloco do método verifica a permissão de acesso à câmera e o bloco restante realiza as leituras.

Figura 21 – classe para leitura de códigos QR

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_scanner);
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
            PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions
                (this, new String[]{Manifest.permission.CAMERA}, REQUEST_CODE);
    }
    cameraView = findViewById(R.id.camera_view);
    barcodeDetector = new BarcodeDetector.Builder(this)
            .setBarcodeFormats(Barcode.QR_CODE)
            .build();
    cameraSource = new CameraSource.Builder(this, barcodeDetector)
            .setFacing(CameraSource.CAMERA_FACING_BACK)
            .setAutoFocusEnabled(true)
            .setRequestedPreviewSize(1600, 1024)
            .build();
    cameraView.getHolder().addCallback(new SurfaceHolder.Callback() {...});

    barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
        @Override
        public void release() {}
        @Override
        public void receiveDetections(Detector.Detections<Barcode> detections) {
            final SparseArray<Barcode> barcodes = detections.getDetectedItems();
            if (barcodes.size() != 0) {
                getContents(barcodes.valueAt(0).displayValue);
            }
        }
    });
}
```

Fonte: Do autor (2019).

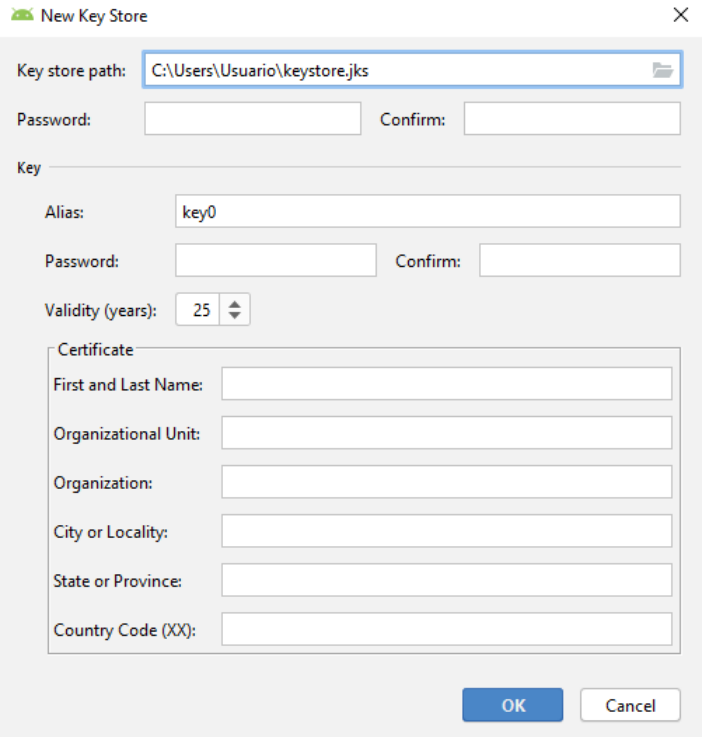
5.2.7 Envio do app à Play Store

Após finalizado o desenvolvimento do projeto, tornou-se necessário enviar o app à loja para realização dos testes de entrega e funcionamento do aplicativo. Para o envio, cria-se um arquivo de *release*, o qual é então enviado à Play Store e posteriormente publicado.

No presente projeto, o primeiro passo foi a criação do arquivo de release. Para tanto, criou-se uma chave de upload, necessária para a criação de arquivos

assinados que são enviados à loja. A Figura 22 mostra a criação da chave de upload a ser utilizada na assinatura do app. Uma recomendação existente é a de utilizar senhas diferentes para o *keystore* e para a chave de upload (que fica armazenada no keystore).

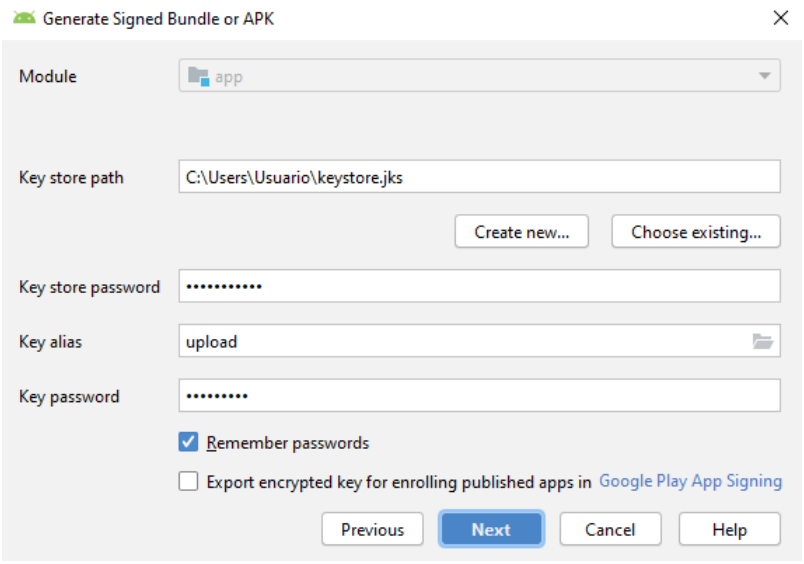
Figura 22 – tela de criação da chave de upload



Fonte: Do autor (2019).

Posteriormente, deu-se prosseguimento ao processo de criação do arquivo a ser enviado para a loja. A IDE então solicita qual o tipo de arquivo a ser criado, podendo ser um Android App Bundle ou um APK. Após a seleção do App Bundle (em conformidade com os objetivos do projeto), é requerido então o keystore, sua senha e a senha da chave de upload. Em seguida, seleciona-se qual será o local de destino para salvar o arquivo no computador e a variante de construção (debug ou release), neste caso sendo a release (versão final do aplicativo). A Figura 23 exibe a etapa de seleção do keystore existente para realização da assinatura do arquivo a ser enviado.

Figura 23 – seleção do keystore para criação do arquivo assinado



Fonte: Do autor (2019).

Com a criação do arquivo assinado contendo o aplicativo, prosseguiu-se para o envio do mesmo à Play Store. Para isso, o arquivo deve ser carregado no Play Console, acessível mediante a criação de uma conta de desenvolvedor e ao pagamento de uma taxa única de US$ 25 para publicação de aplicações.

Após o envio do arquivo e o preenchimento de informações obrigatórias para a publicação do aplicativo, há um prazo de 72 horas entre a confirmação dos dados obrigatórios e a efetiva publicação do app. Durante esse intervalo de tempo, a aplicação passa por análises detalhadas, testes automatizados e testes humanos, a fim de verificar falhas, conformidades de diretrizes, entre outros requisitos.

5.2.8 Resultados obtidos

Com a publicação do aplicativo, deu-se início então ao processo de coleta dos dados propostos nos objetivos específicos do trabalho. Para isso, a aplicação foi instalada em dois dispositivos distintos: um Xiaomi Redmi Note 7 e um Samsung Galaxy J5.

Após a instalação e abertura do app no dispositivo da Xiaomi, verificou-se que nenhum dos módulos da aplicação estavam instalados. Então, o tamanho ocupado

pela aplicação foi registrado, bem como os tamanhos de cada módulo após suas instalações. A aplicação contendo somente o módulo base possuía 17,49 MB, e os módulos “Pessoas” e “Notas” possuíam, respectivamente, 0,14 MB e 0,48 MB.

Em seguida, o processo de instalação do aplicativo e dos módulos foi efetuado da mesma forma apresentada anteriormente no dispositivo da Samsung, onde os tamanhos do módulo base, do módulo “Pessoas” e do módulo “Notas” eram de 11,38 MB, 0,16 MB e 0,51 MB, respectivamente.

Igualmente à primeira proposta descrita pelo projeto, um APK único foi gerado para verificação do espaço ocupado pelo app no formato antigo de desenvolvimento. Em ambos os dispositivos utilizados nos testes, o tamanho do aplicativo instalado ficou em torno de 25 MB.

Com isso, é possível constatar que o tamanho ocupado pelo aplicativo pode variar significativamente para cada dispositivo, uma vez que a Play Store gera um APK único e distinto para cada aparelho, considerando as configurações de divisão informadas no código-fonte da aplicação. No caso do presente projeto, o App Bundle estava configurado para dividir as densidades de tela, as bibliotecas de hardware e os idiomas existentes.

Ainda, verificou-se que o processo de solicitação da instalação dos módulos, bem como o processo de instalação em si, ocorre de forma transparente ao usuário. A única ação necessária a ser realizada pelo utilizador é a solicitação de instalação (realizada a partir do botão “Instalar”), sendo que o restante do processo é completamente gerenciado pela aplicação e pelo próprio dispositivo, em conjunto com a Play Store.

A fim de verificar o processo de atualização de um app que incorpore módulos de recursos dinâmicos, foram realizadas pequenas alterações de layout (tanto no módulo base quanto nos módulos dinâmicos). Com isso, pode-se gerar uma nova versão do aplicativo, o qual foi assinado novamente (cuja chave deve ser a mesma utilizada na versão anterior) e subsequentemente enviado à loja. A publicação da atualização do app, diferentemente do lançamento de um novo aplicativo, é feita em poucos minutos.

Após a publicação da atualização, o aplicativo foi inteiramente atualizado nos dois dispositivos utilizados anteriormente. Então, verificou-se que mesmo com a alteração dos módulos dinâmicos, os mesmos permaneceram instalados e inalterados após a atualização, enquanto as atualizações no módulo base operaram da mesma maneira de um aplicativo tradicional. Assim, caso ocorram alterações em qualquer um dos módulos adicionais (dinâmicos), torna-se necessário a remoção prévia dos mesmos, ou sua remoção e posterior reinstalação, sendo as únicas maneiras de realizar a “atualização” de um módulo dinâmico. Para conhecimento de que um determinado módulo foi atualizado, recomenda-se que tal ação seja informada ao usuário por meio das notas de atualização e versão que são apresentadas na Play Store, para que o referido módulo seja removido e reinstalado.

A desinstalação dos módulos pode ser realizada de duas maneiras: removendo-se a aplicação do dispositivo ou solicitando a remoção do módulo diretamente pela aplicação. Caso a desinstalação do módulo ocorra pelo app, torna-se necessário verificar posteriormente a lista de módulos instalados, pois o método de remoção existente executa tal ação em segundo plano, e não imediatamente ao ser executado.

6 CONSIDERAÇÕES FINAIS

O presente trabalho apresentou o desenvolvimento e a validação de uma aplicação voltada à plataforma Android, baseando-se no formato App Bundle. A proposta do projeto baseou-se em um referencial teórico e em uma pesquisa exploratória, onde houve a pesquisa de recursos e tecnologias para o desenvolvimento da aplicação citada anteriormente.

A criação do aplicativo no novo formato de desenvolvimento (apresentado no referencial) ocorreu de forma simplificada, necessitando apenas as implementações dos métodos utilizados para o download e instalação dos módulos de recursos dinâmicos, baseando-se na documentação oferecida pelo Google. Os processos de criação dos módulos, gerenciamento de dependências e assinatura do app para publicação podem ser realizados com poucos cliques, já que a IDE encarrega-se das atividades relevantes.

Da mesma forma, a incorporação de APIs aos módulos de recursos dinâmicos também se dá de forma facilitada, bastando apenas a configuração das dependências apropriadas nos arquivos de build. Com o grande número de bibliotecas disponíveis atualmente e as extensivas documentações oferecidas pelas suas mantenedoras, abre-se um leque de possibilidades a serem exploradas.

Quanto à validação da proposta, notou-se que os tamanhos de uma mesma aplicação e de seus módulos variam de acordo com o cenário em que estão inseridos. Devido à existência de configurações que permitem a divisão de recursos, o espaço ocupado por um app em um dispositivo pode variar substancialmente,

resultando sempre na economia de espaço de armazenamento em relação a um aplicativo desenvolvido no formato anterior.

Entretanto, o App Bundle possui algumas restrições, conforme visto na apresentação da primeira proposta. A incorporação de recursos que contenham ferramentas e bibliotecas autogeradas para utilização externa (como projetos criados na Unity) a módulos de recursos dinâmicos não é possível, por ser uma tecnologia em desenvolvimento. Bibliotecas utilizadas pelos recursos mencionados devem ser inseridas à aplicação em tempo de construção, e não de execução. Por isso, ocorrem erros de localização de bibliotecas ao executar os módulos que contenham tais recursos.

Por ser um recurso ainda experimental, erros são esperados, e alguns conhecidos e documentados pela mantenedora. Todavia, as funcionalidades básicas estão operacionais, e os benefícios oferecidos são perceptíveis. Com o Android App Bundle, dispositivos recebem aplicativos otimizados para suas configurações por meio do Dynamic Delivery, remove-se a necessidade de gerenciar múltiplos aplicativos para diferentes plataformas e torna-se possível incorporar novos recursos por meio de módulos de recursos dinâmicos, os quais podem ser entregues sob demanda. Por fim, há a redução do tamanho dos apps, dados os recursos citados anteriormente.

No entanto, a ênfase dos módulos de recursos dinâmicos está em recursos que não sejam constantemente atualizados, visto que os módulos permanecem instalados e inalterados nos casos em que um aplicativo que os contenham é atualizado. Assim, pode-se avaliar futuramente formas de realizar a atualização de módulos dinâmicos sem que seja necessário sua remoção e posterior reinstalação.

REFERÊNCIAS BIBLIOGRÁFICAS

Android Developers. **About Android App Bundles.** 2018. Disponível em: <https://developer.android.com/guide/app-bundle>. Acesso em: 12 abr. 2019.

Android Developers. **Add support for Dynamic Delivery.** 2018. Disponível em: <https://developer.android.com/studio/projects/dynamic-delivery>. Acesso em: 16 abr. 2019.

Android Developers. **Download modules with the Play Core Library.** 2018. Disponível em: <https://developer.android.com/guide/app-bundle/playcore>. Acesso em: 16 abr. 2019.

Android Developers. **Introdução ao Android.** 2011. Disponível em: <https://developer.android.com/guide>. Acesso em: 08 abr. 2019.

Android Developers. **Publicar o aplicativo.** 2009. Disponível em: <https://developer.android.com/studio/publish>. Acesso em: 23 abr. 2019.

BALBINOT, Ederson. **Plataforma para Distribuição de Conteúdo em Realidade Virtual ou Realidade Aumentada.** 2018. Monografia (Graduação em Engenharia de Software) – Universidade do Vale do Taquari – Univates, Lajeado, 05 de dezembro de 2018.

CASTRO, Claudio de Moura. **A Prática da Pesquisa.** 2. ed. São Paulo: Pearson Prentice Hall, 2006.

CERVO, Amado L.; BERVIAN, Pedro A.; DA SILVA, Roberto. **Metodologia Científica.** 6. ed. São Paulo: Pearson Prentice Hall, 2007.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas S.A., 2002.

Google Developers. **ML Kit.** 2019. Disponível em: <https://developers.google.com/ml-kit>. Acesso em: 11 nov. 2019.

HOEHLE, Hartmut.; VENKATESH, Viswanath. **Mobile Application Usability: Conceptualization and Instrument Development.** MIS Quarterly, Fayetteville, Arkansas, v.9, n.4, p.435-472, junho, 2015.

HOLZER, Adrian.; ONDRUS, Jan. **Mobile application market: A developer's perspective.** Elsevier, Holanda, Mobile Service Architecture and Middleware, v.28, n.1, p.22-31, fevereiro, 2011.

LECHETA, Ricardo R. **Android essencial.** São Paulo: Novatec, 2016.

MARCONI, Marina de Andrade.; LAKATOS, Eva Maria. **Fundamentos de Metodologia Científica.** 5. ed. São Paulo: Atlas S.A., 2003.

MONTEIRO, João Bosco. **Google Android: crie aplicações para celulares e tablets.** São Paulo: Casa do Código, 2012.

NEVES, Miranilde Oliveira. **A Importância da Investigação Qualitativa no Processo de Formação Continuada de Professores: Subsídios ao exercício da docência.** Revista Fundamentos, v.2, n.1, 2015. Disponível em: <https://revistas.ufpi.br/index.php/fundamentos/article/view/3723/2186>. Acesso em: 16 abr. 2019.

NOVAC, Ovidiu Constantin.; NOVAC, Mihaela.; GORDAN, Cornelia.; BERCZES, Tamas.; BUJDOSÓ, Gyöngyi. **Comparative Study of Google Android, Apple iOS and Microsoft Windows Phone Mobile Operating Systems.** IEEE, Romênia, 14th International Conference on Engineering of Modern Electric Systems (EMES), p.154-159, julho, 2017.

NUDELMAN, Greg. **Padrões de projeto para o Android: soluções de projetos de interação para desenvolvedores.** São Paulo: Novatec, 2013.

SG, Claude. APP Review Times: How fast your app will be published**. AndroidB**. 2016. Disponível em: <https://www.androidb.com/2016/08/app-review-times-for-google-play-aka-how-fast-will-your-app-be-published>. Acesso em: 24 abr. 2019.

SILVEIRA, Denise Tolfo.; GERHARDT, Tatiana Engel. **Métodos de Pesquisa.** 3. ed. Rio Grande do Sul: UFRGS, 2008.

SIUHI, Saidi.; MWAKALONGE, Judith. **Opportunities and challenges of smart mobile applications in transportation.** Elsevier, Holanda, Journal of Traffic and Transportation Engineering (English Edition), v.3, n.6, p.582-592, dezembro, 2016.

Unity Technologies. **Unity – Manual: Building apps for Android.** 2019. Disponível em:<https://docs.unity3d.com/Manual/android-BuildProcess.html>. Acesso em: 11 nov. 2019.