UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ROBERTO RAMOS

APLICATIVO PARA BUSCA E DIVULGAÇÃO DE PROMOÇÕES EM SUPERMERCADOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

ROBERTO RAMOS

APLICATIVO PARA BUSCA E DIVULGAÇÃO DE PROMOÇÕES EM SUPERMERCADOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Vinicius Pegorini.

PATO BRANCO
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

APLICATIVO PARA BUSCA E DIVULGAÇÃO DE PROMOÇÕES EM SUPERMERCADOS

POR

ROBERTO RAMOS

Este trabalho de conclusão de curso foi apresentado no dia 02 de julho de 2019, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Prof. MSc
Vinicius Pegorini

Profª MSc
Andréia Scariot Beulke

Prof MSc
Robison Cris Brito

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

A Deus por todos os bons e maus momentos que tenho em minha vida, pois são eles que nos fazem crescer e valorizar tudo e todos que temos ao nosso redor.

A minha família, em especial a minha esposa que foi a primeira pessoa a me encorajar a enfrentar este desafio, e a suportar minha ausência neste período do curso.

A todos os professores que contribuíram de alguma forma para minha formação.

RESUMO

A busca por um produto, na maioria das vezes, passa principalmente por uma consulta ao preço deste produto, o que tem uma forte influência na decisão do comprador. Em supermercados, devido à grande quantidade de produtos similares, o preço se torna um grande fator decisivo. Como a tecnologia hoje nos proporciona vários recursos, podemos utilizá-la para contribuir na escolha de produtos em que o fator relevante seja o preço. Neste trabalho, foi desenvolvido uma aplicação móvel que possibilita que o usuário realize buscas de um determinado produto e faça comparações de preços entre diferentes supermercados para este produto, proporciona a facilidade de gerar lista personalizada e a posterior visualização em mapa dos locais dos supermercados com a soma total dos valores dos produtos selecionados contidos na lista. O que diferencia esta aplicação é a possibilidade de qualquer usuário realizar inserções de promoções de qualquer produto que esteja sendo comercializado por um supermercado por meio de um *smartphone*. Para o desenvolvimento da aplicação utilizou-se algumas ferramentas como o Android Studio para codificação, o Firebase da empresa Google para armazenamento de imagens, banco de dados e autenticação, que permitiram um ganho de desempenho na implementação.

Palavras-chave: Busca por promoção. Aplicação Móvel. Consulta de preços. Divulgação.

ABSTRACT

The search for a product, most of the time, is mainly due to a query on the price of this product, which has a strong influence on the buyer's decision. In supermarkets, due to the great quantity of similar products, the price becomes a great decisive factor. As technology today provides us with many resources, we can use it to help choose products where the relevant factor is price. In this work, a mobile application was developed that allows the user to search for a particular product and compare prices between different supermarkets for this product, provides the facility to generate a personalized list and the subsequent map visualization of supermarket locations with the total sum of the values of the selected products contained in the list. What differentiates this application is the possibility of any user to make promotions insertions of any product that is being marketed by a supermarket through a smartphone. For the development of the application was used some tools such as Android Studio for coding, the Google company Firebase for image storage, database and authentication, which allowed a performance gain in implementation.

Keywords: Search by promotion. Mobile Application. Price inquiry. Disclosure.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE LISTAGEM DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

APK - *Android Package File*

APP - *Application* (Abreviação de Aplicação)

COTTAPB – Cooperativa de Trabalho dos Agentes Ambientais de Pato Branco

DEX - *Dalvik Executable*

HAL – *Hardware Abstraction Layer*

HTTP - *Hypertext Transfer Protocol*

ID - Identificador

IDE - *Integrated Development Environment*

JSON - *JavaScript Object Notation*

OHA - *Open Handset Alliance*

REST - *Representational State Transfer*

RF - Requisitos Funcionais

RNF - Requisitos Não Funcionais

RPC - *Remote Procedure Call*

SDK - *Software Development Kit*

SMS - *Short Message Service*

UI - *User Interface*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são apresentadas algumas definições sobre o contexto no qual o aplicativo desenvolvido é inserido. Também são apresentados os objetivos, a justificativa do trabalho e, por fim, a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Nos dias atuais, no momento econômico que nosso país se encontra, em que todos buscam economia em suas compras, os comerciantes procuram chamar a atenção de clientes por meio de promoções, sejam elas de maneira digital, divulgadas via redes sociais ou por meio de panfletos impressos entregues de porta em porta ou em semáforos.

Alguns municípios possuem leis que proíbem a distribuição de promoções impressas, como é o caso do município de Pato Branco, no estado do Paraná, que possui a lei municipal nº 3.536, de 17 de março de 2011 (Anexo A), a qual estabelece que "fica também proibida a distribuição de panfletos para ocupantes de automóveis aguardando o fluxo, parados em semáforos, rotatórias e em congestionamentos, em via pública do sistema viário do Município Pato Branco" (PREFEITURA...2011).

De acordo com a Prefeitura Municipal de Pato Branco, "12 toneladas de resíduos recicláveis são recolhidos diariamente pela Cooperativa de Trabalho dos Agentes Ambientais de Pato Branco (COTAAPB)" (PREFEITURA...2018).

Neste contexto visualizou-se a oportunidade de colaborar com a redução na geração de lixo, em parte produzida por panfletagem em papel, com a elaboração de um aplicativo móvel em que o usuário possa buscar e inserir promoções de produtos desejados mediante seu aparelho telefônico, seja ele um cliente ou o próprio comerciante.

O aplicativo terá seu uso, a princípio, destinado a produtos vendidos em supermercados. Esse aplicativo foi desenvolvido na plataforma Android, utilizando para o desenvolvimento, a ferramenta Android Studio, também foram utilizadas ferramentas do Google, Realtime Database, o Cloud Storage e o Authentication, as quais fazem parte da plataforma Firebase.

A intenção é colaborar com o comércio local, estimulando a concorrência entre supermercados e colaborando com o usuário na economia em suas compras.

1.2 OBJETIVOS

A seguir estão o objetivo geral e os objetivos específicos definidos para este trabalho.

1.2.1 Objetivo Geral

Desenvolver um aplicativo móvel, no qual usuários poderão realizar buscas ou inserções de promoções de produtos de supermercados. Oferecer ao usuário opções de busca e inserção de promoções no setor de supermercados, contribuindo com a concorrência entre os mesmos.

1.2.2 Objetivos Específicos

- Possibilitar a inserção de promoções.
- Possibilitar a busca de promoções por região.
- Possibilitar a busca de preços específicos por produto.
- Manter os preços atualizados.

1.3 JUSTIFICATIVA

Em conversas informais com clientes em supermercados, buscou-se conhecer de que maneira eles buscam informações sobre os preços dos produtos no supermercado no qual estão realizando as compras, e se buscam realizar suas compras baseados no preço ou na preferência por determinado supermercado.

A maioria das pessoas informou que realizam compras baseadas nos preços dos produtos existentes no supermercado e ficam sabendo desses preços por meio de panfletos em papel distribuídos de porta em porta ou por divulgações em redes sociais.

Com base nestas informações, na praticidade de uso e mobilidade que o *smartphone* propicia, observou-se a possibilidade de elaboração de um aplicativo para a distribuição de promoções visualizadas pelos clientes quando estão em compras.

A plataforma escolhida foi a Android por ser a mais utilizada em *smartphones* à venda no Brasil segundo a StatCounter, empresa de análise da web (STATCOUNTER, 2018).

A intenção não é definir preços e nem vender produtos, apenas compartilhar, caso desejado pelo cliente, um valor de produto à escolha do cliente.

> Entender e adaptar-se à motivação e ao comportamento do consumidor não é uma opção - é a necessidade absoluta para a sobrevivência competitiva. (SILVA, MERLO E NAGANO[1], 2012 apud ENGEL; BLACKEWELL E MINIARD, 2000).

O aplicativo não requer um banco de dados local, mas necessitará que os dados sejam sincronizados em tempo real. A partir desta necessidade, o banco de dados utilizado para implementação foi o Realtime Database da plataforma Firebase, de propriedade da Google.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. Neste, que é o primeiro capítulo, foram apresentadas as considerações iniciais sobre o contexto em que será inserido o sistema que foi desenvolvido, os seus objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico sobre as ferramentas que foram utilizadas. No Capítulo 3 estão as tecnologias e suas ferramentas que foram utilizadas na construção do sistema. No Capítulo 4 estão os resultados do trabalho desenvolvido. No Capítulo 5 está a conclusão do trabalho. Por fim, estão as referências que foram utilizadas no trabalho.

[1] SILVA, MERLO E NAGANO, 2012 Uma análise dos principais elementos influenciadores da tomada de decisão de compra de produtos de marca própria de supermercados. *REAd. Revista Eletrônica de Administração (Porto Alegre)*, *18*(1), 97-129. https://dx.doi.org/10.1590/S1413-23112012000100004

2 REFERENCIAL TEÓRICO

Este capítulo apresenta uma descrição do sistema Android e algumas funcionalidades da plataforma Firebase, como o Firebase Authentication, Cloud Storage e Realtime Database por serem fundamentais para o desenvolvimento do aplicativo desenvolvido como resultado desse trabalho.

2.1 ANDROID

A plataforma definida para o desenvolvimento do projeto foi a Android, por ser robusta e de fácil utilização e aprendizado. É uma plataforma de desenvolvimento e execução de programas em dispositivos móveis como *smartphones* e baseado no kernel do Linux, que é responsável pelo gerenciamento da memória, dos processos, das *threads*, da segurança dos arquivos e pastas, além de redes e *drivers* (LECHETA, 2015).

Android Inc. era o nome da empresa que desenvolveu a plataforma, localizada em Palo Alto na Califórnia nos Estados Unidos que inicialmente desenvolvia somente aplicativos para celulares, ela foi adquirida pela Google em agosto de 2005.

Quando a plataforma Android nasceu, foi formada uma aliança de várias empresas do setor, a Open Handset Alliance (OHA), para fortalecer o crescimento da plataforma (BRITO, 2017).

De acordo com a empresa de pesquisas Gatner, o número de *smartphone*s que utilizam o Android superou a marca de um bilhão vendidos em 2018, totalizando uma fatia de 85,9% do mercado mundial do setor.

O Quadro 1 apresenta as versões do Android disponíveis até o momento.

Quadro 1 - Versões Android

Versão	Codinome	Data de lançamento	Nível API
9.0	*Pie*	06 de agosto de 2018	28
8.1	*Oreo*	05 de dezembro de 2017	27
8.0	*Oreo*	21 de agosto de 2017	26
7.1	*Nougat*	5 de dezembro de 2016	25

7.0		22 de agosto de 2016	24
6.0	*Marshmallow*	05 de Outubro de 2015	23
5.1	*Lollipop*	10 de março de 2015	22
5.0-5.0.2		12 de novembro de 2014	21
4.4W-4.4W.2	*Android Wear* (*KitKat*)	18 de março de 2014	20
4.4	*KitKat*	31 de outubro de 2013	19
4.3	*Jelly Bean*	24 de julho de 2013	18
4.2.x		13 de novembro de 2012	17
4.1.x		9 de julho de 2012	16
4.0.3–4.0.4	*Ice Cream Sandwich*	16 de dezembro de 2011	15
4.0.1-4.0.2		19 de outubro de 2011	14
3.2-3.2.6	*Honeycomb*	15 de julho de 2011	13
3.1		10 de maio de 2011	12
3.0		22 de fevereiro de 2011	11
2.3.3–2.3.7	*Gingerbread*	9 de fevereiro de 2011	10
2.3		06 de dezembro de 2010	9
2.2-2.2.3	*Froyo*	20 de maio de 2010	8

Fonte: Android Developer (2018).

No Android cada aplicativo dispara um processo no sistema operacional (LECHETA, 2015), sejam para exibir uma tela ou para executar um processo em segundo plano. É possível

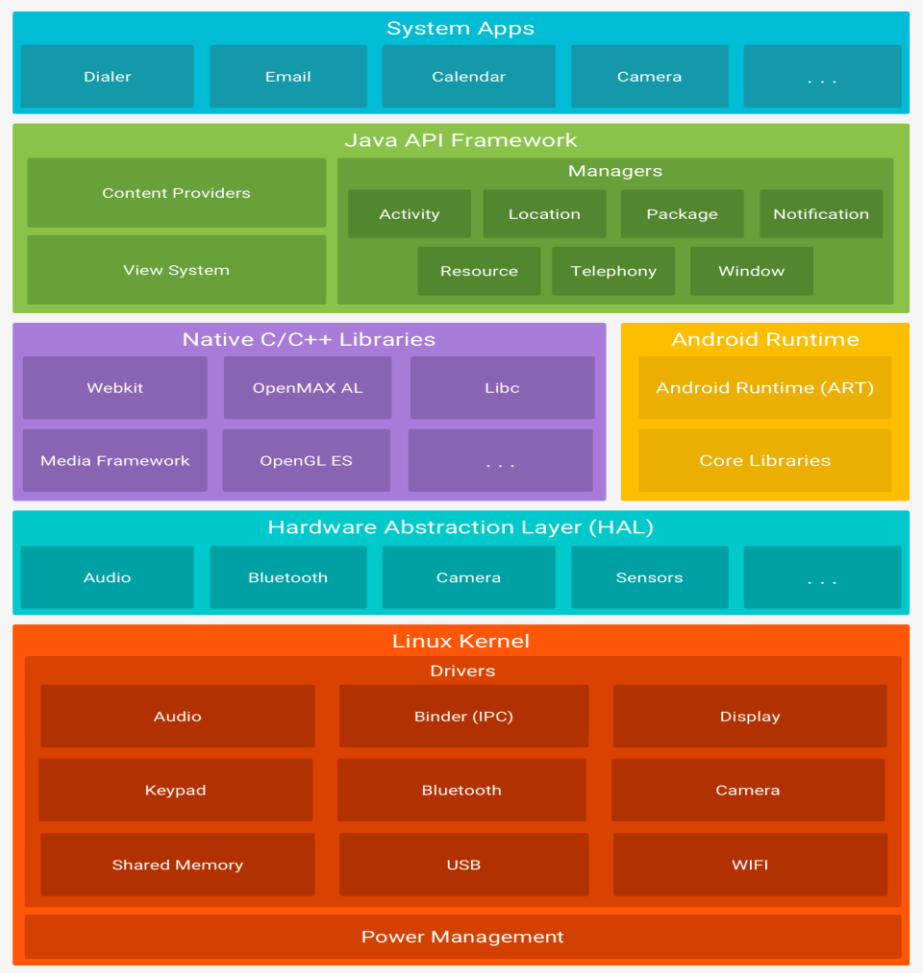
a execução de vários processos simultaneamente, o responsável por permitir e gerenciar a memória do sistema operacional é o kernel.

Caso o uso de memória esteja comprometido o sistema operacional pode encerrar algum processo e reinicia-lo posteriormente se necessário.

Para cada aplicação instalada no celular é criado um usuário no sistema operacional para ter acesso a sua estrutura de diretórios. Dessa maneira, nenhum outro usuário pode ter acesso a essa aplicação (LECHETA, 2015).

O Android é uma pilha de software com base em Linux de código aberto criada para diversos dispositivos e fatores de forma. O diagrama apresentado na Figura 1 mostra a maioria dos componentes da plataforma Android (ANDROID DEVELOPERS, 2018).

Figura 1 - Pilha de software do Android.



Fonte: Android Developers (2018).

De acordo com o que está representado na Figura 1 (ANDROID DEVELOPERS, 2018):

a) Linux Kernel: Permite que o Android aproveite os principais recursos de segurança e as funcionalidades de baixo nível do sistema operacional, e que os fabricantes dos dispositivos desenvolvam drivers de hardware para um kernel já conhecido.
b) Hardware Abstraction Layer (HAL): Fornece interfaces padrão do hardware do dispositivo para a estrutura da Java *Application Programming Interface* API de maior nível. Consiste em módulos de biblioteca, que implementam uma interface para um tipo específico de componente de hardware. Quando uma *framework* API faz uma chamada para acessar o hardware do dispositivo, o sistema Android carrega o módulo da biblioteca para este componente de hardware.
c) Android Runtime: Projetado para executar várias máquinas virtuais em dispositivos de baixa memória executando arquivos *Dalvik Executable* (DEX), um formato de *bytecode* projetado especialmente para Android, otimizado para oferecer consumo mínimo de memória.
d) Native C/C++ Libraries: Conjunto de bibliotecas C/C++ que são fornecidas ao Java Framework APIs para expor a funcionalidade de algumas dessas bibliotecas nativas aos aplicativos. Inclui base de dados SQLite, engine gráfica 2D e 3D, codecs de mídia, gerenciadores de interface entre outras.
e) Java API Framework: Conjunto completo de recursos do sistema operacional Android. Formam os blocos de programação necessários para criar os aplicativos Android simplificando a reutilização de componentes e serviços de sistema modulares e principais, possui um sistema de visualização, gerenciadores de recursos, de notificação e atividade, além de provedores de conteúdo.
f) System Apps: Os aplicativos inclusos na plataforma não têm status especial entre os aplicativos que o usuário opta por instalar. Portanto, um aplicativo terceirizado pode se tornar o navegador da Web, o aplicativo de envio de *Short Message Service* (SMS) ou até mesmo o teclado padrão do usuário Os aplicativos do sistema funcionam como aplicativos para os usuários e fornecem capacidades principais que os desenvolvedores podem acessar pelos próprios aplicativos.

2.2 FIREBASE

É um serviço da empresa Google que oferece infraestrutura e ferramentas necessárias para quem deseja desenvolver, testar e lançar aplicativos de alta qualidade e ampliar a base de usuários (FIREBASE, 2018).

Possui opções para quem deseja criar um aplicativo, melhorar a qualidade do aplicativo ou expandir seus negócios.

Neste trabalho foram utilizadas somente três das funcionalidades da plataforma, o Firebase Authentication, o Realtime Database e o Cloud Storage, as quais são apresentadas nas seções 2.2.1, 2.2.2 e 2.2.3, respectivamente. Todas as funcionalidades possuem planos com capacidades e custos diferenciados, mas para a realização deste trabalho optou-se pelo plano Spark, o qual é gratuito e possui bons limites de capacidade de armazenamento e conexões para o que se pretende neste projeto.

2.2.1 Firebase Authentication

Reconhecer a identidade do usuário se tornou cada vez mais necessário, pois permite que a aplicação salve os dados do usuário na nuvem com segurança e forneça experiência personalizada em todos os dispositivos do usuário.

De acordo com a empresa Google, o Firebase Authentication fornece serviços de *back-end*, *Software Development Kit* (SDK) fáceis de usar e bibliotecas prontas para autenticar usuários. Ele oferece suporte à autenticação por meio de senhas, números de telefone e provedores de identidade federados como Google, Facebook, Twitter e outros (FIREBASE, 2018).

Como principais recursos destaca-se (FIREBASE, 2018):

a) FirebaseUI Auth: solução de autenticação com *drop-in* que lida com os fluxos de *User Interface* (UI) para fazer autenticação dos usuários que podem utilizar números de telefones, *e-mail*, senhas, e provedores de identidade como Google e Facebook, o que otimiza a conversão de meios de autenticação e inscrições para a aplicação. Pode ser personalizada com facilidade para se ajustar ao estilo visual do aplicativo. E possui seu código aberto.
b) SDK do Firebase Authentication: pode dividir o meio de acesso para que o usuário utilize somente *e-mail* e senhas, somente números de telefone, somente por

integração de provedor de identidade federado, ou ainda é possível personalizar o sistema de autenticação.

2.2.2 - Realtime Database

O Realtime Database é um banco de dados hospedado na nuvem no qual os dados são armazenados como *JavaScript Object Notation* (JSON) e sincronizados em tempo real com todos os clientes conectados. Quando o usuário cria apps em plataformas cruzadas com SDKs para iOS, Android e JavaScript, todos os clientes compartilham uma instância do Realtime Database e recebem automaticamente atualizações com os dados mais recentes. (FIREBASE, 2019).

Os principais recursos, de acordo com o fabricante são (FIREBASE, 2019):

a) Em tempo real: ao invés de solicitações *Hypertext Transfer Protocol* (HTTP), o Firebase Realtime Database usa a sincronização de dados. Sempre que os dados são alterados, todos os dispositivos conectados recebem essa atualização em milissegundos.
b) Off-line: Os apps do Firebase permanecem responsivos mesmo off-line, pois o SDK do Firebase Realtime Database mantém seus dados em cartão de memória.
c) Acessível em dispositivos clientes: Não necessita de servidor de aplicativos, pode ser acessado diretamente de um dispositivo móvel ou navegador Web.

O Realtime Database fornece uma linguagem de regras flexíveis baseadas em expressão, denominadas regras de segurança, para definir como os dados são estruturados e quando podem ser lidos e gravados. Por meio da integração com o Firebase Authentication, os desenvolvedores podem definir quem tem acesso, a quais dados e como esses dados podem ser acessados. (FIREBASE, 2019)

2.2.3 Cloud Storage

O Cloud Storage para Firebase é um serviço de armazenamento de objetos na nuvem. Com os SDKs do Firebase para Cloud Storage, é possível fazer o *upload* e o *download* de arquivos nos aplicativos Firebase, independentemente da qualidade da rede (FIREBASE 2018).

Os principais recursos, de acordo com o fabricante são (FIREBASE, 2018):

a) Operações confiáveis: com os SDKs do Firebase para Cloud Storage, uploads e downloads são feitos independentemente da qualidade da rede. Os uploads e downloads são mais confiáveis, o que significa que eles são reiniciados no ponto em que foram interrompidos, poupando tempo e largura de banda dos usuários.
b) Segurança potente: os SDKs do Firebase para Cloud Storage estão integrados ao Firebase Authentication para fornecer uma autenticação simples e intuitiva para os desenvolvedores. Usando modelo de segurança declarativa, o acesso é concedido com base no nome, tamanho, tipo de conteúdo e outros metadados do arquivo.
c) Alta escalabilidade: foi projetado para suportar a escala de exabyte.

3 MATERIAIS E MÉTODO

No Quadro 2 é apresentada uma relação dos materiais utilizados neste trabalho e o método utilizado para modelagem do trabalho.

3.1 MATERIAIS

Quadro 2 - Lista de ferramentas e tecnologias.

Ferramenta / Tecnologia	Versão	Finalidade	Link
Android Studio	3.4	IDE	https://developer.android.com/studio/
Visual Paradigm Community Edition	15.1	Modelagem UML (Casos de uso e diagrama de classes)	https://www.visual-paradigm.com/
Java	8	Linguagem de Programação	https://www.java.com/pt_BR/download/
Firebase Authentication	16.2.0	Serviço de autenticação em várias plataformas	https://firebase.google.com/products/auth/
Realtime Database	16.1.0	Banco de Dados NoSQL em nuvem	https://firebase.google.com/products/realtime-database/
Cloud Storage	16.1.0	Armazenamento em nuvem	https://firebase.google.com/products/storage/
Zxing	3.2.1	Biblioteca para leitura de códigos de barras	https://github.com/zxing/zxing

Fonte: Autoria Própria.

3.1.1 Android Studio

Até algum tempo atrás o Eclipse era o ambiente de desenvolvimento de aplicativos Android mais utilizado para esta finalidade, devido a facilidade de gerenciamento de *plug-ins*. Neste momento uma ferramenta que ganha cada vez mais adeptos é o Android Studio (BRITO, 2017).

Este ambiente, o Android Studio é específico para desenvolvimento de aplicativos que utilizam a plataforma Android. O seu desenvolvimento foi baseado na IDE InteliJ Community Version, ele se mostra mais didático que o Eclipse, pois necessita de menos configurações o que aumenta a produtividade no desenvolvimento de software (BRITO, 2017).

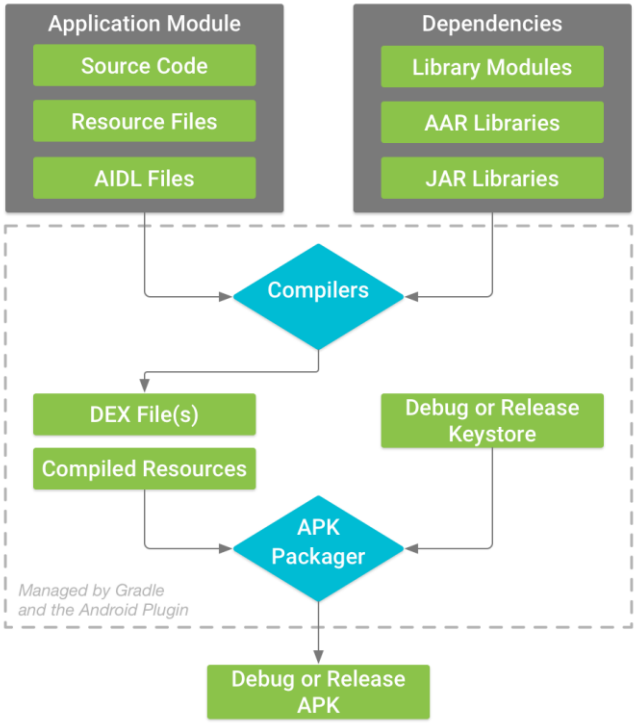
Alguns diferenciais importantes se comparado ao Eclipse (LECHETA, 2015):

- Editor visual mais fluido e com mais opções.
- Sistema de build mais moderno baseado em Gradle (*gradle.org*).
- Diversas utilidades e facilidades ao desenvolver para Android, sendo muito integrado ao Android SDK.
- Templates de projetos para *smartphones*, *tablets*, relógios etc.
- Atualizações e melhorias frequentes.

O que também diferencia o Android Studio do Eclipse é a compilação dos projetos. Enquanto no Eclipse a compilação acontece de maneira clássica Java, no Android Studio é usado o Gradle (gradle.org) que é um kit de ferramentas de compilação avançado, que automatiza e gerencia o processo de compilação, permitindo personalizar e flexibilizar as configurações de compilação.

O processo de compilação envolve processos e ferramentas para converterem o projeto em um pacote de aplicativo Android, como pode ser observado na Figura 2.

Figura 2 - Processo de compilação Android Studio.

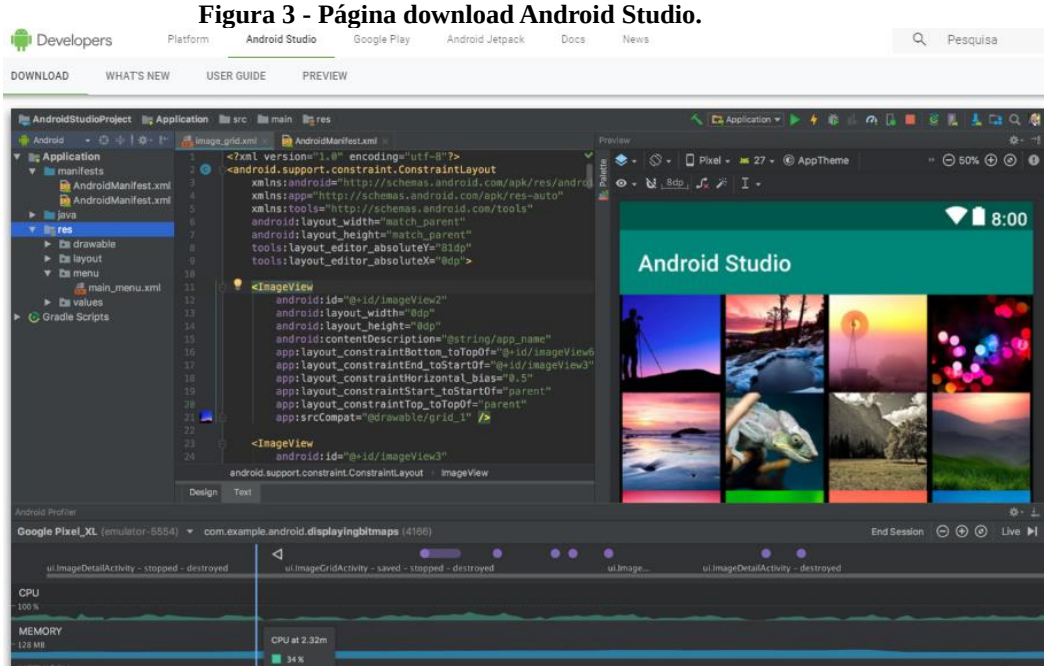


Fonte: Android Developers (2018).

Conforme o fabricante do Android Studio (ANDROID DEVELOPERS, 2018), o processo de compilação de um módulo de aplicativo Android segue as seguintes etapas gerais:

1. Os compiladores convertem seu código-fonte em arquivos DEX, que incluem o *bytecode* que é executado em dispositivos Android e todo o restante em recursos compilados.
2. O *Android Package File* (APK) Packager combina os arquivos DEX e os recursos compilados em um só APK. No entanto, para que seu aplicativo possa ser instalado e implantado em um dispositivo Android, o APK deve ser assinado.
3. O APK Packager assina o APK usando o repositório de chaves de lançamento ou de depuração:
    a) Se você estiver compilando uma versão de depuração do seu aplicativo, ou seja, um aplicativo apenas para teste e geração de perfis, o Packager o assina com o repositório de chaves de depuração. O Android Studio automaticamente configura novos projetos com um repositório de chaves de depuração.
    b) Se você estiver compilando uma versão de lançamento do seu aplicativo que pretende lançar externamente, o Packager o assina usando o repositório de chaves de lançamento.
4. Antes de gerar seu APK final, o Packager usa a ferramenta zipalign para otimizar seu aplicativo para usar menos memória quando ele for executado em um dispositivo.

Figura 3 - Página download Android Studio.

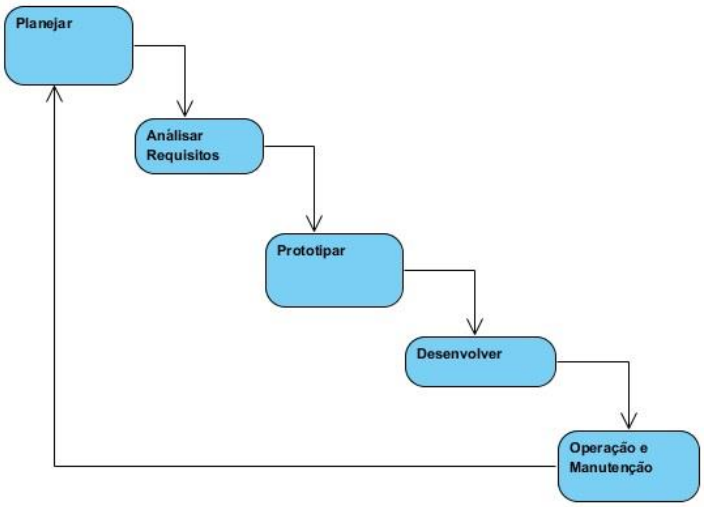


Fonte - https://developer.android.com/studio/.

3.2 MÉTODO

O método se resume em levantamento de requisitos, modelagem e implementação, o ciclo de vida adotado para o planejamento foi o modelo em cascata. A Figura 4 representa este ciclo.

Figura 4 - Ciclo de vida.



Fonte: Autoria própria

Os requisitos necessários para o projeto foram levantados por meio de pesquisas sobre aplicativos com finalidades próximas no Google Play (2018), pesquisas em livros como Brito(2017) e Lecheta (2015), e em conversas com professores da UTFPR.

Os requisitos foram divididos em funcionais, que definem o que o aplicativo fará por meio das funcionalidades que foram implementadas, e não funcionais, que definem o desempenho e as restrições do aplicativo.

Os requisitos funcionais foram organizados em casos de uso tendo atores vinculados a eles.

Com a análise dos requisitos foram definidas as funcionalidades implementadas no aplicativo.

A etapa de desenvolvimento foi realizada em paralelo com a de testes, pois, na medida em que cada tela estava sendo elaborada, era necessário realizar testes para verificar a eficácia da mesma. Um exemplo foi a primeira tela desenvolvida, a tela de autenticação, após seu layout ser definido e codificado iniciou-se a primeira etapa de testes, que foram os testes de autenticação via Facebook e autenticação própria, via Firebase. Para realizar estes testes de

autenticação própria, um usuário de teste foi inserido diretamente no console Firebase, e para testes de autenticação via Facebook utilizou-se a conta pessoal do desenvolvedor do projeto.

Em seguida foi definido o layout e os testes funcionais. Com a finalização da tela de autenticação iniciou-se a elaboração das telas de cadastro nesta sequência: de usuários, de supermercado, de produtos, de promoções, a tela principal com a possibilidade de uso de filtros para que as promoções sejam visualizadas conforme definição personalizada pelo usuário e, para finalizar, a tela que mostra um mapa da região os supermercados onde estão localizadas as promoções filtradas pelo usuário.

Para realização dos testes, optou-se por trabalhar com o aplicativo em funcionamento em um *smatphone* real, que será detalhado posteriormente, e com os resultados dos testes visualizados via console no Firebase. Como exemplo será detalhado o cadastro de produto: utilizando um produto real, no exemplo uma barra de chocolate, inicia-se o teste utilizando a câmera do *smartphone*, a qual realiza a leitura do código de barras impresso na embalagem do produto, retornando a numeração, em seguida, dados do produto como nome, embalagem e conteúdo são inseridos em campos específicos e, após finalizadas as inserções das informações, o botão salvar é pressionado finalizando o processo no *smartphone*, e para validar o teste, o console do Firebase é verificado para que se tenha certeza que os dados foram salvos corretamente.

4 RESULTADO

Neste capítulo será apresentado o resultado deste trabalho que é a implementação de um aplicativo para busca, inserção e divulgação de promoções de produtos vendidos em supermercados.

4.1 ESCOPO DO SISTEMA

O intuito da elaboração do aplicativo é fornecer ao usuário a possibilidade de que possam ser realizadas consultas de promoções de determinado produto e do local onde este produto se encontra, se o usuário encontrar o mesmo produto em outro local com preço menor, e caso desejar, ele poderá realizar a inserção de uma nova promoção, mantendo as existentes para que possam ser comparadas por outros usuários, contribuindo, assim, com a divulgação do preço praticado no supermercado.

Também deverá ser possível ao supermercado divulgar promoções realizadas por ele, sendo necessário a diferenciação de usuário por meio de perfis, sendo eles o consumidor, o supermercado e o administrador, responsável por manter o sistema.

4.2 MODELAGEM DO SISTEMA

Nesta seção será apresentada a modelagem do sistema e sua estrutura.

4.2.1 Requisitos Funcionais e Não Funcionais

O Quadro 3 apresenta os requisitos funcionais (RF).

Quadro 3 - Requisitos Funcionais.

Identificação	Nome	Descrição
RF01	Efetuar login	Permitir entrada de cliente, supermercado ou administrador
RF02	Cadastrar supermercado	Inserção de dados para supermercado, como nome, endereço, telefone, latitude, longitude e cadastro de um usuário para o supermercado.
RF03	Cadastrar usuário	Cadastro de usuário, com dados como nome, login e senha
RF04	Cadastrar produtos	Cadastro de produtos para alimentação do banco de dados, com código de barras, e descrição do produto como nome, marca, embalagem e conteúdo.
RF05	Inserir promoções	Pesquisa um produto por meio de seu código de barras, em seguida informar o preço promocional para esse produto.
RF06	Pesquisar promoções	Pesquisar produto por meio de seu código de barras ou por meio de filtros disponibilizados.
RF07	Pesquisar supermercado	Busca por supermercado, onde são mostradas as promoções para determinado supermercado.
RF08	Cadastrar departamento	Cadastro de departamentos para melhor divisão interna do supermercado.

Fonte: Autoria própria

O Quadro 4 mostra os requisitos não funcionais (RNF) que foram definidos para o aplicativo. Esses requisitos definem as restrições do sistema, as regras de negócios entre outros.

Quadro 4 - Requisitos Não Funcionais.

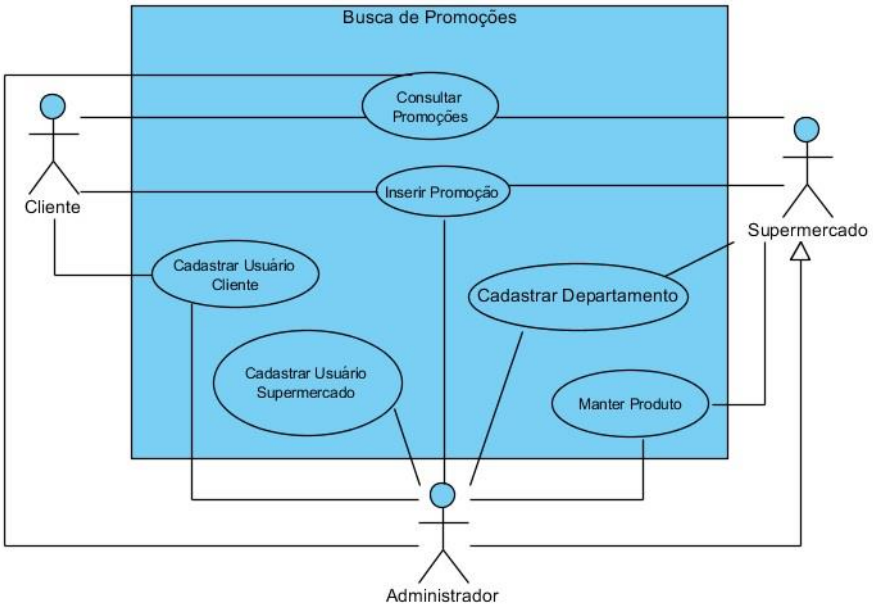
Identificação	Nome	Descrição
RNF01	Acesso ao sistema	Permitido acesso somente por login, seja por usuário e senha cadastrado no sistema ou seja vinculado à rede social.
RNF02	Cadastro de produtos	Permitir somente que o supermercado ou administrador possa realizar o cadastro de produtos vinculados ao código de barras.
RNF03	Atualização de produtos	Busca pelo código de barras, após encontrado retorna o produto com campo para atualização. Permitido somente para administrador de supermercado cadastrado
RNF04	Pesquisa de promoção	Menu com categorias e subcategorias ou busca direta por nome ou código de barras.
RNF05	Pesquisa de produto	Caso um produto pesquisado não esteja cadastrado no banco de dados, um alerta é gerado para o administrador efetuar atualização.
RNF06	Pesquisa por departamento	Menu com lista dos departamentos cadastrados.
RNF07	Pesquisa por supermercado	Menu com lista dos locais cadastrados onde, após realizar a opção do local, mostre os produtos cadastrados neste supermercado.
RNF08	Exibição em mapa	Um mapa é gerado com o valor total dos produtos de cada supermercado cadastrado.
RNF09	Edição perfil	Para cliente com cadastro no aplicativo é permitido edição de nome, senha e foto do perfil, para usuário que utiliza rede social não é permitido nenhum tipo de edição, para supermercado permitido alteração de senha e foto do perfil. Para administrador permissão total.

Fonte: Autoria própria

4.2.2 Casos de uso

O diagrama de casos de uso apresentado na Figura 5 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: cliente, supermercado e administrador. O administrador é o responsável pelo cadastro de clientes ou supermercados, e também de produtos, além de controle total das funcionalidades do aplicativo. O supermercado terá acesso para poder realizar operações de inclusão, consulta e edição dos produtos, além de inserções de promoções com possibilidade de alteração de datas do período da promoção. O cliente, além de consultar promoções, poderá realizar a inserção de promoções, as quais possuem validade diária, devido à possibilidade do supermercado alterar o valor do produto sem aviso prévio.

Figura 5 - Diagrama de casos de uso.



Fonte: Autoria própria

O caso de uso cadastrar usuário cliente é apresentado no Quadro 5.

Quadro 5 - Caso de uso cadastrar usuário cliente.

Caso de Uso: Cadastrar cliente
Atores: Cliente e Administrador
Pré-condições: para cliente não há, para administrador necessário estar autenticado
Pós-condições: Cliente é validado pelo sistema com possibilidade de acesso restrito ao sistema.
Fluxo Principal: 1. O cliente informa um login (*e-mail* válido) e uma senha 2. O sistema verifica se é possível cadastrar (login e senha válidos, ou não repetidos) 3. O sistema registra o início de uma sessão
Fluxo Alternativo: No passo 2, caso o login e senha forem inválidos ou repetidos, será exibida uma mensagem e o sistema retorna ao passo 1
Fluxo Exceção: Caso não seja atendida condição 1, o sistema exibe uma mensagem e não permite cadastro.

Fonte: Autoria própria

O caso de uso cadastrar departamento é apresentado no Quadro 6.

Quadro 6 - Caso de uso cadastrar departamento.

Caso de Uso: Cadastrar departamento
Atores: Administrador e Supermercado
Pré-condições: Estar autenticado no sistema.
Pós-condições: Administrador ou supermercado fará a inserção de um departamento ainda não cadastrado.
Fluxo Principal:

1. [IN] O nome do departamento é digitado. 2. [OUT] O sistema verifica se o departamento inserido ainda não consta no banco de dados. 3. [OUT] O sistema exibe uma mensagem de sucesso.
Fluxo Alternativo: Caso o departamento já esteja cadastrado no banco de dados o nome deste departamento será visualizado no campo de digitação e uma mensagem será exibida.
Fluxo Exceção: Caso não seja atendida condição 1 sistema exibe mensagem e não permite a inserção.

O caso de uso consultar promoções é apresentado no Quadro 7.

Quadro 7 - Caso de uso consultar promoção

Caso de Uso: Consultar promoções
Atores: Cliente, Supermercado e Administrador
Pré-condições: Necessário ter efetuado autenticação no sistema.
Pós-condições: Realiza consulta de promoção do produto desejado.
Fluxo Principal: 1. [OUT] O sistema direciona para tela principal, onde são exibidas as principais promoções e onde existe menu com filtros que podem auxiliar a refinar a busca.
Fluxo Alternativo: Caso promoção não esteja cadastrada, usuário pode realizar um novo cadastro de promoção.
Fluxo Exceção: Caso não seja atendida condição 1 sistema exibe mensagem e não permite acesso

Fonte: Autoria própria

O caso de uso inserir supermercado é apresentado no Quadro 8.

Quadro 8 - Caso de uso inserir supermercado.

Caso de Uso: Inserir supermercado
Atores: Administrador
Pré-condições: Necessário ter efetuado autenticação no sistema.
Pós-condições: Realiza inserção de dados referentes ao supermercado, e gera um usuário para o mesmo.
Fluxo Principal: 1. [OUT] O sistema direciona para tela de cadastro do supermercado.
Fluxo Alternativo: Mensagem de exceção.
Fluxo Exceção: Caso não seja atendida condição 1, o sistema exibe uma mensagem e não permite que o cadastro seja realizado.

Fonte: Autoria própria

O caso de uso manter produto é apresentado no Quadro 9.

Quadro 9 - Caso de uso manter produto.

Caso de Uso: Manter produto
Atores: Administrador e Supermercado
Pré-condições: Necessário autenticação de usuário.
Pós-condições: Realiza inserção de dados referentes ao produto.

Fluxo Principal: 1. [OUT] O sistema direciona para tela de cadastro do produto.
Fluxo Alternativo: Caso produto não esteja cadastrado, uma solicitação deve ser enviada ao administrador.
Fluxo Exceção: Caso não seja atendida condição 1, o sistema exibe uma mensagem e não permite que o cadastro seja realizado.

Fonte: Autoria própria

O caso de uso inserir promoção é apresentado no Quadro 10.

Quadro 10 - Caso de uso inserir preço.

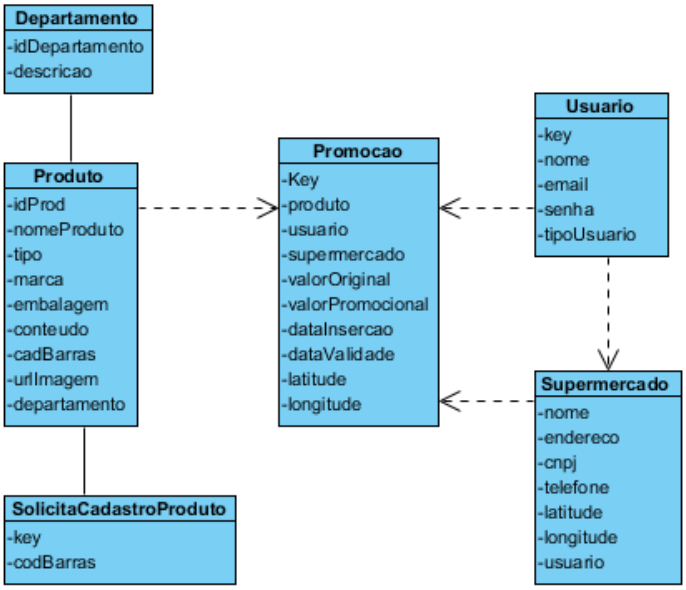
Caso de Uso: Inserir promoção
Atores: Cliente, Supermercado e Administrador.
Pré-condições: Necessário ter efetuado login, e produto cadastrado no sistema.
Pós-condições: Realiza inserção de promoção do produto desejado.
Fluxo Principal: 1. [OUT] O sistema direciona para tela de inserção de promoção 2. [IN] Solicitado inserção de um código de barras, utilizando a câmera do celular ou por digitação, onde é retornado as características do produto cadastrado. 3. [IN] Caso o usuário seja um supermercado é permitido alteração de datas, e valores, caso seja cliente somente os valores podem ser alterados.
Fluxo Alternativo: Caso o produto não esteja cadastrado, um registro é efetuado no banco de dados solicitando o cadastro deste produto.
Fluxo Exceção: Caso não seja atendida condição 1 sistema exibe mensagem e retorna a tela inicial

Fonte: Autoria própria

4.2.3 Diagrama de Classes

O diagrama de classes está apresentado na Figura 6. A principal classe é a classe Promoção, a qual é responsável em armazenar os produtos em promoção, o usuário, o supermercado e as coordenadas do local onde foi anunciada a promoção, as coordenadas se fazem necessárias devido à possibilidade de uma promoção ser inserida em um supermercado que não esteja cadastrado no sistema ou para que as mesmas sejam comparadas com algum supermercado já cadastrado. A classe Produto armazena os dados referentes ao produto, seguindo o padrão de código de barras nacional. A classe Supermercado armazena dados referente ao cadastro do supermercado. A classe Departamento armazena os departamentos que o supermercado necessitar. A classe Usuário armazena dados referente ao cadastro do usuário necessários para acesso ao sistema. A classe SolicitaCadastroProduto é responsável por armazenar o código de barras de um produto que não esteja cadastrado no sistema para que ele seja cadastrado posteriormente pelo administrador do sistema.

Figura 6 - Digrama de classes.



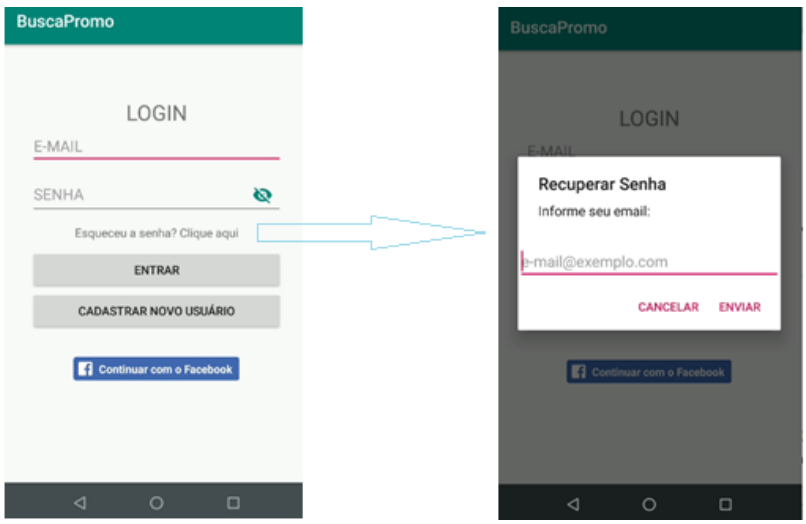
Fonte: Autoria própria

4.3 APRESENTAÇÃO DO SISTEMA

Esta seção apresenta o funcionamento do aplicativo com suas principais telas.

Na Figura 7a é possível visualizar a primeira tela do aplicativo que é a de autenticação de usuário. O usuário cadastrado informa seu e-mail e senha, ou se desejar, pode realizar autenticação utilizando sua conta na rede social Facebook. Nesta tela, também é possível o usuário solicitar a recuperação de senha, clicando no texto “Esqueceu a senha? Clique aqui” (Figura 7b).

Figura 7 - Tela inicial – (a) tela de autenticação; (b) tela de recuperação de senha.

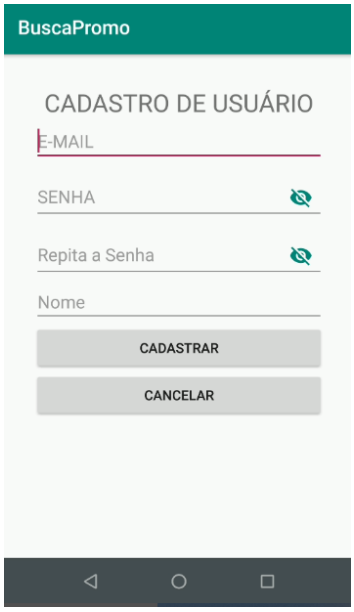


(a) **(b)**

Fonte: Autoria própria

Para cadastrar um novo usuário é necessário clicar no botão “Cadastrar novo usuário”. Em seguida, o usuário será redirecionado para tela de cadastro apresentada na Figura 8. Nessa tela, o usuário informa os dados solicitados para realizar o cadastro.

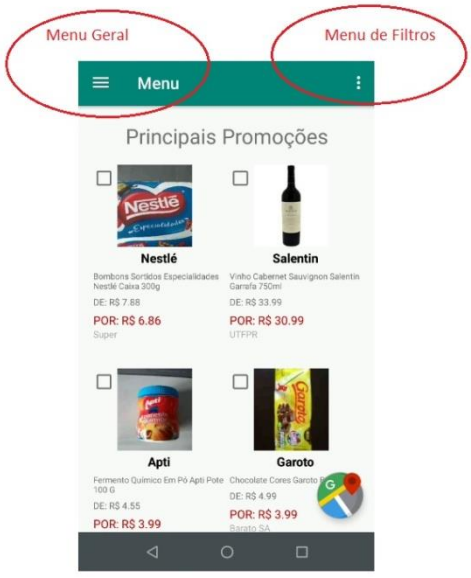
Figura 8 - Cadastro próprio usuário.



Fonte: Autoria própria

Após a autenticação do usuário ser realizada, o sistema abre a tela principal. Nessa tela, as promoções ativas são mostradas em forma de lista. A tela possui dois menus, um menu de filtros para pesquisa de promoções para produtos específicos como mostra a Figura 9.

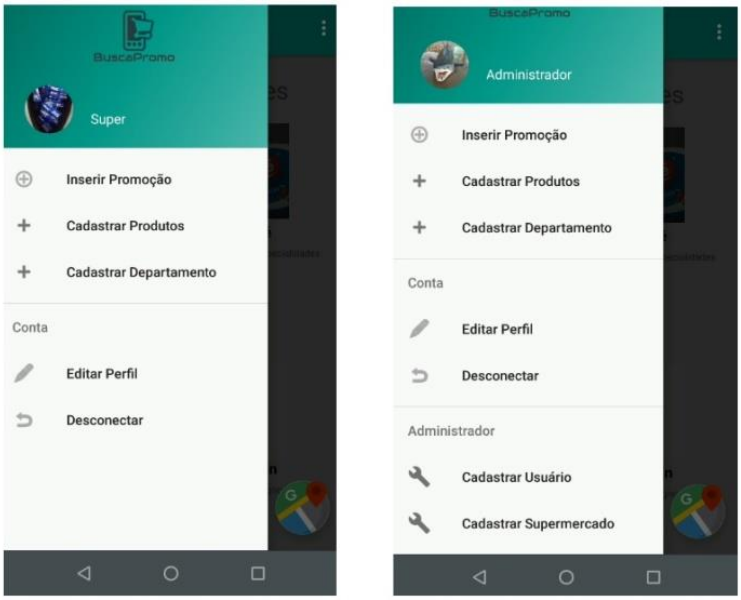
Figura 9 - Menu Geral e menu de filtros.



Fonte: Autoria própria

O menu geral, conta com atalhos para direcionamento para outras telas que, de acordo com perfil do usuário autenticado, podem ser exibidos ou ocultados, como mostram a Figura 10a para menu do usuário supermercado, Figura 10b para usuário administrador, Figura 11a para cliente Figura 11b para usuário autenticado via Facebook.

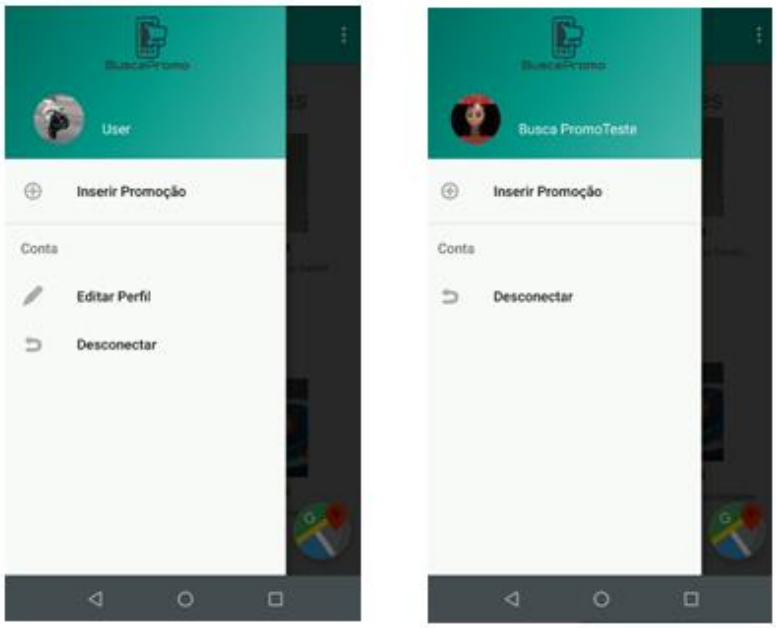
Figura 10 - Tela inicial - (a) menu supermercado e (b) menu administrador.



(a) **(b)**

Fonte: Autoria própria

Figura 11 - Tela inicial - (a) menu usuário e (b) menu Facebook.

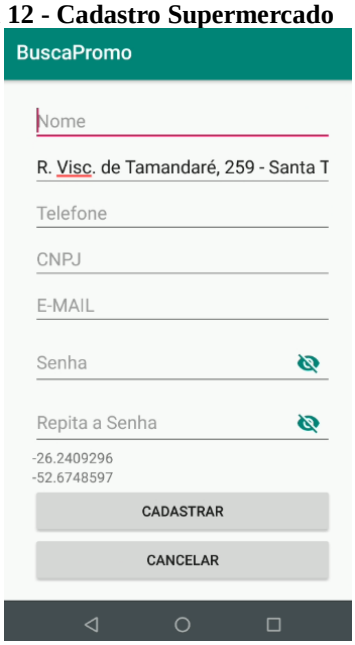


(a) (b)

Fonte: Autoria própria

A seguir são detalhadas as telas que somente o usuário com perfil de administrador do sistema tem acesso. A Figura 12 mostra a tela de cadastro de supermercado, pode-se verificar, além de dados básicos de um cadastro, campos que informam a latitude e longitude capturadas pela localização do próprio *smartphone*, os quais servem para gerar um mapa, e retornam o endereço fornecido pelo Google Maps.

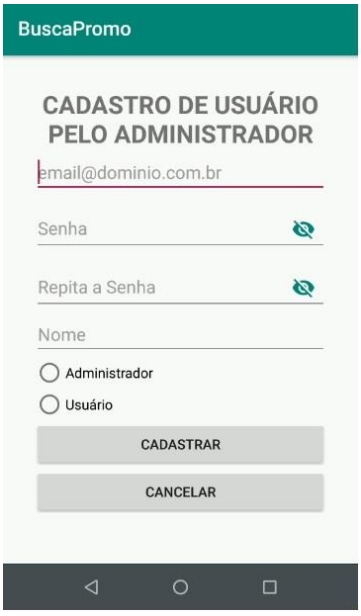
Figura 12 - Cadastro Supermercado



Fonte: Autoria própria

Na Figura 13 é apresentado o cadastro de usuário via administrador, o qual poderá incluir mais de um administrador ao sistema.

Figura 13 - Cadastro usuário via administrador.

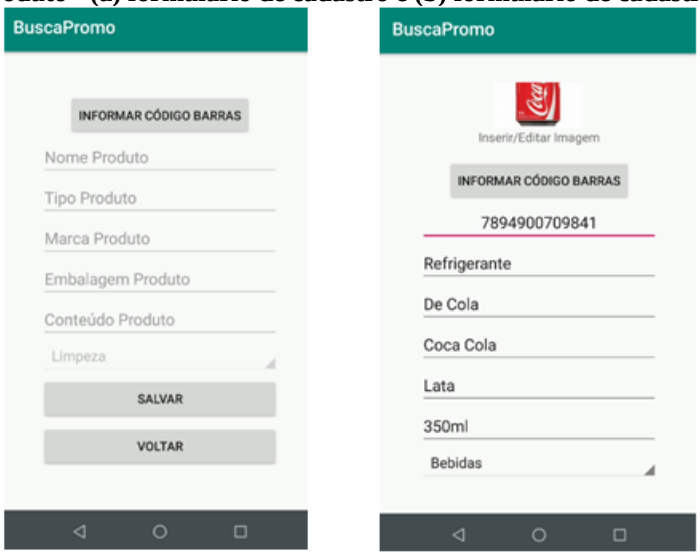


Fonte: Autoria própria

As telas detalhadas a seguir são de uso comum ao administrador e ao supermercado.

Na Figura 14a é exibida a tela de cadastro de produtos. Essa tela possui um botão que, ao ser clicado, solicita confirmação se o usuário deseja informar o código de barras por meio de digitação ou capturar essa informação utilizando a câmera do *smartphone*. Caso o produto esteja cadastrado no banco de dados, os campos são preenchidos de acordo com o cadastro (Figura 14b). Caso contrário, o usuário preenche todos os campos, inclusive devendo inserir uma imagem que corresponda ao produto que está sendo inserido.

Figura 14 - Cadastro de produto - (a) formulário de cadastro e (b) formulário de cadastro preenchido.



(a) (b)

Fonte: Autoria própria

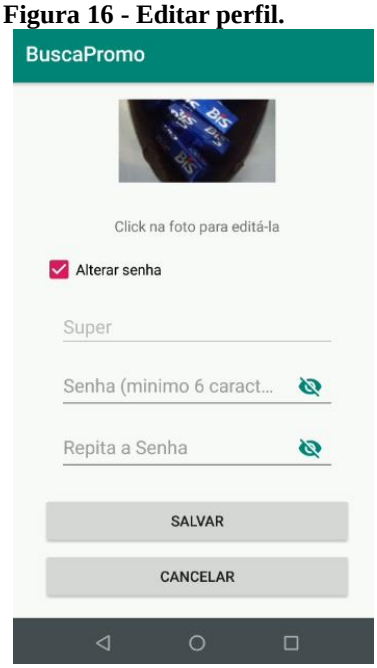
Na Figura 15 é mostrada a tela de cadastro de departamento, que possui somente um campo de texto que se auto completa conforme o usuário digita, caso um departamento que está sendo inserido esteja cadastrado, o mesmo será mostrado e não será permitida a inserção em duplicidade.

Figura 15 - Cadastro de departamento.

Fonte: Autoria própria

A tela de edição de perfil, da Figura 16, é de uso comum, exceto para o usuário que autentica-se via Facebook, devido ao seu cadastro ser externo. Nesta tela, existe uma diferenciação entre os perfis de cliente e supermercado, que para o cliente é permitido alteração de nome e senha e, para o supermercado, é permitido alteração somente de senha, devido ao cadastro do supermercado ser diferenciado.

Figura 16 - Editar perfil.

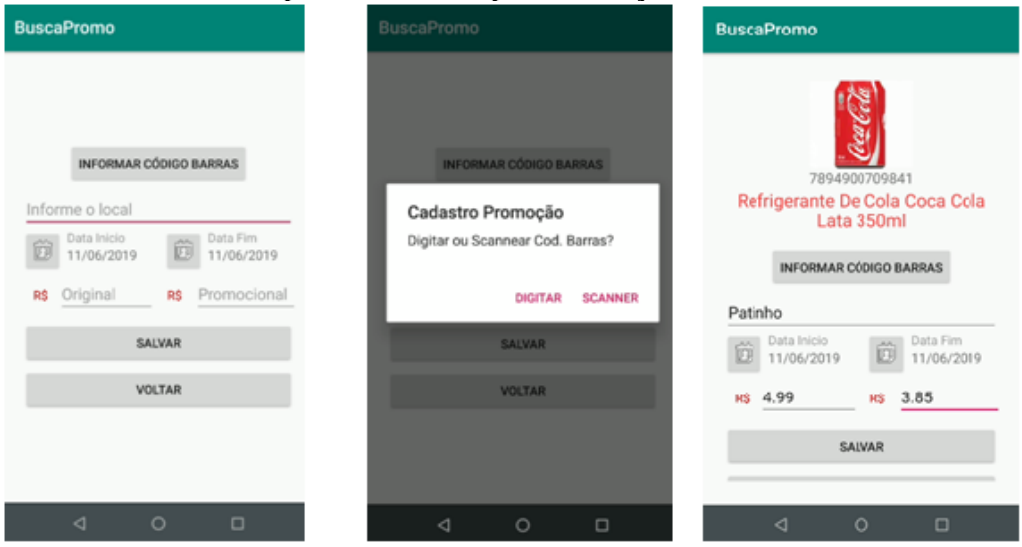


Fonte: Autoria própria

A principal tela de interação permitida a todos os usuários é a tela de inserção de promoções e está detalhada na Figura 17a. Nessa tela, existe um botão que, ao ser clicado, solicita confirmação se o usuário deseja digitar ou utilizar o código de barras para encontrar o produto, Figura 17b. Quando o produto é encontrado são exibidas a descrição e a imagem. Em seguida o usuário deverá preencher o campo “local” informando o nome do supermercado em que se encontra o produto. Esse campo de texto se auto completa conforme ocorre a digitação e, caso o local seja um supermercado cadastrado, o seu nome aparece numa lista, caso contrário, o usuário deve informar um nome para o local, conforme apresentado na Figura 17c.

Na tela de inserção de promoções também existem dois campos de data. O cliente não tem permissão para alterá-las, somente os usuários com perfil de supermercado ou administrador, pois o cliente não tem domínio sobre o valor e prazo de uma determinada promoção, assim, ele poderá somente informar os valores que estão sendo oferecidos no momento em que se encontra no supermercado. E para finalizar, existem dois campos de valores, o de valor original do produto e o de valor promocional, os quais devem ser informados, não permitindo a inserção de um valor promocional maior do que o valor original.

Figura 17 - Inserir promoção - (a) tela de inserção promoções, (b) como informar do código de barras e (c) tela preenchida com o produto desejado.

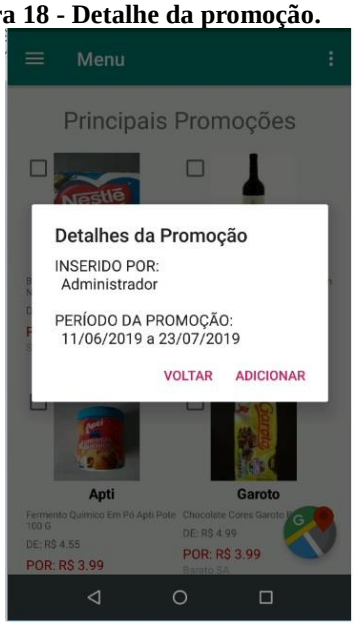


(a) (b) (c)

Fonte: Autoria própria

Na tela principal, exibida na Figura 9, é disponibilizada a opção de clicar na imagem da promoção e, assim, verificar detalhes como a periodicidade da mesma, e o usuário que fez a inserção, como é mostrada na Figura 18.

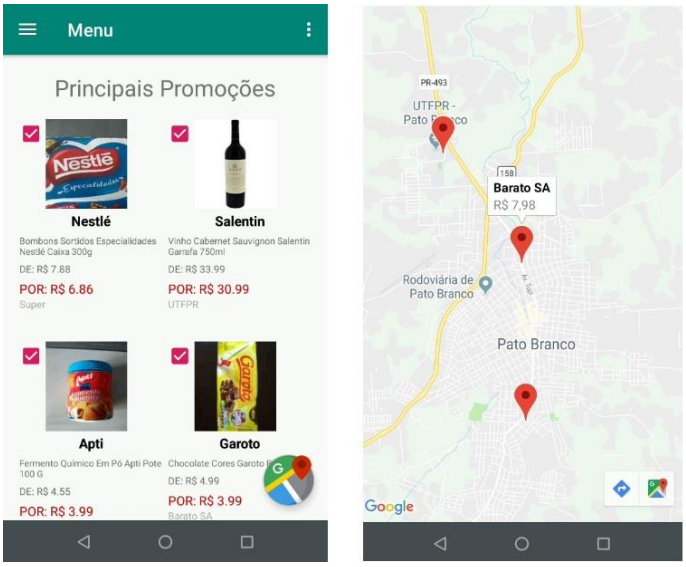
Figura 18 - Detalhe da promoção.



Fonte: Autoria própria

Nessa tela é possível selecionar promoções por meio de um campo do tipo *check box* ou filtrar como informado anteriormente, e após realizado a seleção é possível gerar um mapa clicando no botão que fica localizado no canto inferior direito da tela da Figura 19a. Assim as promoções serão agrupadas por local, somando os valores e possibilitando ao usuário comparar os valores para vários locais com seus produtos desejados, como mostra a Figura 19b.

Figura 19 - Geração mapa - (a) seleção das promoções e (b) mapa gerado com as promoções selecionadas.



(a) (b)

Fonte: Autoria própria

4.4 IMPLEMENTAÇÃO DO SISTEMA

Para o desenvolvimento do aplicativo foi utilizado o Android Studio, o qual permite que seja definida uma versão mínima do Android que esteja ainda em uso nos *smartphones*. A versão mínima definida foi a 4.0 de API nível 15 chamada de "*Ice Cream Sandwich*" para uma maior compatibilidade de uso.

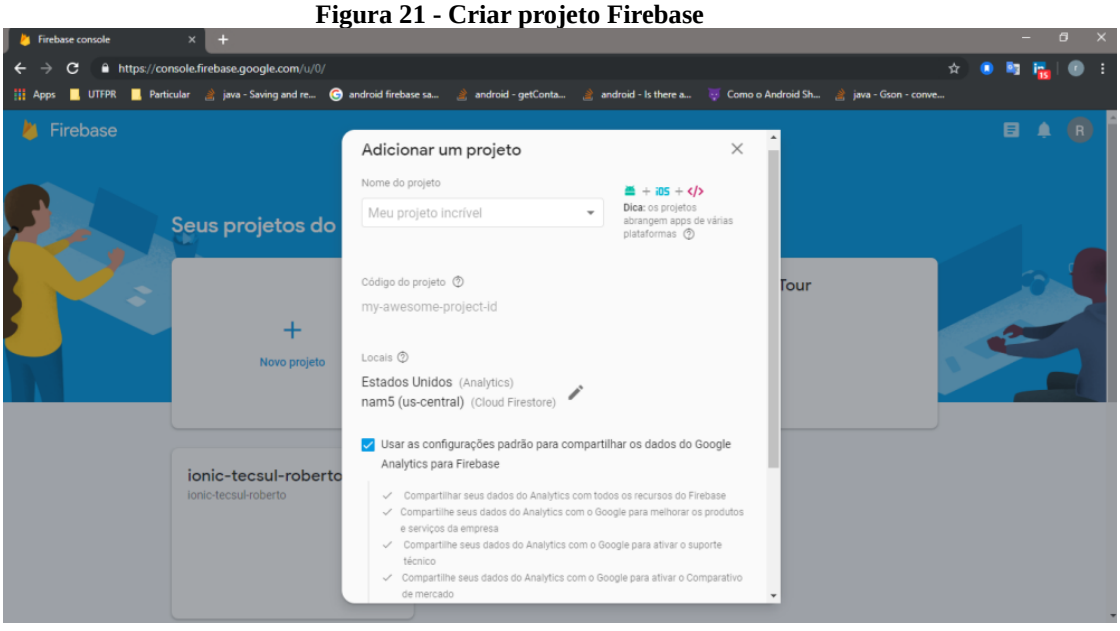
O aparelho *smartphone* utilizado para teste foi um Motorola Moto G5S com versão Android 8.1 versão API 27 chamada de "*Oreo*", como mostra a Figura 20.

Figura 20 - Motorola Moto G5S.

Fonte https://www.motorola.com.br/

Um dos primeiros passos foi a vinculação do projeto desenvolvido no Android Studio com um projeto no console do Firebase. Após cadastro no site https://firebase.google.com/, é possível em poucos passos criar um projeto, como pode ser observado na Figura 21.

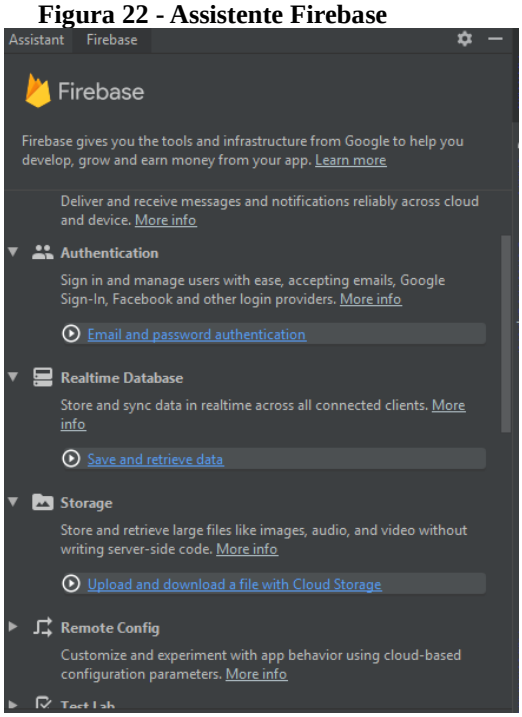
Figura 21 - Criar projeto Firebase



Fonte https://firebase.google.com/

Ao finalizar o projeto no Firebase é gerado um arquivo com o nome de google-services.json, o qual é inserido no projeto do Android Studio, permitindo assim a vinculação entre ambos os projetos, que ao final se tornam um só. A próxima etapa foi escolher quais ferramentas foram utilizadas no projeto, sendo possível realizar isso somente por meio do Android Studio, como mostra a Figura 22.

Figura 22 - Assistente Firebase



Fonte: Autoria própria

A principal parte do código que é responsável pela comunicação com o Firebase está na classe ConfiguraçãoFirebase da Listagem 1. A classe apresenta as instâncias necessárias para o acesso ao banco de dados exibido na linha 16, autenticação pode ser verificado na linha 24 e armazenamento de imagens que está na linha 39.

Listagem 1 - Configuração Firebase.

```
1    package utfpr.edu.br.buscapromo.dao;
2
3    import com.google.firebase.auth.FirebaseAuth;
4    import com.google.firebase.database.DatabaseReference;
5    import com.google.firebase.database.FirebaseDatabase;
6    import com.google.firebase.storage.FirebaseStorage;
7    import com.google.firebase.storage.StorageReference;
8
9    public class ConfiguracaoFirebase {
10
11       private static DatabaseReference referenciaFirebase;
12       private static FirebaseAuth autenticacao;
13       private static FirebaseStorage storage;
14       private static StorageReference referenceStorage;
15
```

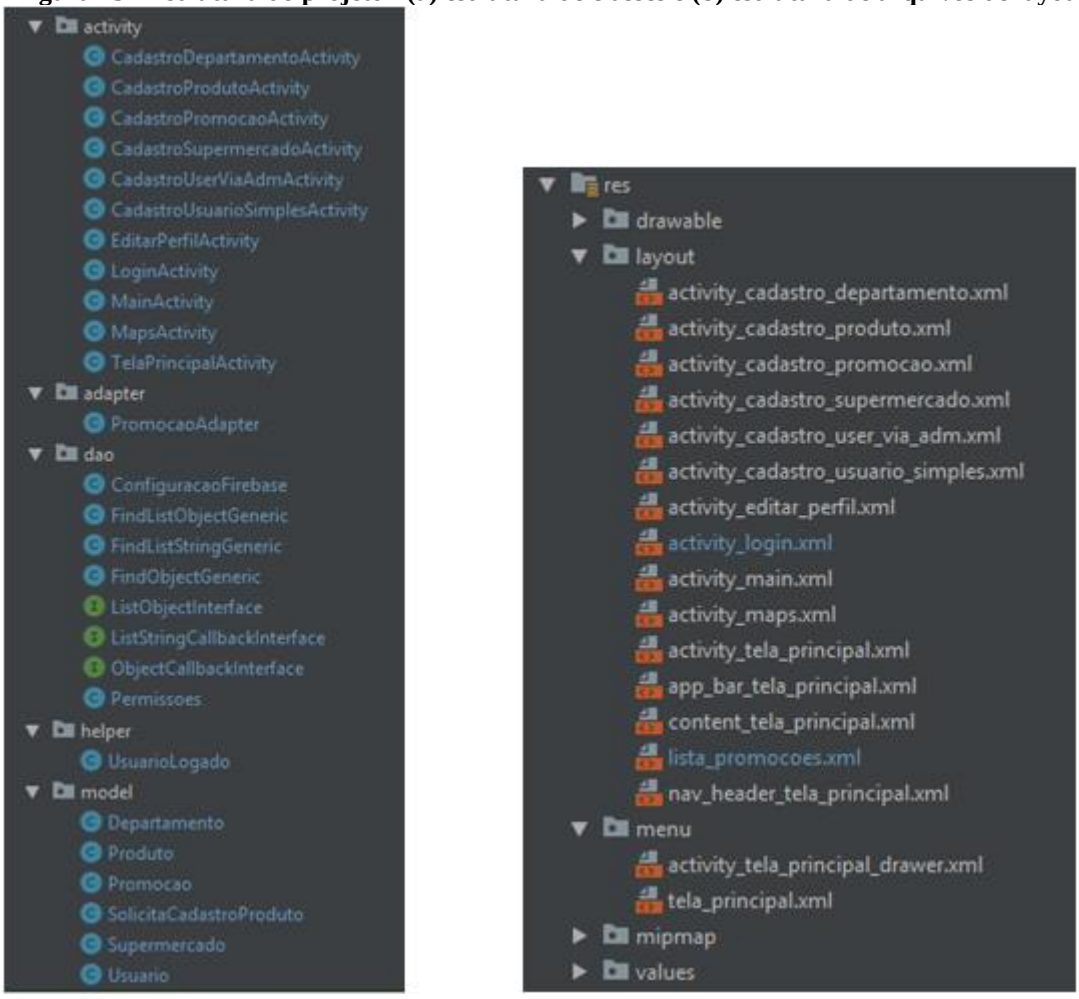
```
16      public static DatabaseReference getFirebase(){
17          if(referenciaFirebase == null){
18              referenciaFirebase = FirebaseDatabase.getInstance()
                .getReference();
19          }
20
21          return referenciaFirebase;
22      }
23
24      public static FirebaseAuth getFirebaseAuth(){
25          if( autenticacao == null){
26              autenticacao = FirebaseAuth.getInstance();
27          }
28
29          return autenticacao;
30      }
31
32      public static FirebaseStorage getFirebaseStorage(){
33          if(storage ==  null){
34              storage = FirebaseStorage.getInstance();
35          }
36          return storage;
37      }
38
39      public static StorageReference getFirebaseStorageReference(){
40          if(referenceStorage == null){
41              referenceStorage = FirebaseStorage.getInstance()
                .getReference();
42          }
43          return referenceStorage;
44      }
45  }
46
```

Fonte: Autoria própria

Para que o projeto ser construído de maneira organizada, as classes foram agrupadas em pacotes separados de acordo com o tipo. Na Figura 23a é possível visualizar a estrutura de classes do projeto desenvolvido, e na Figura 23b é apresentada a estrutura de arquivos de layout das telas.

No pacote *model* encontram-se as classes que modelam o aplicativo, como foi detalhado anteriormente, não sendo necessária a demonstração do código no documento.

Figura 23 - Estrutura do projeto - (a) estrutura de classes e (b) estrutura de arquivos de layout.



(a) (b)

Fonte: Autoria própria

No pacote *activity*, estão todas as codificações de telas do projeto, os códigos para o correto funcionamento das mesmas, e todas as classes *activity* que geram uma tela, possuem um arquivo eXtensible Markup Language (XML) correspondente para codificação gráfica, que ficam no pacote layout. Nos arquivos XML são desenvolvidas as interfaces gráficas da aplicação, contendo componentes como botões, campos de inserção de texto, campos para imagens, entre outros, na Figura 23b são visualizados esses arquivos XML.

Em campos de imagem foi utilizada a biblioteca Picasso, disponível no site https://square.github.io/picasso/. Essa biblioteca facilita utilização de imagens que não estejam na memória do *smatphone*, bastando indicar o local de armazenamento no banco e ela faz o processo de carregamento e processamento da imagem. O código necessário para buscar a imagem é:

Picasso.get().load("url").into(imageView);

No campo URL é inserido o endereço em que a imagem está armazenada e no campo *imageView*, é informado o destino da imagem.

Para que fosse possível realizar a autenticação utilizando a conta da rede social Facebook foi necessário criar um projeto com o mesmo nome no endereço https://developers.facebook.com. Após criar o projeto, é gerado um identificador (ID) do aplicativo, que é inserido no arquivo AndroidManisfest.xml para que, após o aplicativo esteja instalado em um *smartphone*, o mesmo tenha acesso à autenticação via Facebook. Parte do código da tela de autenticação é apresentada na Listagem 2, na linha 63 inicia-se a comunicação com o Facebook, caso obtenha sucesso a linha 68 direciona para o método de validação do usuário que se inicia na linha 200. Após o usuário ser validado retorna para linha 85, a qual informa ao Firebase que usuário está validado e permite acesso ao sistema.

Listagem 2 - Autenticação via Facebook.

```
63              callbackManager = CallbackManager.Factory.create();
64              LoginManager.getInstance().registerCallback(callbackManager,
new FacebookCallback<LoginResult>() {
65
66                @Override
67                public void onSuccess(LoginResult loginResult) {
68
handleFacebookAccessToken(loginResult.getAccessToken());
69                }
70
71                @Override
72                public void onCancel() {
73                    Toast.makeText(getApplicationContext(),
R.string.cancel_login, Toast.LENGTH_SHORT).show();
74                }
75
76                @Override
77                public void onError(FacebookException error) {
78
79                    Toast.makeText(LoginActivity.this, "Verifique conexão
de dados! \n" +
80                            "Verifique usuário e senha! \n" +
81                            "Tente novamente!",
Toast.LENGTH_SHORT).show();
82                }
83            });
84
85            autenticacao = FirebaseAuth.getInstance();
86            firebaseAuthListener = new FirebaseAuth.AuthStateListener() {
87                @Override
88                public void onAuthStateChanged(@NonNull FirebaseAuth
firebaseAuth) {
89                    FirebaseUser user = firebaseAuth.getCurrentUser();
90                    if (user != null) {
91                        Toast.makeText(getApplicationContext(),"Login
efetuado via Facebook com sucesso!", Toast.LENGTH_LONG).show();
92                    }
93                }
```

```
94          };
95
.
.
.
200     private void handleFacebookAccessToken(AccessToken accessToken) {
201         progressBar.setVisibility(View.VISIBLE);
202
203         AuthCredential credential =
FacebookAuthProvider.getCredential(accessToken.getToken());
204
205
autenticacao.signInWithCredential(credential).addOnCompleteListener(this,
new OnCompleteListener<AuthResult>() {
206             @Override
207             public void onComplete(@NonNull Task<AuthResult> task) {
208                 progressBar.setVisibility(View.GONE);
209                 if (!task.isSuccessful()) {
210                     Toast.makeText(getApplicationContext(),"Login
efetuado via Facebook com sucesso!", Toast.LENGTH_LONG).show();
211
212                 }
213                 abrirTelaPrincipal();
214             }
215         });
216     }
```

Fonte: Autoria própria

O código responsável pela autenticação via cadastro próprio, ou seja, via Firebase Authentication está disposto na Listagem 3. O método exibido na linha 161 é responsável por validações dos campos de *e-mail* e senha, seguindo para o método de validação de autenticação de acesso no Firebase para exibido da linha 180, o qual verifica o *e-mail* e senha que foram informados. Caso a validação seja positiva o usuário autenticado é salvo nas preferências do *smatphone*, linha 189, para que não seja necessária autenticação sempre que o aplicativo for iniciado. Em seguida o usuário é direcionado para a tela principal, como pode ser observado na linha 190.

Listagem 3 - Autenticação via cadastro próprio.

```
149    @Override
150     protected void onStart() {
151         super.onStart();
152         autenticacao.addAuthStateListener(firebaseAuthListener);
153     }
154
155     @Override
156     protected void onStop() {
157         super.onStop();
158         autenticacao.removeAuthStateListener(firebaseAuthListener);
159     }
160
161     public void btnLoginOnclick(View view) {
162         if (!edtEmailLogin.getText().toString().equals("") &&
!edtSenhaLogin.getText().toString().equals("")) {
163             usuario = new Usuario();
```

```
164              usuario.setEmail(edtEmailLogin.getText().toString());
165              usuario.setSenha(edtSenhaLogin.getText().toString());
166
167              validarLogin();
168          } else {
169              Toast.makeText(LoginActivity.this, "Preencha os campos de
E-Mail e Senha", Toast.LENGTH_SHORT).show();
170          }
171      }
172
173      @Override
174      protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
175          super.onActivityResult(requestCode, resultCode, data);
176
177          callbackManager.onActivityResult(requestCode, resultCode,
data);
178      }
179
180      private void validarLogin(){
181          progressBar.setVisibility(View.GONE);
182          autenticacao = ConfiguracaoFirebase.getFirebaseAuth();
183
autenticacao.signInWithEmailAndPassword(usuario.getEmail().toString(),
usuario.getSenha().toString()).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
184              @Override
185              public void onComplete(@NonNull Task<AuthResult> task) {
186                  if ( task.isSuccessful()){
187
188                      UsuarioLogado usuarioLogado = new
UsuarioLogado(LoginActivity.this);
189
usuarioLogado.salvarUsuarioPreferencias(usuario.getEmail(),
usuario.getSenha());
190                      abrirTelaPrincipal();
191                      Toast.makeText(LoginActivity.this, "Login efetuado
com sucesso", Toast.LENGTH_SHORT).show();
192                  }else {
193                      Toast.makeText(LoginActivity.this, "Usuário ou
senha não conferem!!", Toast.LENGTH_SHORT).show();
194                  }
195              }
196          });
197      }
```

Fonte: Autoria própria

O código responsável por recuperar a senha está na Listagem 4. Na linha 96 está o código responsável em gerar uma tela de alerta onde o usuário poderá inserir seu *e-mail*, que será utilizado para receber um *link* que possibilite a recuperação da senha, na linha 114 tem-se o código que informa ao Firebase o *e-mail* para onde deverá ser enviado o link de recuperação.

Listagem 4 - Recuperação de senha.

```
96          tvRecuperarSenha.setOnClickListener(new View.OnClickListener()
{
97              @Override
98              public void onClick(View v) {
99                  AlertDialog.Builder builder = new
AlertDialog.Builder(LoginActivity.this);
100                 builder.setCancelable(false);
101                 builder.setTitle("Recuperar Senha");
102                 builder.setMessage("Informe seu email:");
103                 builder.setView(edEmailRecuperar);
104
105                     builder.setPositiveButton("Enviar", new
DialogInterface.OnClickListener() {
106                         @Override
107                         public void onClick(DialogInterface dialog,
int which) {
108                             if (edEmailRecuperar.length() > 6 ) {
109
110                                 autenticacao =
FirebaseAuth.getInstance();
111
112                                 String emailRecuperar =
edEmailRecuperar.getText().toString();
113
114
autenticacao.sendPasswordResetEmail(emailRecuperar).addOnCompleteListener(n
ew OnCompleteListener<Void>() {
115                                     @Override
116                                     public void onComplete(@NonNull
Task<Void> task) {
117
118                                         if (task.isSuccessful()) {
119
Toast.makeText(getApplicationContext(), "Email enviado, verifique sua caixa
de entrada!", Toast.LENGTH_SHORT).show();
120                                             recreate();
121                                         } else {
122
Toast.makeText(getApplicationContext(), "Falha ao enviar email!",
Toast.LENGTH_SHORT).show();
123                                             recreate();
124                                         }
125                                     }
126                                 });
127
128                         }else
129
130                         {
131                             Toast.makeText(getApplicationContext(),
"Necessário informar seu email!", Toast.LENGTH_SHORT).show();
132                             recreate();
133                         }
134                     }
135                     });
136
137                     builder.setNegativeButton("Cancelar", new
DialogInterface.OnClickListener() {
138                         @Override
```

```
139                             public void onClick(DialogInterface dialog,
int which) {
140                                 recreate();
141                             }
142                         });
143                     alerta = builder.create();
144                     alerta.show();
145                 }
146             });
147         }
```

Fonte: Autoria própria

O cadastro de usuários é basicamente o mesmo para todos os tipos de usuários, o que difere, são campos como CNPJ e endereço que o usuário supermercado possui e o cliente não, entre outros, na Listagem 5 está o código de cadastro de supermercado. Neste código está incluso parte do código de cadastro de usuário como pode ser observado na linha 101, onde se inicia a construção do objeto usuário, pois quando se cadastra um supermercado, automaticamente é gerado um usuário para este supermercado e, na linha 107, é exibido o início da construção do objeto supermercado. A linha 125 contém o método que é responsável pela validação do cadastro, onde são validados campos como o tamanho mínimo de senha (6 dígitos), a necessidade de repetir senha para validar a digitação bem como a obrigatoriedade do *e-mail* ser válido, se essa validação for positiva inicia-se o método da linha 161 que faz a inserção do usuário no banco de dados, como pode-se observar na linha 164 quando um novo cadastro é gerado, é solicitado ao Firebase uma *key* (chave) para em seguida realizar a gravação dos dados nesta *key* gerada, e para finalizar o cadastro, neste caso do supermercado, na linha 177 encontra-se o método responsável em gravar no banco de dados o cadastro do supermercado.

Listagem 5 - Cadastro de supermercado.

```
97              btnCadastrar.setOnClickListener(new View.OnClickListener() {
98                  @Override
99                  public void onClick(View v) {
100                     if
(senha1.getText().toString().equals(senha2.getText().toString())) {
101                         usuario = new Usuario();
102                         usuario.setEmail(email.getText().toString());
103                         usuario.setSenha(senha1.getText().toString());
104                         usuario.setNome(nome.getText().toString());
105                         usuario.setTipoUsuario("Supermercado");
106
107                         supermercado = new Supermercado();
108
109                         supermercado.setNome(nome.getText().toString());
110
supermercado.setEndereco(endereco.getText().toString());
111
supermercado.setTelefone(telefone.getText().toString());
112                         supermercado.setCnpj(cnpj.getText().toString());
```

```
113
supermercado.setLatitude(latitude.getText().toString());
114
supermercado.setLongitude(longitude.getText().toString());
115                     supermercado.setUsuario(usuario);
116                     cadastrarUsuario();
117                     insereSupermercado(supermercado);
118                 } else {
119                     Toast.makeText(CadastroSupermercadoActivity.this,
"Senhas Informadas Não Conferem!", Toast.LENGTH_LONG).show();
120                 }
121             }
122         });
123     }
124
125     private void cadastrarUsuario() {
126
127         autenticacao = ConfiguracaoFirebase.getFirebaseAuth();
128         autenticacao.createUserWithEmailAndPassword(
129                 usuario.getEmail(),
130                 usuario.getSenha()
131         ).addOnCompleteListener(CadastroSupermercadoActivity.this, new
OnCompleteListener<AuthResult>() {
132             @Override
133             public void onComplete(@NonNull Task<AuthResult> task) {
134
135                 if (task.isSuccessful()) {
136                     insereUsuario(usuario);
137                     finish();
138                     autenticacao.signOut();
139                 } else {
140
141                     String erroExcecao = "";
142
143                     try {
144                         throw task.getException();
145                     } catch (FirebaseAuthActionCodeException e) {
146                         erroExcecao = "Senha muito fraca, deve conter
mínimo 6 caracteres com letras e números!";
147                     } catch (FirebaseAuthInvalidCredentialsException
e) {
148                         erroExcecao = "E-mail inválido, digite novo e-
mail!";
149                     } catch (FirebaseAuthUserCollisionException e) {
150                         erroExcecao = "E-mail já cadastrado!";
151                     } catch (Exception e) {
152                         erroExcecao = "Erro ao efetuar cadastro!";
153                         e.printStackTrace();
154                     }
155                     Toast.makeText(CadastroSupermercadoActivity.this,
"Erro!" + erroExcecao, Toast.LENGTH_LONG).show();
156                 }
157             }
158         });
159     }
160
161     private boolean insereUsuario(Usuario usuario) {
162         try {
163             reference =
ConfiguracaoFirebase.getFirebase().child("usuarios");
164             String key = reference.push().getKey();
```

```
165              usuario.setKey(key);
166              reference.child(key).setValue(usuario);
167              Toast.makeText(CadastroSupermercadoActivity.this, "Usuário
Cadastrado com Sucesso!!", Toast.LENGTH_LONG).show();
168              return true;
169          } catch (Exception e) {
170              Toast.makeText(CadastroSupermercadoActivity.this, "Erro ao
salvar usuário", Toast.LENGTH_LONG).show();
171              e.printStackTrace();
172              return false;
173          }
174
175      }
176
177      private boolean insereSupermercado(Supermercado supermercado) {
178          try {
179              reference =
ConfiguracaoFirebase.getFirebase().child("supermercados");
180              reference.push().setValue(supermercado);
181              Toast.makeText(CadastroSupermercadoActivity.this,
"Supermercado Cadastrado com Sucesso!!", Toast.LENGTH_LONG).show();
182              return true;
183          } catch (Exception e) {
184              Toast.makeText(CadastroSupermercadoActivity.this, "Erro ao
salvar supermercado", Toast.LENGTH_LONG).show();
185              e.printStackTrace();
186              return false;
187          }
188      }
```

Fonte: Autoria própria

No pacote *dao* encontram-se todas as classes necessárias para comunicação com o Firebase, para busca de promoções, produtos, validação de usuário e permissões.

Para busca no banco de dados, foram implementadas três classes diferentes com suas respectivas interfaces.

A classe que retorna uma lista de objetos é a FindListObjectGeneric, apresentada na Listagem 6, como pode ser observado na linha 25 da Listagem 6 o método recebe como parâmetros, um Objeto que é a finalidade da busca, uma *String* que serve de filtro para a requisição, o *Context* que é contexto, ou seja a tela que faz a requisição, e por fim a interface que faz a solicitação que é a ListObjectInterface exibida na Listagem 7.

Listagem 6 - FindListObjectGeneric.

```
17   public class FindListObjectGeneric {
18
19       private DatabaseReference databaseReference;
20       private Context context;
21       private Object objectFind;
22       private List<Object> objects;
23
24
25       public void findListObjectGeneric(Object objeto, String filtro1,
Context c, final ListObjectInterface callback) {
```

```
26           this.objectFind = objeto;
27           this.context = c;
28
29           objects = new ArrayList<>();
30           databaseReference =
FirebaseDatabase.getInstance().getReference();
31           databaseReference.child(filtro1).addValueEventListener(new
ValueEventListener() {
32               @Override
33               public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
34                   objects.clear();
35                   for (DataSnapshot posSnapshot :
dataSnapshot.getChildren()) {
36
objects.add(posSnapshot.getValue(objectFind.getClass()));
37                   }
38                   callback.onCallback(objects);
39               }
40
41               @Override
42               public void onCancelled(@NonNull DatabaseError
databaseError) {
43                   Toast.makeText(context, "ERRO!!" + databaseError,
Toast.LENGTH_LONG).show();
44               }
45           });
46       }
47   }
```

Listagem 7 - ListObjectInterface.

```
public interface ListObjectInterface {
6        void onCallback (List<Object> objects);
7    }
```

A classe que retorna uma lista de *String* é a FindListStringGeneric, exibida na Listagem 8, na linha 26 pode-se visualizar que o método recebe duas *Strings* como filtro da busca, o contexto e sua interface que é a ListStringCallbackInterface, apresentada na Listagem 9. Esta classe pode ser utilizada por exemplo, para retornar uma lista de departamentos do supermercado que estejam armazenados no banco de dados.

Listagem 8 - FindListStringGeneric

```
18   public class FindListStringGeneric {
19
20       private DatabaseReference databaseReference;
21       private Context context;
22       private String filtro1;
23       private String filtro2;
24       private ListStringCallbackInterface callback;
25
26       public void listaStringGenericCallback(final String filter1, final
String filter2, Context c, final ListStringCallbackInterface callback) {
27           this.context = c;
```

```
28            this.filtro1 = filter1;
29            this.filtro2 = filter2;
30            this.callback = callback;
31
32            databaseReference =
FirebaseDatabase.getInstance().getReference();
33            databaseReference.child(filtro1).addValueEventListener(new
ValueEventListener() {
34                @Override
35                public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
36                    final List<String> filtroList = new
ArrayList<String>();
37
38                    for (DataSnapshot depSnapshot :
dataSnapshot.getChildren()) {
39                        String filtroSelect =
depSnapshot.child(filtro2).getValue(String.class);
40                            filtroList.add(filtroSelect);
41                    }
42
43                    ArrayAdapter<String> filtroAdapter = new
ArrayAdapter<String>(context,
44                            android.R.layout.simple_spinner_item,
filtroList);
45
filtroAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropd
own_item);
46                    callback.onCallback(filtroAdapter);
47                }
48
49                @Override
50                public void onCancelled(@NonNull DatabaseError
databaseError) {
51                    Toast.makeText(context, "ERRO!!" + databaseError,
Toast.LENGTH_LONG).show();
52
53                }
54            });
55        }
56
57    }
58


public interface ListStringCallbackInterface {
6
7        void onCallback ( ArrayAdapter<String> myAdapter);
8    }
```

Fonte: Autoria própria

Listagem 9 - ListStringCallbackInterface

A classe que retorna um único objeto específico é a FindObjectGeneric, exibida na Listagem 10. Na linha 22 está o método responsável em receber três *Strings* que servem de filtro para busca no banco de dados, o objeto a qual a busca é direcionada, o contexto e sua interface sendo a ObjectCallbackInterface, apresentada na Listagem 11.

Listagem 10 - FindObjectGeneric

```
15      private DatabaseReference databaseReference;
16        private Context context;
17        private String filtro1;
18        private String filtro2;
19        private String filtro3;
20        private Object objectFind;
21
22        public void findObjectCallback(String filter1, String filter2,
String filter3, Object object, Context c, final ObjectCallbackInterface
callback) {
23            this.filtro1 = filter1;
24            this.filtro2 = filter2;
25            this.filtro3 = filter3;
26            this.context = c;
27            this.objectFind = object;
28
29            databaseReference =
FirebaseDatabase.getInstance().getReference();
30
31
databaseReference.child(filtro1).orderByChild(filtro2).equalTo(filtro3)
32                    .addValueEventListener(new ValueEventListener() {
33                        @Override
34                        public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
35                            for (DataSnapshot posSnapshot :
dataSnapshot.getChildren()) {
36                                if (dataSnapshot.exists()) {
37                                    objectFind =
posSnapshot.getValue(objectFind.getClass());
38                                }
39                            }
40                            if (!dataSnapshot.exists()) {
41
42                            }
43                            callback.onCallback(objectFind);
44                        }
45
46                        @Override
47                        public void onCancelled(@NonNull DatabaseError
databaseError) {
48                            Toast.makeText(context, "ERRO!!" +
databaseError, Toast.LENGTH_LONG).show();
49
50                        }
51                    });
52        }
```

Fonte: Autoria própria

```
public interface ObjectCallbackInterface {
4         void onCallback(Object object);
5     }
```

Listagem 11 - ObjectCallbackInterface,

A implementação se fez necessária desta maneira, devido ao funcionamento assíncrono do Firebase, assim o sistema aguarda o retorno do banco de dados para prosseguir a execução do código.

O pacote *adapter* contém somente uma classe, sendo a responsável em montar a lista de promoções que retornam do banco de dados de acordo com a seleção do usuário, sempre de maneira dinâmica, com esta lista sendo reaproveitada para cada seleção que o usuário realizar, para isso ela utiliza o componente RecyclerView do *Android*, colocando novos valores nas *views* sempre que a tela subir/descer, outro *design-patterns* utilizado é o *View Holder* que tem por objetivo manter as referências da *view* ajudando na reciclagem da tela. O código que realiza esta ação está na Listagem 12. Na linha 137 inicia-se o método responsável em identificar os componentes no arquivo xml correspondente para que a lista seja recriada sempre que necessário, na linha 103 tem início o método que gera um tela de alerta onde o usuário visualiza informações da promoção sempre que clicar na imagem do produto correspondente a promoção desejada. Para selecionar o *check Box* correspondente a uma promoção se faz necessário o código iniciado na linha 94. E na linha 62 tem-se o método responsável em gerar a exibição na lista, das informações da promoção que retornam do bando de dados.

Listagem 12 - PromoçãoAdapter.

```
31   public class PromocaoAdapter  extends
RecyclerView.Adapter<PromocaoAdapter.ViewHolder> implements Serializable {
32
33       private List<Promocao> promocaoList;
34       private Context context;
35       private  List<Promocao> promocoesSelect = new ArrayList<>();
36       private AlertDialog alerta;
37       private Date dataIn = new Date();
38       private Date dataOut = new Date();
39       private SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyyMMdd");
40       private SimpleDateFormat dateFormat2 = new
SimpleDateFormat("dd/MM/yyyy");
41
42
43       @GlideModule
44       public final class MyAppGlideModule extends AppGlideModule {
45           // leave empty for now
46       }
47
48       public PromocaoAdapter(List<Promocao> list, Context c) {
49           context = c;
50           promocaoList = list;
51       }
52
53       @Override
54       public PromocaoAdapter.ViewHolder onCreateViewHolder( ViewGroup
viewGroup, int i) {
55
56           View itemView =
LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.lista_promocoe
s, viewGroup, false);
57
58           return new PromocaoAdapter.ViewHolder(itemView);
59       }
```

```
60
61       @Override
62       public void onBindViewHolder(final PromocaoAdapter.ViewHolder
holder, final int position) {
63
64           final Promocao item = promocaoList.get(position);
65
66           holder.txtMarcaProduto.setText(item.getProduto().getMarca());
67
holder.txtDescricaoProduto.setText(item.getProduto().getNomeProduto() +" "
+ item.getProduto().getTipo()
68                   + " " + item.getProduto().getMarca()
69                   + " " + item.getProduto().getEmbalagem()
70                   + " " + item.getProduto().getConteudo());
71
72           holder.txtValorOriginal.setText("DE: R$ " +
item.getValorOriginal());
73           holder.txtValorPromocional.setText( "POR: R$ " +
item.getValorPromocional());
74           holder.txtLocal.setText(item.getSupermercado());
75
76           String datValid = item.getDataValidade().toString() ;
77           String datIn = item.getDataInsercao().toString();
78           try {
79               dataIn = dateFormat.parse(datIn);
80               dataOut = dateFormat.parse(datValid);
81
82           } catch (ParseException e) {
83               e.printStackTrace();
84           }
85
86           // ---- busca de imagem do produto
87           DisplayMetrics displayMetrics =
context.getResources().getDisplayMetrics();
88           final int height = (displayMetrics.heightPixels / 6);
89           final int width = (displayMetrics.widthPixels / 4);
90
Picasso.get().load(item.getProduto().getUrlImagem()).resize(width,
height).into(holder.imgProduto);
91           // ----
92
93
94           holder.checkboxPromocao.setOnClickListener(new
View.OnClickListener() {
95               @Override
96               public void onClick(View v) {
97                   if(holder.checkboxPromocao.isChecked()){
98                       promocoesSelect.add(promocaoList.get(position));
99                   }
100              }
101          });
102
103          holder.imgProduto.setOnClickListener(new
View.OnClickListener() {
104              @Override
105              public void onClick(View v) {
106
107                  AlertDialog.Builder builder = new
AlertDialog.Builder(context);
108                  builder.setCancelable(true);
109                  builder.setTitle("Detalhes da Promoção");
```

```
110                 builder.setMessage("INSERIDO POR:\n" + "\t" +
item.getUsuario()
111                         + " \n\nPERÍODO DA PROMOÇÃO:\n" + "\t" +
dateFormat2.format(dataIn) +" a "+ dateFormat2.format(dataOut));
112
113                 builder.setPositiveButton("Adicionar", new
DialogInterface.OnClickListener() {
114                     @Override
115                     public void onClick(DialogInterface dialog, int
which) {
116                       holder.checkboxPromocao.setChecked(true);
117                     }
118                 });
119                 builder.setNegativeButton("Voltar", new
DialogInterface.OnClickListener() {
120                     @RequiresApi(api =
Build.VERSION_CODES.LOLLIPOP_MR1)
121                     @Override
122                     public void onClick(DialogInterface dialog, int
which) {
123                      alerta.dismiss();
124                     }
125                 });
126                 alerta = builder.create();
127                 alerta.show();
128             }
129         });
130     }
131
132     @Override
133     public int getItemCount() {
134         return promocaoList.size();
135     }
136
137     public static class ViewHolder extends RecyclerView.ViewHolder {
138
139         protected TextView txtMarcaProduto;
140         protected TextView txtDescricaoProduto;
141         protected ImageView imgProduto;
142         protected TextView txtValorOriginal;
143         protected TextView txtValorPromocional;
144         protected TextView txtLocal;
145         protected CheckBox checkboxPromocao;
146
147         public ViewHolder(View itemView) {
148             super(itemView);
149
150             txtMarcaProduto = (TextView)
itemView.findViewById(R.id.txtMarcaProduto);
151             txtDescricaoProduto = (TextView)
itemView.findViewById(R.id.txtDescricaoProduto);
152             imgProduto = (ImageView)
itemView.findViewById(R.id.imgProduto);
153             txtValorOriginal =
itemView.findViewById(R.id.txtValorProdutoOficial);
154             txtValorPromocional =
itemView.findViewById(R.id.txtValorProdutoUsuario);
155             txtLocal = itemView.findViewById(R.id.txtLocal);
156             checkboxPromocao =
itemView.findViewById(R.id.checkboxPromocao);
157         }
```

```
158       }
159
160       public List<Promocao> getPromocoesSelect() {
161           if(promocoesSelect.isEmpty()){
162               return promocaoList;
163           }else {
164               return promocoesSelect;
165           }
166       }
167   }
```

Fonte: Autoria própria

O pacote *helper* contém uma classe Listagem 13, que é responsável por armazenar qual usuário está autenticado, basicamente para o mesmo não ter que efetuar nova autenticação sempre que for utilizar o aplicativo. Na linha 24 os dados de *e-mail* e senha são armazenados na preferências do usuário, utilizando o armazenamento do *smartphone*.

Listagem 13 - UsuárioLogado.

```
6     public class UsuarioLogado {
7
8         private Context context;
9         private SharedPreferences preferences;
10        private String NOME_ARQUIVO = "app.preference";
11        private int MODE = 0;
12        private SharedPreferences.Editor editor;
13
14        private final String EMAIL_USUARIO_LOGADO =
"email_usuario_logado";
15        private final String SENHA_USUARIO_LOGADO =
"senha_usuario_logado";
16
17        public UsuarioLogado(Context contextParametro){
18            context = contextParametro;
19            preferences = context.getSharedPreferences(NOME_ARQUIVO,
MODE);
20
21            editor = preferences.edit();
22        }
23
24        public void salvarUsuarioPreferencias(String email, String senha){
25            editor.putString(EMAIL_USUARIO_LOGADO, email);
26            editor.putString(SENHA_USUARIO_LOGADO, senha);
27            editor.commit();
28        }
29
30        public String getEmail_Usuario_Logado(){
31            return preferences.getString(EMAIL_USUARIO_LOGADO, null);
32        }
33
34        public String getSenha_Usuario_Logado(){
35            return preferences.getString(SENHA_USUARIO_LOGADO, null);
36        }
37
38    }
```

Fonte: Autoria própria

5 CONCLUSÃO

O aplicativo desenvolvido não tem como finalidade determinar preços de produtos, nem tampouco realizar propagandas de determinado supermercado ou produto, mas auxiliar na divulgação de promoções que auxiliem o consumidor a economizar ao realizar as compras do dia a dia, de maneira colaborativa, ou seja, quanto mais utilizado maior será seu banco de dados de produtos e supermercados, o que pode proporcionar um melhor desenvolvimento para o aplicativo, visto que novas funcionalidade podem ser implantadas como a geração de avisos e listas personalizadas, comparação com históricos de preços em períodos específicos escolhidos pelo usuário.

Neste documento, apresentou-se a documentação, as ferramentas e modelagem necessária para o desenvolvimento do aplicativo, de maneira que englobou-se grande parte do que se necessita para uma boa execução de um projeto como este.

Durante o desenvolvimento do projeto surgiram algumas dificuldades, como, por exemplo, o pouco conhecimento da linguagem Android, que foi sendo aperfeiçoada no decorrer da implementação, vários métodos foram refeitos neste processo, pois a cada nova implementação observou-se que métodos já implantados poderiam ser melhorados, como exemplo é a comunicação com o banco de dados que trabalha de maneira assíncrona, onde inicialmente a cada tela que necessitava de comunicação o código era reescrito. Posteriormente isso foi aperfeiçoado e métodos genéricos foram criados, melhorando de maneira significativa o projeto. Outra dificuldade é referente ao cadastro de produtos, em que não foi possível encontrar um banco de dados público que pudesse ser aproveitado para a elaboração do projeto, todos os que foram pesquisados são de propriedade de alguma empresa que deve ser pago para utilizá-lo. portanto, optou-se por utilizar um banco de dados colaborativo.

Como trabalhos futuros pretende-se revisão constante do código, a fim de redução de linhas de código, melhorando a segurança, velocidade de processamento e comunicação. Projeta-se desenvolver um sistema web que possa servir de apoio para cadastro e controle dos produtos disponibilizados no aplicativo, e observa-se a possibilidade de expansão para outros mercados, como farmácias, postos de combustíveis, pequenas lojas de confecções ou outros setores interessados.

REFERÊNCIAS

ANDROID Developers. 2018. Disponível em: < https://developer.android.com/guide/platform/>. Acesso em: 02 out. 2018.

BRITO, Robison Cris. **Android** : com Android Studio - passo a passo. 1. ed. Rio de Janeiro: Ciência Moderna Ltda., 2017. 326 p. v. 1.

CAMARA MUNICIPAL DE PATO BRANCO. **Leis ordinárias nº 3536/2011**. Disponível em: <http://www.camarapatobranco.com.br/legislacoes/lei-no-35362011>. Acesso em: 20 set. 2018.

FIREBASE. **Firebase por plataforma**. Disponível em: <https://firebase.google.com>. Acesso em: 02 out. 2018.

GARTNER. **Gartner says worldwide sales of smartphones recorded first ever decline during the fourth quarter of 2017**. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017>. Acesso em: 01 out. 2018.

LECHETA, Ricardo R. **Google Android** : Aprenda a criar aplicações para dispositivos móveis com Android SDK. 4. ed. São Paulo: Novatec, 2015. 1016 p. v. 1.

MUNICÍPIO DE PATO BRANCO. **Um jeito consciente para destinar o lixo em pato branco**. Disponível em: <http://www.patobranco.pr.gov.br/noticias/sustentabilidade/um-jeito-consciente-para-destinar-o-lixo-em-pato-branco>. Acesso em: 20 set. 2018.

STATCOUNTER. **Mobile operating system market share brazil**. 2018. Disponível em: <http://gs.statcounter.com/os-market-share/mobile/brazil>. Acesso em: 03 out. 2018.

ANEXOS

Prefeitura Municipal de Pato Branco
ESTADO DO PARANÁ
GABINETE DO PREFEITO

LEI Nº 3.536, DE 17 DE MARÇO DE 2011

Acrescenta e altera disposições à Lei n° 2.614, de 26 de abril de 2006, que proíbe a distribuição de propagandas mediante fixação de panfletos em veículos estacionados em vias e logradouros públicos e dá outras providências.

A Câmara Municipal de Pato Branco, Estado do Paraná, aprovou e eu, Prefeito Municipal, sanciono a seguinte Lei:

Art. 1º A Lei n° 2.614, de 26 de abril de 2006, passa a vigorar acrescida do art. 1°-A, com a seguinte redação:

"Art. 1°-A. Fica também proibida a distribuição de panfletos para ocupantes de automóveis aguardando o fluxo, parados em semáforos, rotatórias e em congestionamentos, em via pública do sistema viário do Município Pato Branco."

Art. 2° O art. 2° da Lei n° 2.614, de 26 de abril de 2006, passa a vigorar com a seguinte redação:

"Art. 2º Não se aplicam as disposições desta lei aos panfletos de propagandas distribuídos a pedestres, os quais obrigatoriamente deverão conter no rodapé em negrito, mensagem de cunho educativo e informativo, com os seguintes dizeres:

I – Não jogue este folheto em vias públicas;
II – Doe sangue, doe vida;
III – Colabore com a limpeza da cidade;
IV – Evite a contaminação ambiental, preserve a natureza;
V – Ajude e faça a sua parte no combate a dengue;
VI – Respeite o patrimônio público, um dia você pode precisar dele."

Art. 3º Esta lei entra em vigor na data de sua publicação.

Esta Lei decorre do projeto de lei nº 214/2010, de autoria do Vereador Osmar Braun Sobrinho – PR.

Gabinete do Prefeito Municipal de Pato Branco, 17 de março de 2011.

ROBERTO VIGANO
Prefeito Municipal

Anexo A – Lei Municipal N° 3.536.