INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JULIO CESAR OLIVEIRA DA SILVA

PHOTOCLICK: UMA APLICAÇÃO MÓVEL DE SERVIÇOS DE FOTOGRAFIA UTILIZANDO FLUTTER

Jorge Luis Boeira Bavaresco

PASSO FUNDO
2020

JULIO CESAR OLIVEIRA DA SILVA

PHOTOCLICK: UMA APLICAÇÃO MÓVEL DE SERVIÇOS DE FOTOGRAFIA UTILIZANDO FLUTTER

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Câmpus Passo Fundo, como requisito parcial para a aprovação na disciplina de Projeto de Conclusão II (PC II).

Orientador (a): Jorge Luis Boeira Bavaresco

PASSO FUNDO

2020

JULIO CESAR OLIVEIRA DA SILVA

PHOTOCLICK: UMA APLICAÇÃO MÓVEL DE SERVIÇOS DE FOTOGRAFIA UTILIZANDO FLUTTER

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____
Jorge Luis Boeira Bavaresco

_____
Lisandro Lemos Machado

_____
Carmen Vera Scorsatto

_____
Rafael Marisco Bertei

PASSO FUNDO
2020

AGRADECIMENTOS

Primeiramente gostaria de agradecer a minha família e amigos, pelo apoio, incentivo a estudar e alcançar meus objetivos e o principal: Nunca desistir, nada é impossível, com trabalho duro e dedicação tudo pode ser conquistado.

Agradeço ao professor Jorge Luis Boeira Bavaresco que, através de seus árduos conhecimentos com desenvolvimento, requisitos e sua disponibilidade para me auxiliar, contribuíram imensamente para realizar o desenvolvimento deste aplicativo, com uma nova tecnologia.

Tenho a agradecer meus colegas pela amizade, companheirismo e ajuda, onde sempre um apoiou o outro para todos conseguirem chegar até aqui, um fazendo o outro crescer.

Também gostaria de agradecer aos demais professores, todos contribuíram para elevar meus conhecimentos e ajudar a definir minha trajetória profissional, ensinando muito mais do que o conteúdo, mas também a ser uma pessoa dedicada e de cabeça erguida para enfrentar dificuldades do cotidiano.

RESUMO

A partir do contato com fotógrafos em determinada cidade, verificou-se que alguns profissionais possuem uma agenda lotada ou até mesmo com várias pessoas solicitando orçamento/valores dos serviços prestados nas redes sociais, sendo que, na maioria das vezes, a demora da resposta poderia ser evitada focando apenas em um aplicativo que faça esse processo de divulgação. O aplicativo visa entregar ao usuário a divulgação de serviço dos fotógrafos, permitindo também realizar outras funções interessantes para o usuário final, como adicionar à lista de favoritos e também adicionar à lista de contratação, desta forma favorecendo a comunicação entre o usuário e o profissional. O trabalho foi desenvolvido com o Framework Flutter criado pela Google, em conjunto com a plataforma Firebase, onde foram utilizados os serviços de banco de dados NoSQL, hospedagem de arquivos e Autenticação de usuários. O resultado obtido foi uma aplicação onde o acesso a telas e ações são controladas de acordo com o tipo do usuário que está autenticado (Fotógrafo ou usuário comum), tendo a possibilidade de listar os Banners/Publicações dos fotógrafos de acordo com sua categoria de foto, podendo adicionar estas na listagem dos favoritos e também na listagem de contratações. Os testes foram realizados apenas por um usuário Fotógrafo através do emulador do Android Studio, para averiguar se o usuário conseguiria identificar o intuito de cada botão/ícone posicionado na tela, onde retornou de forma positiva a organização de todas as telas.

Palavras-chave: Aplicativo. Flutter. Fotografia. Firebase.

ABSTRACT

From the contact with photographers in a given city, it was found that some professionals have a busy schedule or even with several people requesting quotes/values for services provided through social networks, and, in most cases, the delay in response could be avoided by focusing only on an application that does this disclosure process. The proposed application aims to deliver the photographers' service disclosure to the user, allowing also to perform other interesting functions for the end user, such as adding to the favorites list and also adding to the hiring list, untangling favoring communication between the user and the professional. The work was developed with the Flutter Framework created by Google, together with the Firebase platform, where NoSQL database services, file hosting and User Authentication were used. The result obtained was an application where access to screens and actions are controlled according to the type of user who is authenticated (Photographer or ordinary user), with the possibility of listing the Banners/Posts of the photographers according to their photo category and you can add these to the favorites list and also to the hiring list.

Keywords: Application. Flutter. Photography. Firebase.

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES E DE SIGLAS

APP – Application (Aplicação)

UML – Unified Modeling Language (Linguagem Unificada de Modelagem)

IU – User Interface (Interface do Utilizador)

FCM - Firebase Cloud Messaging

PHP - Hypertext Preprocessor

SUMÁRIO

1. INTRODUÇÃO

O curso Tecnologia de Sistemas para Internet (TSPI) do Instituto Federal Sul-Rio-Grandense (IFSUL) solicita, para encerrar e por fim obter a formação, um Projeto de Conclusão II (PC2) em que o aluno possui a possibilidade de escolher um tema ou assunto que deseja se basear para desenvolver e realizar a apresentação mediante a aprovação, onde o tema escolhido foi desenvolvimento de uma aplicação móvel para fotógrafos utilizando Flutter.

A partir do contato com fotógrafos em determinada cidade, verificou-se que alguns possuem uma agenda lotada ou até mesmo com várias pessoas solicitando orçamento/valores dos serviços prestados através de redes sociais, sendo que, na maioria das vezes, a demora da resposta poderia ser evitada focando apenas em um aplicativo que faça esse processo de divulgação.

O alcance do smartphone no Brasil continua crescendo no País, cada vez mais as pessoas utilizam esse aparelho tanto na vida pessoal quanto no trabalho. Segundo a pesquisa Global Mobile Consumer Survey 2018, realizada pela Delloite - líder em serviços de Auditoria - estima-se que:

> Dados do IBGE1 indicam que em 92% das casas brasileiras há pelo menos um telefone móvel. Ainda de acordo com a pesquisa, no País, o celular já é o equipamento mais utilizado para o acesso à internet (95%), tomando a frente do computador (64%). Portanto, falar de telefonia móvel no Brasil é falar de um país continental, fortemente regulamentado, repleto de oportunidades e com uma população apaixonada pela conectividade (OGAWA, 2018).

O aplicativo desenvolvido seguiu a mesma linha de raciocínio de outros que já existem no mercado, como por exemplo o Uber e o *iFood*, no qual se busca um responsável ou profissional para exercer um trabalho ou serviço.

Após pesquisa realizada com uma fotógrafa para levantamento de requisitos, foi alcançado e apontado qual é o principal ponto para ser implementado na nova ferramenta: a facilidade de comunicação entre as duas partes, já que hoje ao se realizar pesquisa por um aplicativo que faça algo semelhante nesta área, obtiveram-se apenas ferramentas de controle de agenda.

O objetivo é facilitar a comunicação para o fotógrafo e o cliente, já que uma das formas mais empregadas para comunicação é o celular e atualmente esse processo de divulgação geralmente é feito através de redes socias.

Desta forma, se chegou a seguinte questão: Como dar praticidade para quem precisa contratar um fotógrafo e como facilitar para quem precisa divulgar/vender seu serviço, visto que precisa ser algo simples para manuseio, mas de boa usabilidade? Para resolver este problema, foi realizado o desenvolvimento de um aplicativo Mobile utilizando a tecnologia da google denominada Flutter, que visa obter informações de um serviço realizado por fotógrafos, favorecendo a comunicação entre o usuário que deseja contratar um fotógrafo e o profissional fotógrafo, com o fim de facilitar o contato entre os dois meios entre outras funções.

1.1. OBJETIVOS

Nesta seção, serão destacados os objetivos do projeto, separados em dois subitens: Objetivo Geral e Objetivos Específicos.

1.1.1. Objetivo geral

Modelar, desenvolver e implementar o Aplicativo hibrido, de solicitação/contratação de serviços fotográficos com interação entre usuário e prestador do serviço, com o intuito de realizar a fácil comunicação entre ambos.

1.1.2. Objetivos específicos

- Realizar seu cadastro e *login*, utilizando *Authentication*;
- Criar, visualizar, editar e remover anúncios de serviços – permitido somente para o Fotógrafo - com apresentação conforme categoria especificada pelo usuário – Cliente.
- Implementar o modelo lógico do sistema, analisando todas as situações possíveis que podem ocorrer dentro do projeto.
- Separar o acesso do fotógrafo e do contratante, pois cada um terá um objetivo específico no manuseio do aplicativo.

- Expandir anúncios, permitindo obter maior conhecimento sobre o profissional que deseja contratar, como por exemplo fotos, descrição, preço, etc.
- Utilizar a linguagem Flutter da Google, assim como as ferramentas que o Firebase entrega.

2. REFERENCIAL TEÓRICO

Nas próximas seções serão apresentados os temas e assuntos que auxiliaram no resultado para o conhecimento e realização do trabalho.

2.1. Tecnologias utilizadas

A seguir serão apresentadas as tecnologias que foram utilizadas para o desenvolvimento da aplicação móvel onde contará com um breve esclarecimento sobre as mesmas.

2.1.1. Firebase

Firebase é uma plataforma da Google, onde nela o usuário pode usufruir de suas ferramentas de desenvolvimento, para dispositivos móveis e *web*. Serve, entre outros fatores, para ajudar desenvolvedores web e mobile a criar aplicações de alta qualidade e performance.

A Google transformou essa plataforma em uma solução completa e compacta de *back-end,* tanto para desenvolvimento mobile como também para *web*. A plataforma Possui um SDK e um console para criar aplicações, facilitando o entendimento do processo a ser realizado (AVRAM, 2019).

Para utilização, foi dividido em três planos desta ferramenta: Plano *Spark* (plano gratuito), Plano *Flame* (Preço fixo para apps em expansão) e o plano Bronze (Pagamento por utilização).

No Plano *Spark*, utilizado para o desenvolvimento do aplicativo móvel, contêm os seguintes serviços do Firebase (2019):

- **Test A/B:** Com esse recurso, se torna possível executar, analisar e escalonar experimentos de produtos e de *marketing*. Também é possível testar

campanhas de engajamento para ver se elas estão funcionando conforme o esperado.

- **Google Analytics:** O núcleo do Firebase é o Google *Analytics*, uma solução de análise gratuita e ilimitada. Os recursos do Firebase são integrados ao *Analytics*, que fornece geração ilimitada de relatórios para até 500 eventos distintos.
- **App Indexing (Indexação de apps no Firebase):** O app com essa ferramenta é levado para a Pesquisa Google pela Indexação de apps no Firebase. Se os usuários possuem o app instalado, podem inicializá-lo e ir diretamente para o conteúdo que estão procurando.
- **Firebase Authentication:** Fornece serviços de *back-end*, fáceis de usar e bibliotecas de *User Interface* (IU), prontas para autenticar usuários no seu app. Ele oferece suporte à autenticação por meio de senhas, números de telefone e provedores de identidade federados como Google, Facebook, Twitter e muito mais.
- **Cloud Messaging (FCM):** O *Firebase Cloud Messaging* (FCM) é uma solução de mensagens entre plataformas que permite o envio confiável de notificações sem custo.
- **Crashlytics:** É uma ferramenta de relatório de falhas e em tempo real que ajuda a monitorar, priorizar e corrigir problemas de estabilidade que comprometem a qualidade do seu aplicativo.
- **Dynamic Links:** Adapta os links de acordo com a plataforma em que forem abrir.
- **Remote config:** O Configuração remota do Firebase é um serviço em nuvem que permite a alteração do comportamento e da aparência do app sem exigir que os usuários façam o *download* de uma atualização.
- **Monitoramento de desempenho:** Auxilia a receber *insights* sobre as

características de desempenho dos apps iOS, Android e da Web.

- **Predictions:** Aplica o *machine learning* aos dados de análise para criar segmentos dinâmicos de usuários com base no comportamento previsto.
- **Invites:** Trata-se de uma solução pronta para uso para indicações e compartilhamento de apps por e-mail ou SMS.

Além desses serviços, também é disponibilizado o Realtime Database, Cloud Firestore, Storage, Cloud Functions, Phone Auth, Hospedagem, Test Lab, ML Kit e Google Cloud Plataform, onde a maioria desses são gratuitos, porém com limite de acessos (Firebase, 2019).

Essa ferramenta tem tudo para se tornar uma das mais promissoras no mercado, porém, como destaca o blog rocketseat, o Firebase possui algumas desvantagens, como: Controle e Acesso (limitação de números de acesso de acordo com o serviço utilizado), Limitação da Plataforma, Documentação (documentações do Firebase deixam a desejar no quesito de exemplos práticos). Mas possui vantagens, onde torna-se muito mais promissor, como: Estrutura Pronta, Rápida Implementação, Segurança, Múltiplas Ferramentas,Facilmente Escalável (rocketseat, 2019).

2.1.2. Flutter

O Flutter é o framework móvel do Google, assim como o Firebase, que permite construir aplicativos Android e iOS com apenas um código. Ele também é código aberto e gratuito, onde a versão 1.0 foi lançada em 5 de dezembro de 2019.

Para os usuários, o Flutter entrega uma boa usabilidade nos aplicativos. Para os desenvolvedores, o Flutter reduz a barra de entrada para a criação de aplicativos móveis, acelera o desenvolvimento e reduz o custo e a complexidade da produção de aplicativos nas plataformas. Já aos designers, o Flutter ajuda a

fornecer a visão original do design, sem perda de fidelidade ou comprometimento. Ele também atua como uma ferramenta produtiva de prototipagem (Flutter, 2019).

Ele é voltado para desenvolvedores que, de maneira simples e prática, desejam criar aplicativos móveis de boa usabilidade (e alcançar mais usuários com uma única ferramenta).

Permite aos programadores criarem aplicativos móveis e enviar o mesmo conjunto de recursos tanto para IOS como para Android, fazendo com que sejam reduzidas as manutenções, por exemplo. Embora o público alvo inicial desta ferramenta não sejam os *designers*, o *framework* consegue entregar aos profissionais que desejam que suas visões originais sejam entregues de forma consistente, com alta usabilidade a todos usuários no celular.

Basicamente, o Flutter é divido em 3 camadas em sua *Engine*. A primeira é responsável por todo o *Framework* Flutter escrito em Dart onde é realizado o desenvolvimento. A Segunda, é algo que dificilmente deve-se preocupar, pois se trata do *core* do Flutter, onde o cérebro dele está, pois lá estará tratamentos específicos de cada sistema e a *engine* gráfica (OpenGL ES), que é o diferencial do Flutter (MAGALHÃES, 2019).

2.1.3. SERVQUAL

A visão dos clientes quanto a qualidade dos serviços pode ser avaliada com uma ferramenta de análise chamada SERVQUAL. Foi proposta por Parasuraman, Zeithaml e Berry (1985), cujo método afirma existir um espaço entre as expectativas dos clientes e o serviço que foi prestado.

O SERVQUAL mede a qualidade do serviço baseando-se nas expectativas do cliente em contraponto com a percepção que esse mesmo cliente tem em relação ao serviço que recebeu.

Na prestação do serviço deve-se aplicar essa metodologia em que ponto e como melhorar. Por exemplo, ao comprar um produto, o mesmo receberá uma avaliação no momento (visualmente), podendo receber um *feedback* sobre ele, isso acontece porque o produto é visível, podendo olhar e avaliar a qualidade. Se não for o que você esperava, poderá apontar as falhas, caso contrário poderá

elogiar e levantar as qualidades, de qualquer forma estará recebendo um *feedback*, usufruindo dele para então melhorar.

Já no serviço, a insatisfação se torna inviável ser apontada como se faz com o produto. Às vezes, o cliente gosta ou não do serviço, mas não consegue expressar de forma exata sua opinião. Isso que prejudica a tomada de decisões e dificulta tomar atitudes para melhorar o serviço.

A metodologia busca saber quais fatores o cliente considera mais importantes na prestação de um serviço. Dividido em duas etapas, o método consiste em 2 entrevistas que devem ser realizadas com vários clientes.

Feitas as entrevistas, basta pegar a avaliação que o cliente deu ao seu serviço (2ª entrevista) e subtrair a nota que ele atribuiu ao que considerava ideal (1ª entrevista).

O resultado será o nível de qualidade do seu serviço na visão dos seus clientes. Resultados positivos significam que a qualidade está acima do esperado pelo cliente, resultados negativos significam que seu serviço está deixando a desejar naquele requisito (RAMOS, 2017).

O SERVQUAL vai ser utilizado em trabalhos futuros, devido a não ter tido tempo hábil, já que para aplicar este método é necessário que o aplicativo esteja sendo utilizado por alguns usuários.

3. METODOLOGIA

Este capítulo irá tratar a modelagem e desenvolvimento do aplicativo que foi desenvolvido onde realiza a divulgação dos serviços de fotógrafos com suas ações, bem como o estudo de caso feito.

No desenvolvimento do sistema em questão, para ter acesso total as funcionalidades da aplicação, o usuário precisa realizar sua autenticação por E-mail/Senha. Se for seu primeiro acesso, o aplicativo irá redirecionar para a tela de registro, nesta o usuário irá selecionar se é fotógrafo ou não. Se não for selecionado “Sou fotógrafo”, irá exibir uma seção para definir gostos e opções para um futuro motor de busca para as características dos fotógrafos. Se for selecionado “Sou fotógrafo”, irá direcionar para criação do seu perfil, informando qualidades, preferências de ensaios, após isso poderá criar um *Banner* para ser exibido na aplicação, de acordo com a categoria selecionada.

Depois disso, o cliente será direcionado para a *Dashboard* da aplicação, onde terá a exibição dos Banners fotográficos. O cliente também poderá adicionar Banners a sua lista, para desta forma não perder o mesmo de vista na listagem dos demais.

As etapas a seguir serão focadas para a modelagem e desenvolvimento da aplicação e detalhadas conforme andamento.

3.1. Análise

Nesta etapa serão aprestados os aplicativos que possuem a mesma metodologia da proposta apresentada e o levantamento de necessidades que foi realizada para o desenvolvimento da aplicação, visto que através de pesquisas não foram encontrados aplicativos já existentes que entregam o mesmo resultado. Entre os pesquisados, foram selecionados o iFood (App Store, 2019), e o Uber (App Store, 2019).

3.1.1. Análise do contexto atual

Após analisar o mercado em relação a aplicativos para fotógrafos, foi identificado que atualmente existem aplicativos mais voltados para controle de gestão, tendo isso em vista surgiu a ideia para desenvolver uma aplicação para

divulgar os serviços para usuários que buscam um fotógrafo para registrar um momento especial de sua vida. Outras funcionalidades para a aplicação foram identificadas, como por exemplo adicionar uma promoção na lista de favoritos, para conferir mais tarde sem precisar acessar novamente a listagem das publicações.

Para estudar melhor a ideia e identificar melhorias a serem desenvolvidas no levantamento de requisitos, foi realizado o contato com dois fotógrafos. Eles confirmaram que as principais dificuldades são para concentrar a divulgação em apenas um local, já que as mesmas são realizadas em redes sociais distintas. Porém para isso acontecer é necessário ganhar seguidor em seu perfil profissional, já que os usuários poderão ver sua publicação apenas depois de seguir seu perfil.

Também relataram que possuem uma agenda lotada, não dando a devida atenção através do chat das redes sociais. Na maioria das vezes, a demora da resposta poderia ser evitada focando apenas em um aplicativo que faça esse processo de divulgação, tendo as informações necessárias para não ter dúvidas sobre o trabalho exercido pelo profissional.

3.1.2. iFood

O iFood oferece um serviço acessível de entrega de comidas via aplicativo. Por meio do app, é possível consultar uma lista de restaurantes próximos à sua localização e seus respectivos produtos. Ao concluir um pedido, a comida é entregue na localização escolhida pelo cliente.

Após baixar o aplicativo e iniciá-lo, o mesmo trabalha com autenticação, solicitando o usuário a realizar o *login* pelo Facebook ou Gmail. Após realizar o Login, é solicitado ao usuário para permitir que o aplicativo identifique sua localização ou então pode ser pesquisado manualmente, para que o iFood consiga apresentar as ofertas na localidade.

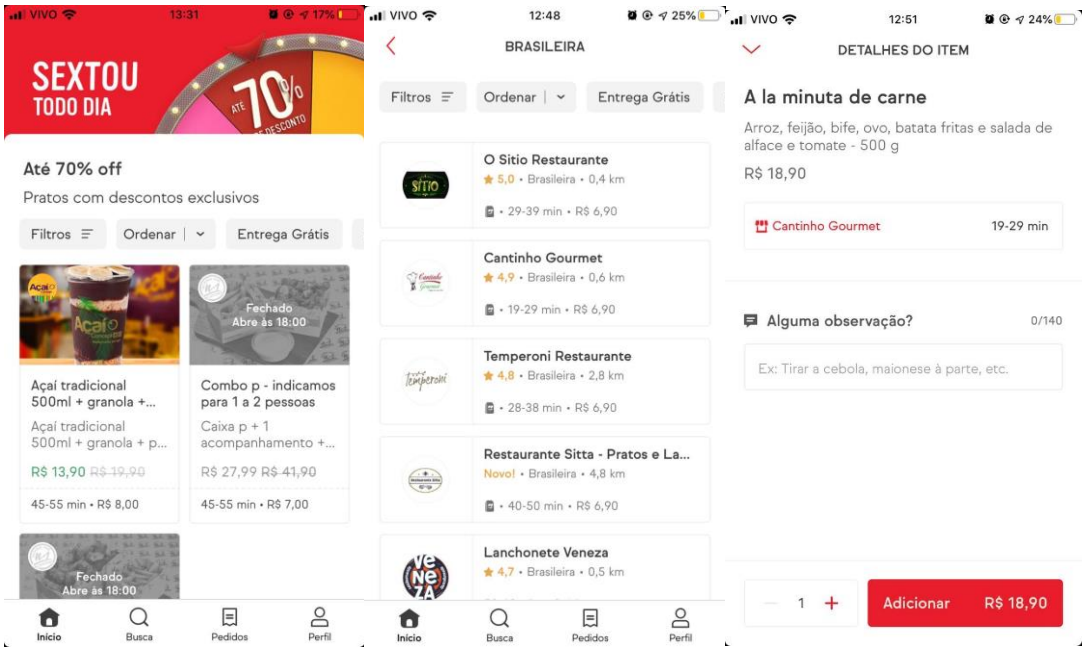
Em sua tela inicial, é exibido alguns catálogos de ofertas, como por exemplo “pratos com descontos”, “Frete Grátis”, entre outros, conforme apresentado na Figura 1.

Figura 1 - Capturas de tela iFood - Inicial

Fonte: Aplicativo iFood, 2019

Na sequência, ao selecionar um catálogo ou realizar uma busca, será apresentado as opções de acordo com o selecionado, onde você pode então ver detalhes do que deseja comprar, e após isso realizar a compra, conforme destacado na Figura 2.

Figura 2 - Capturas de tela iFood – Filtros/finalização



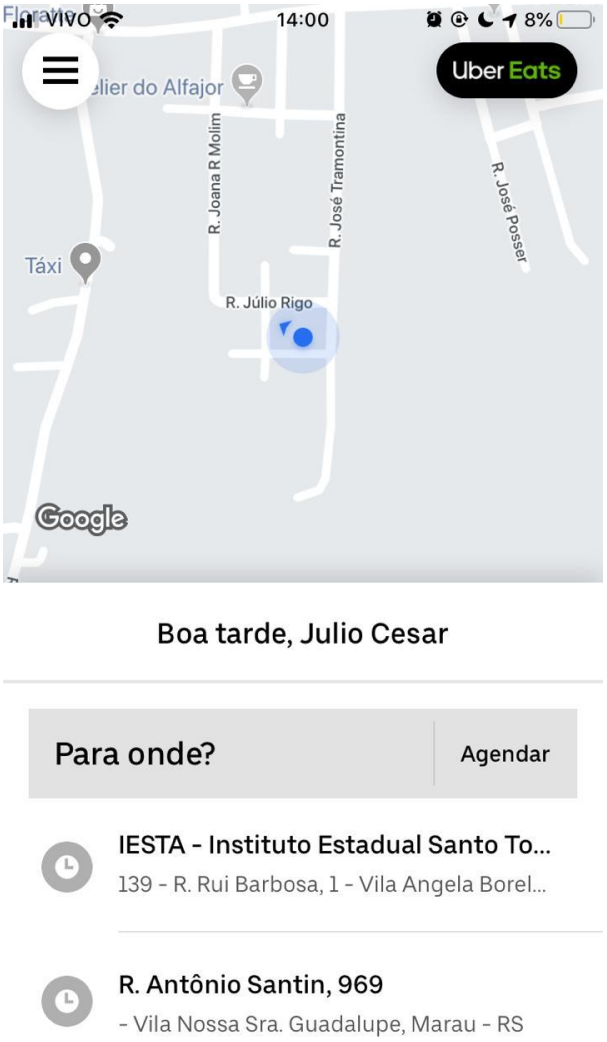
Fonte: Aplicativo iFood, 2019

3.1.3. Uber

A Uber é um aplicativo que está transformando a maneira como as pessoas se movimentam pelas cidades. Ao conectar, de forma simples, motoristas parceiros e usuários, ajudando a deixar cidades mais acessíveis, oferecendo mais opções para usuários e mais oportunidades de negócios para motoristas parceiros.

Após realizar o Login no app, se torna necessário permitir que o mesmo visualize a sua localização – obrigatório para localizar motoristas – antes de começar a utilizá-lo. Após isso, o usuário é direcionado para a página inicial, que apresenta um mapa da sua localização conforme a Figura 3.

Figura 3 - Capturas de tela Uber - Inicial



Fonte: Aplicativo Uber, 2019

Agora, existem duas formas para contratar o serviço de um motorista: Informando o seu destino (percurso), ou então realizando um agendamento de um serviço. Sendo assim, o aplicativo apresentará a localização do motorista, quanto irá custar o serviço e também a forma de pagamento, conforme a Figura 4.

Figura 4 - Capturas de tela Uber - Serviço

Fonte: Aplicativo Uber, 2019

3.2. Modelagem

Nas próximas seções serão apresentados os diagramas representando a aplicação proposta, tendo a amostra também das interações perante o usuário com o sistema.

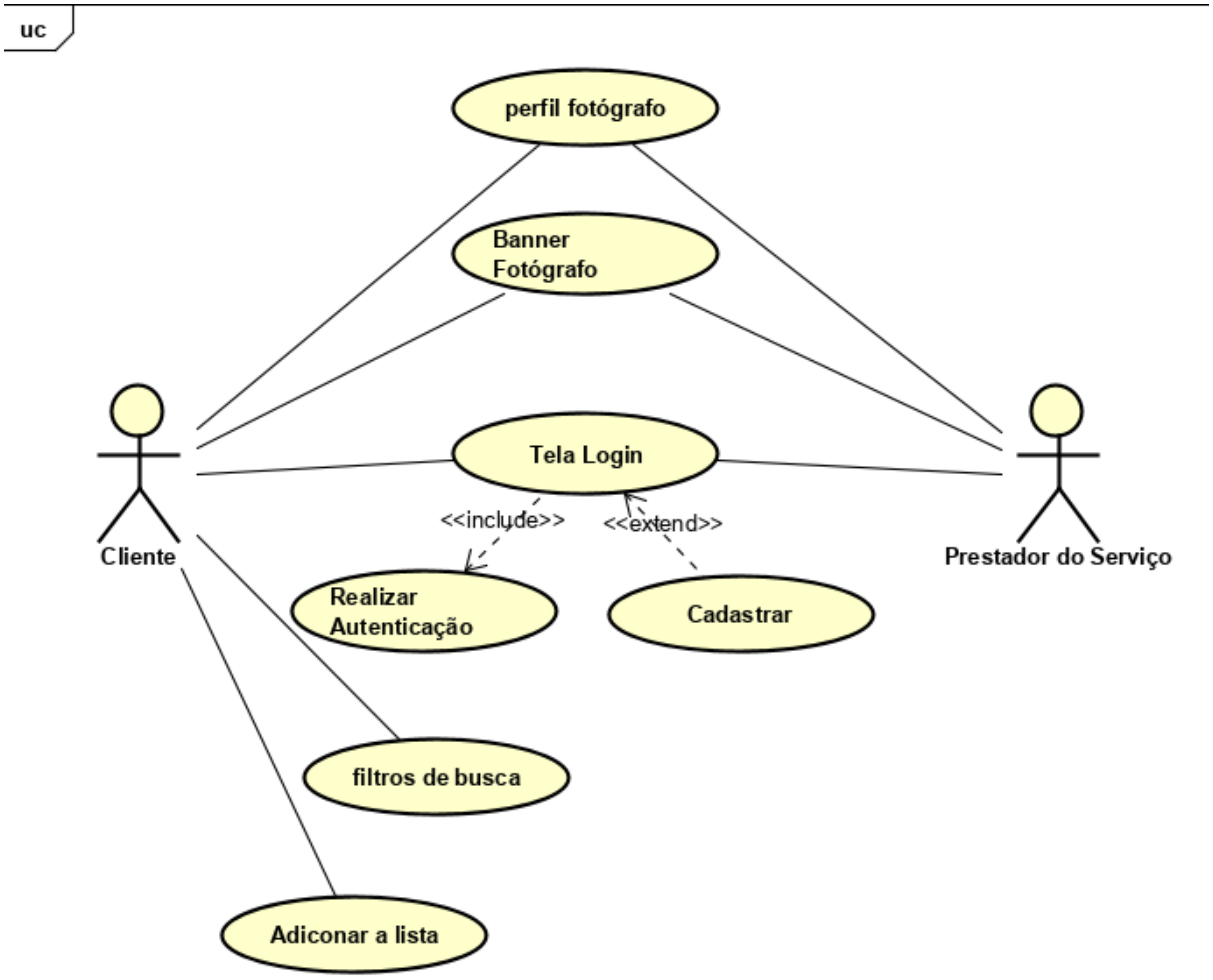
3.3. Diagrama de Casos de Uso

Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz (RIBEIRO, 2012).

Esse diagrama representa a interação entre um ator e o aplicativo/sistema. Alguns exemplos práticos de assimilar sobre caso de uso: login em um sistema, criar publicação, inserir pedidos. Cada caso de uso tem uma descrição de como irá funcionar cada parte do aplicativo ou sistema em questão.

A Figura 5 representa o caso de uso desta aplicação.

Figura 5 – Diagrama Caso de Uso



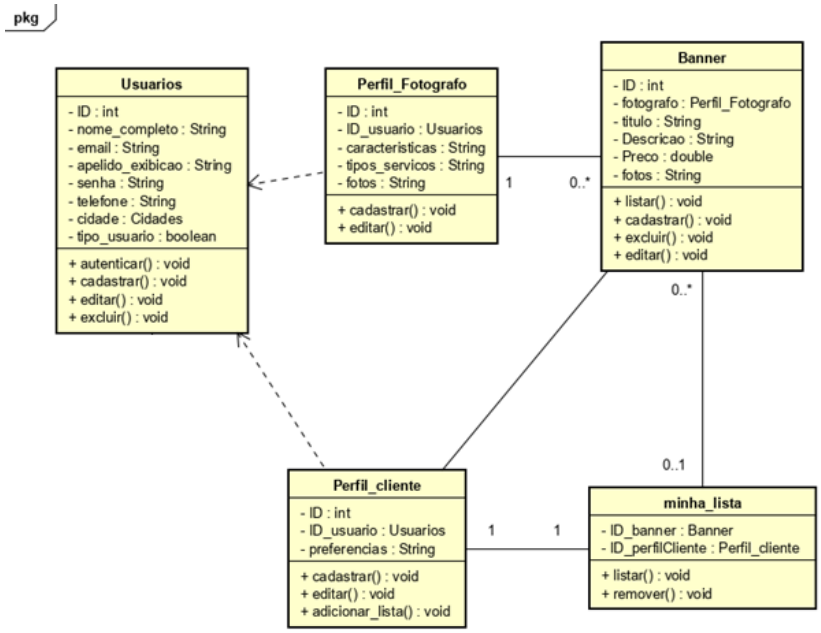
Fonte: Do autor, 2019

3.4. Diagrama de Classes

O diagrama de classes é considerado por muitos autores como o mais importante e o mais utilizado diagrama da UML. Seu objetivo é permitir a visualização das classes que irão compor o sistema com seus respectivos atributos e métodos, bem como em demonstrar como as classes do sistema se relacionam, se complementam e transmitem informações entre si. Este diagrama apresenta uma visão estática de como as classes estão organizadas, preocupando-se em definir a estrutura lógica das mesmas. O diagrama de classes serve como base para a construção da maior parte dos demais diagramas da UML (SILVA, 2009).

O diagrama de classes apresentado a seguir na Figura 6, representa a estrutura dos dados assim como suas ligações, de acordo com necessidade levantada para utilização da aplicação. A classe Usuários representa os usuários do tipo fotógrafo e cliente. Ambos possuem ligação com a classe Banner, que podem ser cadastradas, editadas e excluídas pelo Fotógrafo, e o cliente pode visualizar e adicionar a lista de Favoritos.

Figura 6 - Diagrama de Classes



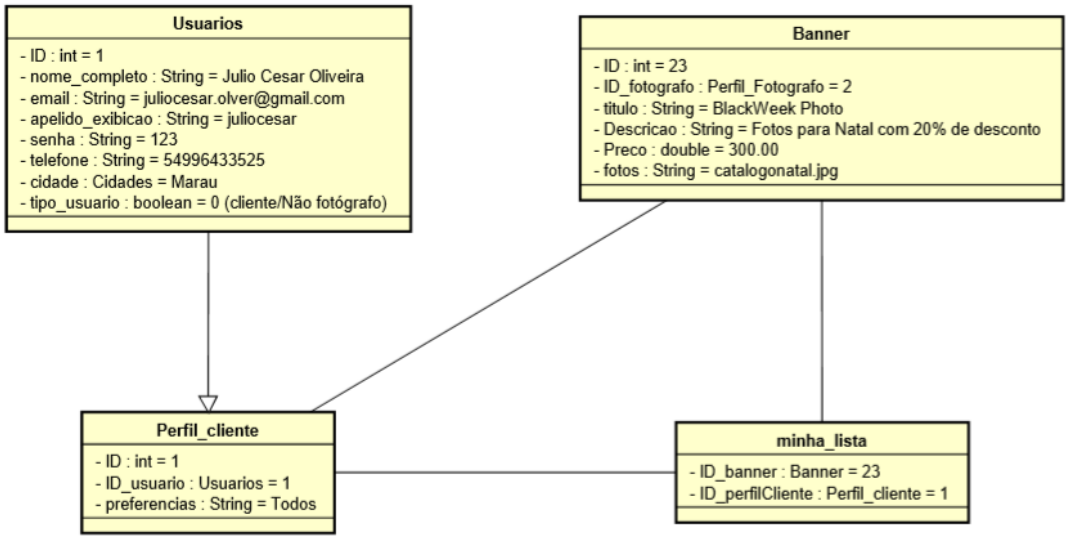
Fonte: Do autor, 2019

3.5. Diagrama de Objetos

O diagrama de objetos é uma variação do diagrama de classes e utiliza quase a mesma notação. A diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes.

Na próxima imagem (Figura 7) é apresentado o diagrama de objetos criado para exibir, em tempo de execução, os objetos a aplicação, junto de seus inter-relacionamentos com as outras entidades.

Figura 7 – Diagrama de Objetos (cliente)



Fonte: Do autor, 2019

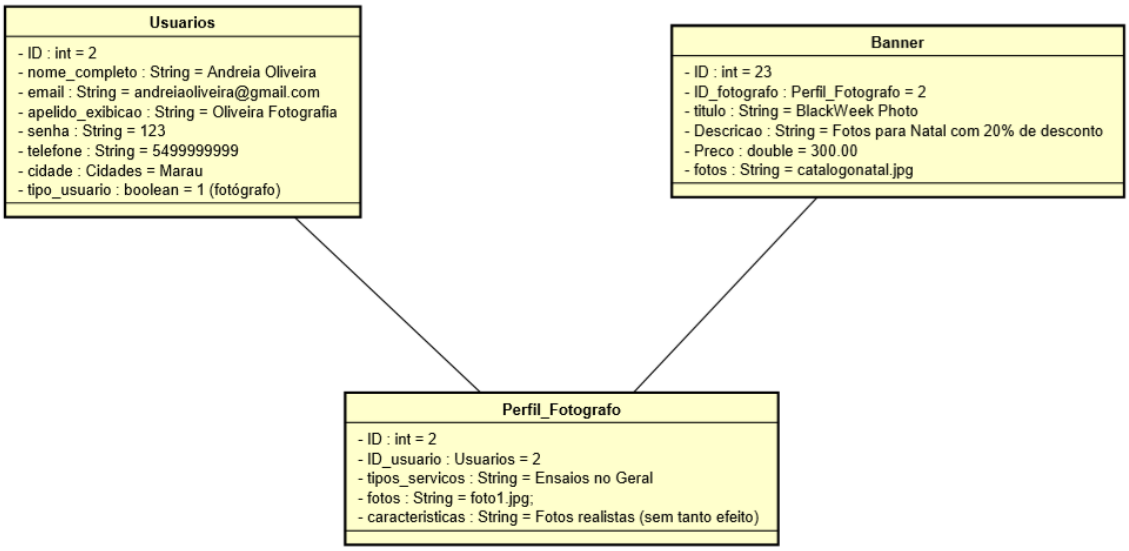
Este diagrama de objeto trata do manuseio do aplicativo perante a um usuário que seria o cliente, onde a tabela “Usuários” refere-se as informações que ambos usuários (fotógrafo e cliente) tem em comum. Já a “perfil_cliente” é específica para o cliente, tendo em consideração que são campos que diferem da tabela “perfil_fotografo”.

O mesmo terá acesso a listagem dos banners que são registrados pelo

fotógrafo, e após isso o usuário poderá adicionar a “minha lista” os banners que mais lhe chamar a atenção.

Já na próxima imagem (Figura 8), encontra-se o diagrama de objetos de usuário quando fotógrafo.

Figura 8 – Diagrama de Objetos (fotógrafo)



Fonte: Do autor, 2019

Neste diagrama de objetos, está ilustrado o manuseio do usuário fotógrafo, onde a tabela “usuarios” são os campos existentes em comum, quando trata-se dos usuários no geral. O fotógrafo terá acesso ao “perfil_fotografo”, que desta forma registrará informações importantes sobre seu trabalho e afins.

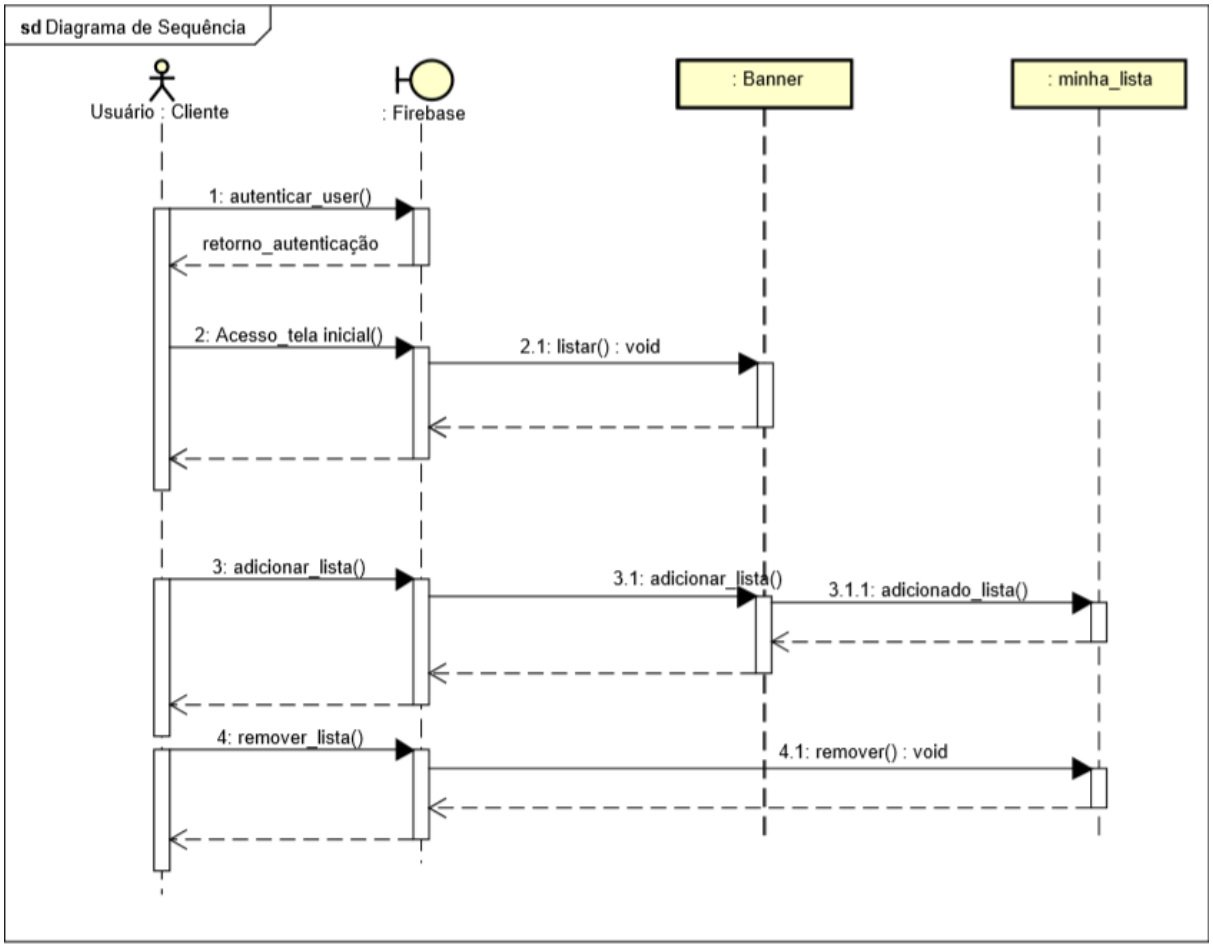
Logo após, o usuário poderá criar seus Banners, adicionando informações importantes para que o cliente possa ver e entrar em contato após aprovar o mesmo.

3.6. Diagrama de Sequência

Um diagrama de sequência descreve a maneira como os grupos de objetos colaboram em algum comportamento ao longo do tempo. Ele registra o comportamento de um único caso de uso e exibe os objetos e as mensagens passadas entre esses objetos no caso de uso (Wikipedia, 2019).

O diagrama de sequência a seguir (Figura 9) representa a unificação do processo entre os objetos, de acordo com a proposta da aplicação, onde o usuário para executar qualquer ação deverá estar autenticado com o Firebase, onde este retorna se a autenticação foi realizada com sucesso. Logo após isso, é concedido o acesso as funções dentro do Banner, assim como criar um novo Banner.

Figura 9 – Diagrama de Sequência



Fonte: Do autor, 2019

4. DESENVOLVIMENTO DO APLICATIVO

Este capítulo descreve o processo utilizado para o desenvolvimento da solução proposta, a estrutura das camadas da aplicação e os recursos utilizados para o seu funcionamento.

No aplicativo foi desenvolvido uma tela inicial denominada *Dashboard,* sua função é reunir todos os Banners cadastrados pelo fotógrafo, em formato de mosaico. O usuário poderá acessar e visualizar as informações dos Banners através do menu, sem a necessidade de estar autenticado, assim como visualizar os fotógrafos cadastrados.

Para cadastrar um novo Banner, o usuário deve ser do tipo fotógrafo, opção que deve ser selecionada ao realizar o cadastro no aplicativo. O Fotógrafo poderá visualizar todas as publicações, porém poderá editar e/ou excluir apenas as suas.

Foi desenvolvido ações para realizar dentro de um Banner, como adicionar aos favoritos e adicionar a tela de contratação, porém ambas ações se tornam necessárias que o usuário esteja autenticado para realizar a ação. Após adicionar um Banner a lista de favoritos ou de contratados, o usuário terá acesso a ambas as telas através do menu, podendo listar os itens favoritos, tendo a opção de remover. Já a tela de contratação, são exibidos detalhes da publicação escolhida, como valor e descrição. Essa tela ainda não possui um sistema de finalização de contrato, porém contém a simulação de aguardo e finalização da solicitação.

4.1. FERRAMENTAS DE DESENVOLVIMENTO E RECURSOS

Esta seção apresenta as ferramentas e recursos utilizados no desenvolvimento da aplicação.

4.1.1. IDE (Integrated Development Environment) e Servidores

Para o desenvolvimento do sistema foi utilizado a aplicação IDE (Ambiente de Desenvolvimento Integrado), a qual foi necessário para a escrita do código e execução do emulador Android. A IDE escolhida foi o Android Studio, por sua compatibilidade com desenvolvimento Mobile, especificamente com suporte a Flutter.

O gerenciamento do banco de dados foi feito através do Firebase, mais especificamente Cloud Firestore – um banco NoSQL.

4.1.2. Bibliotecas e Frameworks

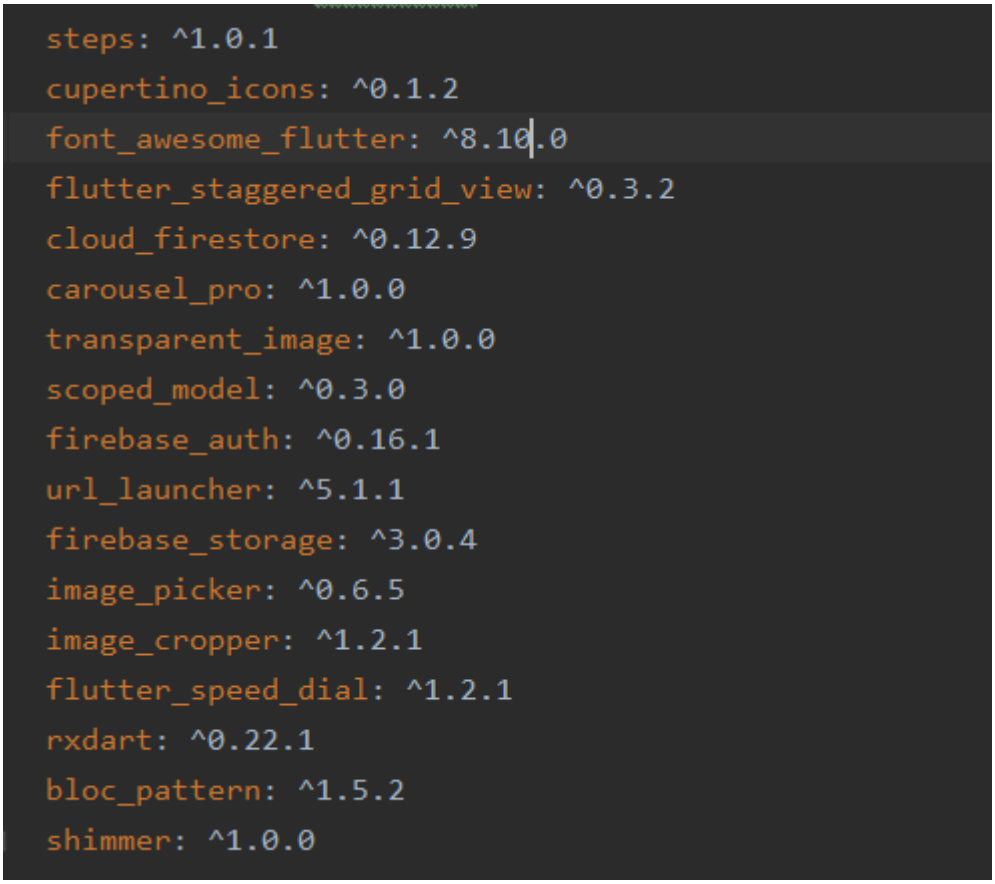
Para a construção do aplicativo foram utilizados recursos de diferentes bibliotecas. Os recursos foram aplicados para auxiliar na criação da interface de usuário, para realizar o *upload* e recorte da imagem, para gravar os dados e tratar da comunicação com a aplicação, entre o banco de dados e o Aplicativo. As bibliotecas empregadas na aplicação são:

- **Biblioteca Cupertino Icons ^0.1.2:** Foi utilizada para adicionar ícones ao aplicativo, em opções de texto, menus e também em forma de botão, dando um destaque maior em questão de atratividade.
- **Biblioteca Flutter staggered grid view ^0.3.2:** A Biblioteca *Flutter Staggered Grid View* é uma biblioteca utilizada para organizar uma tela em formato de mosaico, por exemplo, com fotos em diferentes tamanhos, mas uma ligada a outra, em forma de retângulo. Neste projeto foi utilizada para desenvolver a tela inicial do aplicativo (*Dashboard*).
- **Biblioteca Cloud Firestore ^0.12.9:** O Cloud Firestore é um banco de dados NoSQL hospedado na nuvem que os apps do iOS, do Android e da Web podem acessar diretamente por meio de *SDKs* nativos. A biblioteca tem o propósito de utilizar a *API* Cloud Firestore, da Google. Entrega uma facilidade para realizar a integração do Flutter com o Banco de Dados, onde após configurar o projeto apontando para o banco de dados, pode utilizar as funcionalidades com apenas alguns comandos.
- **Carousel Pro ^1.0.0:** A bliblioteca *Carousel Pro* tem a finalidade de exibir as imagens em forma de Slides, podendo configurar o tempo de duração que cada imagem é exibida (no caso de conter mais de uma), o tamanho, tipo de animação, entre outras funcionalidades.

- **Transparent Image ^1.0.0:** A biblioteca *Transparent Image*, é uma biblioteca simples onde seu único papel é adicionar uma imagem transparente a um *Widget* do Flutter.
- **Firebase Auth ^0.16.1:** Biblioteca responsável por realizar a autenticação de um usuário com o Firebase *Autentication*: sendo ele um cadastro de novo usuário, ou login.
- **Firebase Storage ^3.0.4:** O Firebase Storage é um serviço de armazenamento de objetos que pode ser acessado via Google Cloud Platform, ele foi criado para os desenvolvedores de aplicativos armazenarem o conteúdo gerado pelo usuário como, por exemplo, fotos ou vídeos.
- **Image Picker ^0.6.5:** Biblioteca responsável por integrar o aplicativo com as funções da câmera ou galeria do celular, possibilitando tirar uma foto ou carregar uma imagem existente.
- **Image Cropper ^1.2.1:** Esta biblioteca foi utilizada para realizar o recorte e edição de uma imagem, após selecionar no momento do upload.
- **Shimmer ^1.0.0:** Esta biblioteca fornece uma maneira fácil de adicionar efeito de brilho no projeto Flutter, foi utilizada então para dar um efeito de “carregando”, ou seja, enquanto busca resultados no banco de dados, cria uma lista de itens com efeito de brilho.

Na Figura 10 podemos verificar as bibliotecas utilizadas para o desenvolvimento do aplicativo.

Figura 10 - Bibliotecas do Projeto



Fonte: Do autor, 2020

4.2. Estrutura da Aplicação

Esta seção apresenta a arquitetura detalhada das camadas da aplicação, e os componentes que foram utilizados na criação do sistema.

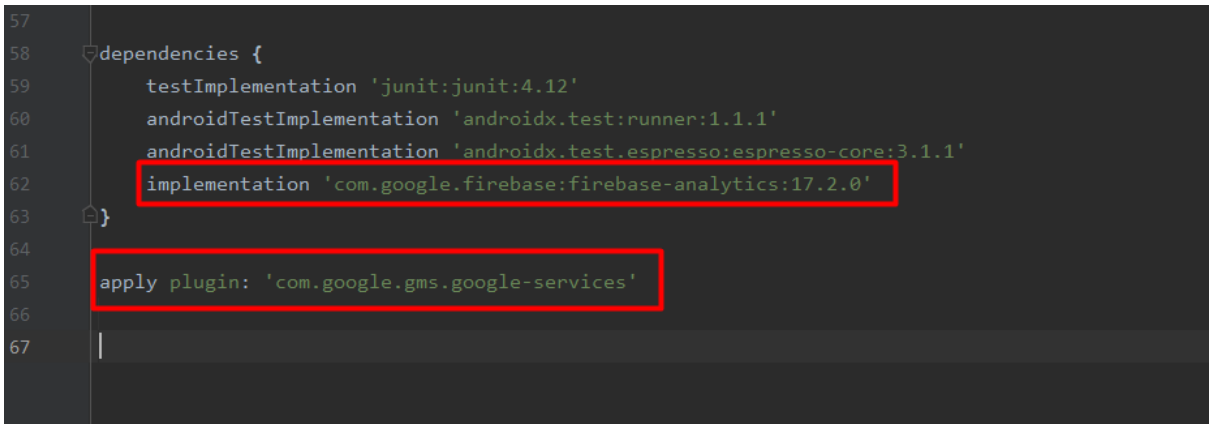
4.2.1. Integração com o Firebase

A primeira etapa do desenvolvimento foi a configuração e integração do aplicativo Flutter com o Firebase, onde foram utilizados alguns recursos do mesmo, então além da biblioteca que foi importada, também foi necessário realizar a configuração.

Após criar uma conta no Firebase e ativar o que será utilizado na aplicação, foi necessário vincular o aplicativo desenvolvido na aplicação do Firebase, através do package. Após realizar este processo, a Google disponibiliza um arquivo ***google-services.json*** para adicionar no projeto flutter no diretório ***android/app***, desta forma

iniciando o processo de integração. Após isso, deve-se adicionar no arquivo **build.gradle** o seguinte código (Figura 11):

Figura 11 - Implementação Firebase no projeto



Fonte: Do autor, 2020

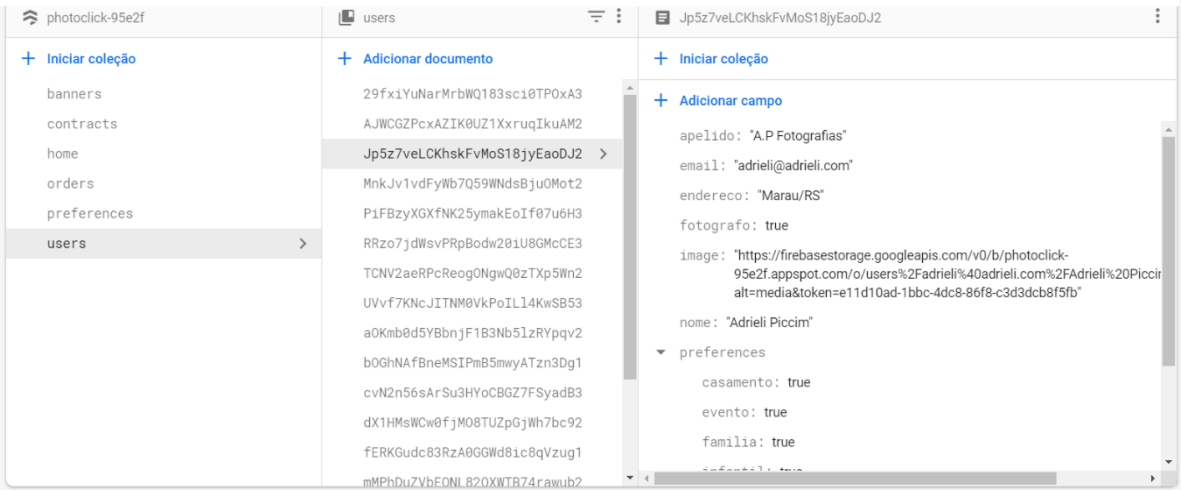
Este arquivo possui os dados referentes as configurações dos projetos, onde verifica se o SDK do flutter está configurado, o idioma, entre outras configurações que é gerado por padrão junto com a estrutura inicial.

4.2.1.1. Estrutura e Configuração do Banco de Dados

O banco de dados do projeto é o *Firebase*. Ele é um banco on-line com versão gratuita, de propriedade da Google. O *Firebase* usa a infraestrutura do Google e é dimensionado automaticamente, uma funcionalidade que traz segurança e praticidade para o desenvolvedor e os usuários das aplicações desenvolvidas. Este banco de dados é NoSQL, o que significa que não há tabelas ou linhas, os dados são armazenados em documentos, que são organizados em coleções. Para utilizar o banco de dados basta ter uma conta Google.

A Figura 12 mostra o banco de dados com informações fictícias de trabalhadores, adicionadas para testes.

Figura 12 - Coleção de Usuário no Firebase



Fonte: Do autor, 2020

A Figura 12 demonstra a organização da coleção dos usuários, dentro da coleção são listados os documentos dos usuários (dados fictícios), gerando uma chave única para cada documento. Dentro do documento, contém as informações do usuário, que neste caso é um usuário fotógrafo. Caso o usuário não for um fotógrafo, dentro do documento "*users*", teria também duas coleções, com os "favoritos", e os "pedidos de contrato" (*favorites* e *orders*).

Como pode ser observado, ao lado esquerdo nas coleções, existem as listas de coleções utilizadas no projeto:

- **Banners:** Banners/Serviços cadastrados pelos fotógrafos para visualização dos usuários;
- **Contracts:** Lista de "Solicitações de Contrato", onde contém os Banners que os usuários clicaram em "Contratar Serviço";
- **Home:** Foi adicionada caso a tela inicial do aplicativo seja apresentada imagens estáticas no mosaico (*Grid View*);
- **Preferences:** Criada para quando for adicionada a sessão de configurações do aplicativo (projeto futuro);
- **Users:** Usuários do aplicativo;
    - **Favorites:** Armazena os Banners em que o usuário clicou para ser adicionado a sua lista de favoritos, por isso foi associado diretamente a coleção dos usuários;

- **Orders:** Armazena apenas o ID da solicitação do serviço, para de forma rápida localizar e separar as ordens de acordo com o usuário logado.

4.2.2. Estrutura de Arquivos - Widgets

Todo *front-end* de uma aplicação que foi desenvolvido com Flutter é criado com base em um conjunto de *Widgets*. *Widget* é um componente que visualmente define a interface de uma parte da tela.

Para visualizar isso de uma forma fácil, podemos pensar em um Quebra-Cabeça conforme a Figura 13, onde cada *widget* é uma peça do mesmo e, ao final, este conjunto de peças representará uma interface completa.

Figura 13 - Widgets em forma de quebra-cabeça

Fonte: MEDIUM, 2020.

Para esse comportamento, o Flutter divide seus widgets em duas partes:

- Layout: São *widgets* responsáveis por determinar a organização e posicionamentos de outros *widgets*. Estes *widgets* servem para delimitar áreas em nossas telas que serão preenchidos por outros *widgets*;

- Interface: *Widgets* responsáveis por criar componentes visuais que serão exibidos para os usuários. Estes *widgets* servem para determinar os componentes que irão compor a interface do aplicativo.

A seguir, irei detalhar como cada um funciona, para melhor entendimento.

4.2.2.1. Widgets de Layout

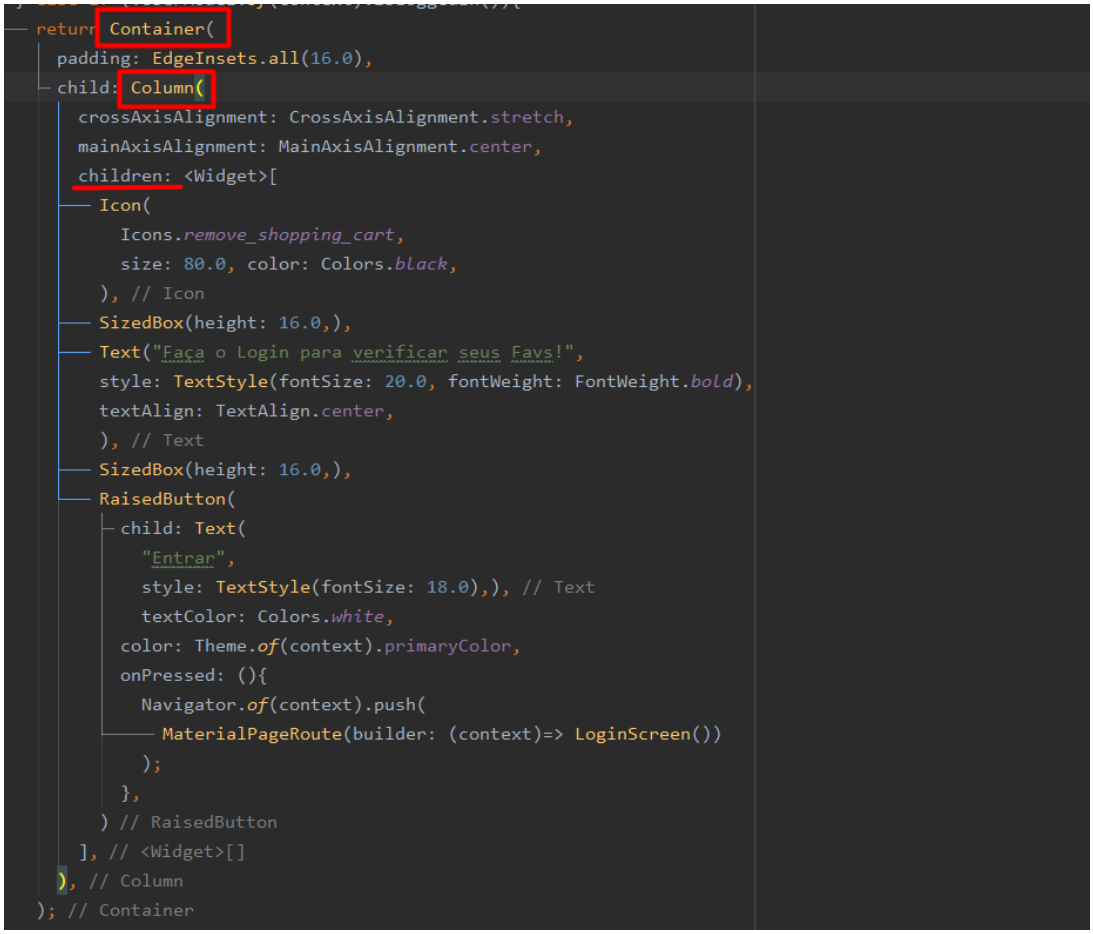
Como dito anteriormente, este é responsável por organizar onde cada parte da tela irá se posicionar. OS *widgets* de *Layout* são considerados os elementos pai de cada tela da aplicação, onde dentro dele que serão utilizados os elementos que farão a composição do *design*.

Alguns exemplos de elementos de *layout* são:

- **Scaffold**: *Widget* responsável por montar um *layout* padrão para o aplicativo, contendo um elemento *appBar*, e o conteúdo da tela.
- **Container**: Um *widget* que combina *widgets* comuns de cor, posicionamento e dimensionamento. Possui uma característica de comportar *widgets* em sua estrutura, separando-os com bordas ou espaços.
- **Column**: Um *widget* que exibe seus filhos em uma matriz vertical.
- **Row**: Um *widget* que exibe seus filhos em uma matriz horizontal.

A Seguir, na Figura 13 é apresentado um exemplo de um *Widget* de *Layout*, que foi implementado na tela listagem dos favoritos:

Figura 14 - Widget de Layout no Flutter



```
return Container(
  padding: EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Icon(
        Icons.remove_shopping_cart,
        size: 80.0, color: Colors.black,
      ), // Icon
      SizedBox(height: 16.0,),
      Text("Faça o Login para verificar seus Favs!",
      style: TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold),
      textAlign: TextAlign.center,
      ), // Text
      SizedBox(height: 16.0,),
      RaisedButton(
        child: Text(
          "Entrar",
          style: TextStyle(fontSize: 18.0),), // Text
          textColor: Colors.white,
        color: Theme.of(context).primaryColor,
        onPressed: (){
          Navigator.of(context).push(
            MaterialPageRoute(builder: (context)=> LoginScreen())
          );
        },
      ) // RaisedButton
    ], // <Widget>[]
  ), // Column
); // Container
```

Fonte: Do autor, 2020

Conforme destacado na Figura 14, essa parte da listagem dos favoritos implementa o retorno de uma função, onde retorna um *Widget Layout* do tipo Container, que por sua vez em um de seus elementos, está dando espaçamento em todos os ângulos da tela (*padding*), ou seja, na parte superior, inferior, esquerda e direita está sendo aplicado uma margem de 16 pixels.

Em seguida, está sendo atribuído um elemento *child* (filho do Container), que por sua vez utiliza outro *widget Layout*, o *Column*, e neste são atribuídos os *widegts* de Interface na propriedade *children*, podendo adicionar mais do que um.

4.2.2.2. Widgets de Interface

Os *Widgets* de *Interface* servem para criar componentes visuais que serão incluídos na estrutura de um *Widget* de *Layout*, fazendo assim uma organização da tela de acordo com seu elemento principal.

Não é permitido incluir um *Widget* de Interface sem um *Widget* de *Layout*, caso contrário irá resultar em uma inconsistência na aplicação, mas como foi utilizado a IDE para auxílio no desenvolvimento, este tipo de situação foi previsto, apresentando dicas no desenvolvimento do que é necessário para usar cada *Widget*, mesmo em alguns casos não sendo muito claro.

Alguns exemplos deste tipo de *Widget* são botões, *labels*, ícones, textos ou qualquer outro componente que compõe a interface de uma aplicação móvel, conforme destacado na Figura 15.

Figura 15 - Widget de Interface no Flutter

```
child: Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Icon(
      Icons.remove_shopping_cart,
      size: 80.0, color: Colors.black,
    ), // Icon
    SizedBox(height: 16.0,),
    Text("Faça o Login para verificar seus Favs!",
    style: TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold),
    textAlign: TextAlign.center,
    ), // Text
    SizedBox(height: 16.0,),
    RaisedButton(
      child: Text(
        "Entrar",
        style: TextStyle(fontSize: 18.0),), // Text
        textColor: Colors.white,
      color: Theme.of(context).primaryColor,
      onPressed: (){
        Navigator.of(context).push(
          MaterialPageRoute(builder: (context)=> LoginScreen())
        );
      },
    ) // RaisedButton
```

Fonte: Do autor, 2020

4.2.3. Estrutura de Arquivos – Estados

Para que o aplicativo consiga ter uma comunicação direta, em tempo de execução com o usuário, o Flutter trabalha com dois tipos de estados, onde quando um usuário precisa de uma resposta ou aguarda uma alteração na tela, isso é denominado como Estado. Imagine que você está em um aplicativo de compras, achou um produto que lhe agrade e, em seguida, clica na opção de “adicionar ao carrinho”, neste momento o carrinho (na maioria das vezes) adiciona o número um sobre ele, informando que contém um item no carrinho, isso é a mudança de estado na tela do aplicativo, sem recarregar ou sem perder nenhuma informação que contém na mesma.

No Flutter existe um *Widget* chamado ***Stateful*****,** onde permite a alteração e o *Widget* chamado ***Stateless***, que é utilizado quando não tem a necessidade de alterar alguma informação em tempo real. É possível imaginar que poderia ser utilizado sempre o *Stateful*, mesmo não tendo alteração de tela, porém isso causa um problema de desempenho, já que o aplicativo precisará sempre “assistir” as ações da tela para realizar uma alteração de estado.

4.2.3.1. Estado Stateful

Como dito anteriormente, o *Widget* de estado *Stateful* é responsável por alterar o estado de um componente sem a necessidade de atualizar a tela ou redirecionar para outra. Para isso funcionar, o *Stateful* utiliza a estrutura da imagem a seguir (Figura 16).

Figura 16 - Widget estado Stateful (BannerScreen)

```
class BannerScreen extends StatefulWidget {
  //Recebe banner
  final BannerData banner;
  BannerScreen(this.banner);
  @override
  _BannerScreenState createState() => _BannerScreenState(banner);
}

class _BannerScreenState extends State<BannerScreen> {
  bool isLoading = false;
  //Recebe o Banner e salva aqui
  final BannerData banner;
  final GlobalKey<ScaffoldState> scaffoldKey = GlobalKey<ScaffoldState>();

  _BannerScreenState(this.banner);
  @override
  Widget build(BuildContext context) {
    final Color primaryColor = Theme.of(context).primaryColor;
```

Fonte: Do autor, 2020

No exemplo da Figura 16, A classe *BannerScreen* irá retornar uma nova instância da classe *_BannerScreenState* sempre que um estado for alterado na aplicação. Este evento permite que um aplicativo em Flutter possa atualizar informações em tempo real, sem necessidade de uma atualização na tela.

Para essa classe, a atualização ocorre na variável booleana *isLoading*, que é definida como false, porém quando é feita alguma alteração na tela, é alterada para *true* e então é criado um *preloader* na tela, impedindo ação de click na mesma.

Um exemplo muito claro é quando utiliza um Widget do tipo *CheckBox*, que ao clicar sobre, precisa ocorrer uma alteração de estado na tela para mostrar ao usuário como marcado, então basta adicionar na ação do botão o método *setState*(), que irá notificar a tela que houve uma alteração.

Também, pode ser utilizado o *notifyListeners*(), mas este é mais utilizado nos *Models*, por exemplo ao clicar para adicionar um Banner aos favoritos, existe uma função no modelo *FavoriteModel* que se chama addFavoriteItem, onde ao final da execução da função é utilizada o *NotiyListeners*, fazendo com que o *Widget* receba

uma atualização no *ScopedModel* avisando que ocorreu uma mudança nos valores e de estado, conforme demonstração na Figura 17.

Figura 17 - notifyListeners na função addFavoriteItem

```
void addFavoriteItem(FavoriteBanner favoriteBanner){
  banners.add(favoriteBanner);

  Firestore.instance.collection("users").document(user.firebaseUser.uid)
  .collection("favorites").add(favoriteBanner.toMap()).then((doc){
    favoriteBanner.fid = doc.documentID;
  });

  notifyListeners();
}
```

Fonte: Do autor, 2020

4.2.3.2. Estado Stateless

O *Widget* de estado *Stateless* é utilizado quando necessário utilizar uma tela em que não ocorrerá alteração de estado, ou seja, uma mensagem estática, um formulário de cadastro simples, uma imagem fixa, etc. Na Figura 18 é apresentado na classe *FavButton*, que foi implementado este tipo de estado.

Figura 18 - Widget de Estado Stateless (FavButton)

```
class FavButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return FloatingActionButton(
      child: Icon(Icons.favorite, color: Colors.white),
      onPressed: (){
        Navigator.of(context).push(
          MaterialPageRoute(builder: (context) => FavScreen())
        );
      },
      backgroundColor: Theme.of(context).primaryColor,
    ); // FloatingActionButton
  }
}
```

Fonte: Do autor, 2020

Na Figura 18, foi criado um *Widget* específico para o *FavButton*, o botão flutuante onde a função é apenas conceder acesso rápido a lista dos favoritos, onde foi utilizado na maioria das telas.

A ação é retornar um *FloatingActionButton* (botão flutuante), o *child* (elemento filho) é um ícone no formato de coração/favorito, com a cor branca. O *onPressed* é a função que o botão irá realizar ao ser clicado, neste caso é utilizado o *Navigator.push* (Adicionando a rota para ser navegada, que irá criar apenas uma tela por cima da que está atualmente, possibilitando clicar na seta para voltar a tela anterior) utilizando o método *MaterialPageRoute* (responsável por criar a rota para a nova tela), onde neste é chamado o novo *Widget*, com a listagem dos favoritos.

4.2.4. Estrutura de Arquivos - Models

O Flutter utiliza uma estrutura um pouco diferente do padrão de outros *frameworks*, em questão de estrutura de aplicação. O mesmo não segue um padrão, como por exemplo o *MVC*. A estrutura é o próprio desenvolvedor que define, conforme achar melhor, porém se preferir, existem plugins para instruir nesta parte, um específico inclusive para *MVC*, porém não foi utilizado neste projeto.

Para este projeto, foi criado uma camada Modelo para cada CRUD, onde sua função é realizar a comunicação com os dados da aplicação, chamando por funções no Widget para realizar cada ação.

A camada *Model* tem um papel muito importante, onde por exemplo, o *Model* do usuário é definido no arquivo principal do projeto, onde é possível chamar qualquer atributo do *UserModel* em qualquer tela, desta forma se torna prático verificar se o usuário é um fotógrafo em qualquer local do projeto, ou quaisquer outro parâmetro que foi definido na coleção.

Porém para realizar este processo, não basta apenas importar o *Model* e utilizar, é necessário aplicar o mesmo em um *Scoped Model*, conforme aplicado no arquivo *UserModel* na Figura 19.

Figura 19 - Arquivo Model (UserModel)

```
void signIn({@required String email, @required String senha,
  @required Function onSuccess, @required Function onFail}) async{
  isLoading = true;
  notifyListeners();

  try {
    final AuthResult result = await _auth.signInWithEmailAndPassword(
        email: email, password: senha);

    userId = result.user.uid;
    firebaseUser = result.user;

    print("Usuario" + userId);
    await _loadCurrentuser();

    onSuccess();
    isLoading = false;
    notifyListeners();
  } on PlatformException catch (e){

    onFail(getErrorString(e.code));
  }
  isLoading = false;
  notifyListeners();
}
```

Fonte: Do autor, 2020

4.2.5. Estrutura de Arquivos – Scoped Model

Todo componente do Flutter é definido por um *Widget*, que nada mais é que uma parte da tela, ou comparando com elementos HTML, uma div com propriedades específicas para cada tipo de *Widget*. Porém, para se comunicar um *Widget* com outro ou usar um *Model* para trazer dados a uma tela (por exemplo), não é possível simplesmente tentar importar o arquivo, isso não irá resultar em sucesso.

Para este tipo de situação, é utilizado o *Scoped Model*, que é responsável pela comunicação entre *widgets* que precisam receber e compartilhar dados de uma Classe, ou atualizar dados da tela de acordo com determinada ação. Devido ao fato de podermos vincular apenas uma classe ao widget *ScopedModel*, em aplicativos com mais classes que compartilham dados, não é recomendado o uso do scoped_model pois o estado pode ser alterado em mais de uma classe, mas esse não é o caso do aplicativo em questão. Na Figura 20 é mostrado como foi utilizado o *ScopedModel* para que toda a aplicação tenha acesso.

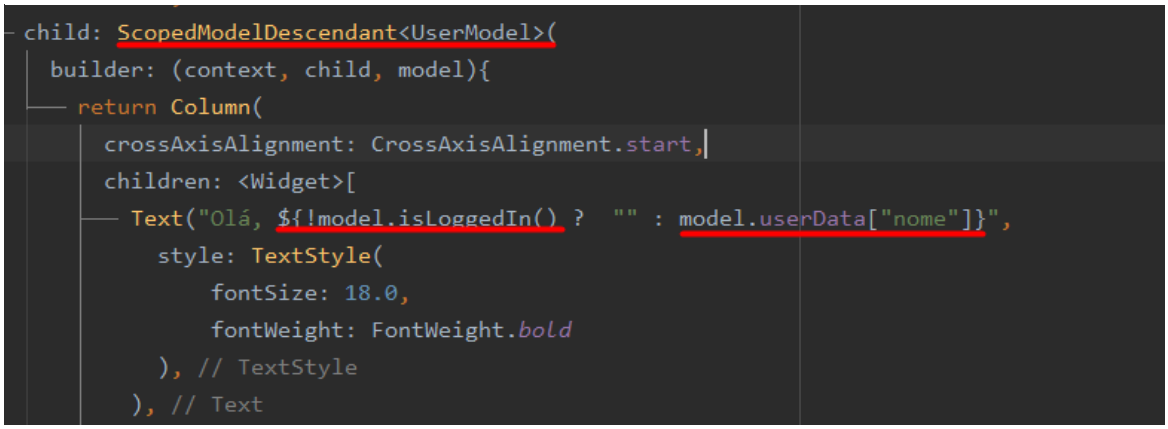
Figura 20 - ScopedModel no cabeçalho da aplicação

```
return ScopedModel<UserModel>(
  model: UserModel(),
  child: ScopedModelDescendant<UserModel>(
    builder: (context, child, model){
      return ScopedModel<FavoriteModel>(
        model: FavoriteModel(model),
        child: MaterialApp(
          title: 'PhotoClick',
          theme: ThemeData(
            primarySwatch: Colors.blue,
            primaryColor: const Color.fromARGB(255, 4, 125, 141),
            scaffoldBackgroundColor: const Color.fromARGB(255, 0, 204, 204),
            //visualDensity: VisualDensity.comfortable,
            appBarTheme: const AppBarTheme(
                elevation: 0
            ), // AppBarTheme
          ), // ThemeData
          debugShowCheckedModeBanner: false,
          home: HomeScreen(),
        ), // MaterialApp
      ); // ScopedModel
    },
  ) // ScopedModelDescendant
); // ScopedModel
```

Fonte: Do autor, 2020

Como pode ser observado no arquivo main.dart (Figura 20), para iniciar o projeto é retornado um *Widget ScopedModel* utilizando o *UserModel*, e quando for necessário acionar alguma ação do *model* Usuário, basta utilizar o ScopedModelDescendent, conforme na Figura 21.

Figura 21 - Exemplo do uso do ScopedModelDescendent



Fonte: Do autor, 2020

Neste trecho do código da Figura 21, é validado se o usuário está logado, através da função *isLoggedIn*, que foi cadastrada no *model*. Caso a resposta da função retorne *True*, é utilizado o ternário que também busca o nome do usuário, através do *scopedModel*, onde é acessado o objeto *UserData*, e obtido o nome do mesmo.

4.2.6. Estrutura de Arquivos – Arquivo Inicial (Main)

Como Flutter tem como linguagem o Dart, desta forma todos os arquivos são de extensão *dart*. Ao iniciar um novo projeto, automaticamente é criado um arquivo chamado “main.dart”, onde ao rodar o aplicativo, sempre irá observar neste arquivo para dar início ao simulador.

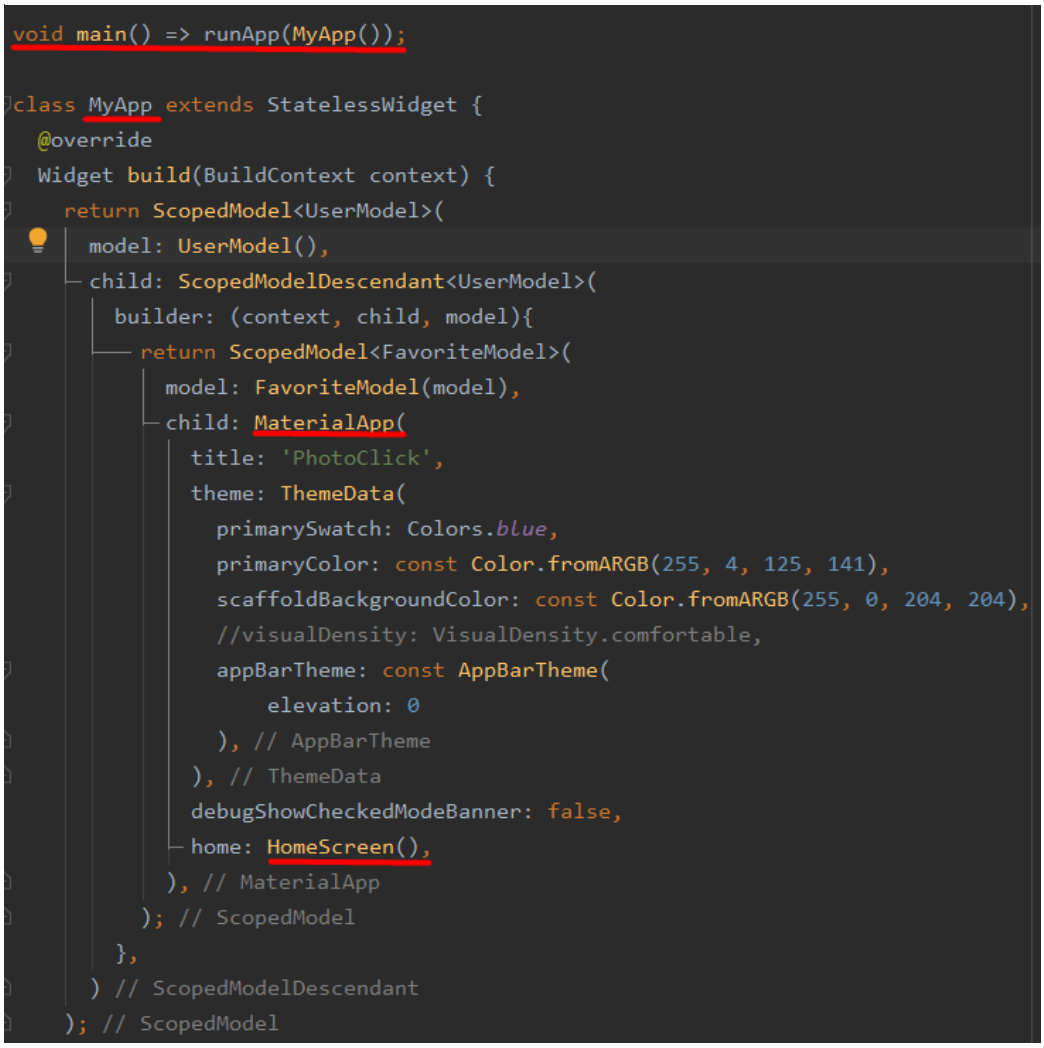
Antes de entrar neste assunto, é necessário entender um pouco mais sobre *Material Design*. "*Material Design*" é uma "linguagem de *design*" desenvolvida pelo Google. Essa nova metodologia de *design* é uma tendência para os *Design*.

Projetado para ser intuitivo e de simples compreensão, o Material Design possui diversas particularidades, tendo como objetivo agrupar os conceitos de um bom design com a inovação, proporcionando uma experiência uniforme através de diversas plataformas diferentes, sejam smartphones, computadores ou relógios inteligentes.

Entendendo isso, todo arquivo *main* começa com um método denominado ***main*(),** o mesmo é o ponto de partida de qualquer aplicativo. Executando o método

***runApp*()** dentro do ***main*()**, o Flutter constrói o *widget* informado como parâmetro. Neste caso, um widget do tipo *MaterialApp* (fornece um *layout* orientado ao *material design*). Na Figura 22 é apresentado um trecho do que foi implementado no arquivo **main.dart**.

Figura 22 - Arquivo Inicial da aplicação (main.dart)



```
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ScopedModel<UserModel>(
      model: UserModel(),
      child: ScopedModelDescendant<UserModel>(
        builder: (context, child, model){
          return ScopedModel<FavoriteModel>(
            model: FavoriteModel(model),
            child: MaterialApp(
              title: 'PhotoClick',
              theme: ThemeData(
                primarySwatch: Colors.blue,
                primaryColor: const Color.fromARGB(255, 4, 125, 141),
                scaffoldBackgroundColor: const Color.fromARGB(255, 0, 204, 204),
                //visualDensity: VisualDensity.comfortable,
                appBarTheme: const AppBarTheme(
                    elevation: 0
                ), // AppBarTheme
              ), // ThemeData
              debugShowCheckedModeBanner: false,
              home: HomeScreen(),
            ), // MaterialApp
          ); // ScopedModel
        },
      ) // ScopedModelDescendant
    ); // ScopedModel
```

Fonte: Do autor, 2020

Na Figura 22, pode ser observado que o **método *main*** roda a **classe MyApp**, que contém todo o projeto encapsulado nele, então após compilar, tudo se inicia aqui.

Um grande detalhe é que foi utilizado o *model* do *UserModel* e do *FavoriteModel* antes de iniciar de fato o *MaterialApp*, já que preciso monitorar em alguns pontos do aplicativo dados destas duas coleções, então de qualquer ponto do aplicativo é possível chamar o *UserModel* e o *FavoriteModel*. O *title* seria o título do

aplicativo, o atributo *theme* foi utilizado o Widget *ThemeData* para definir escala de cores no aplicativo. O *appBar* seria a barra superior do aplicativo, e por fim o *home* seria a página inicial do aplicativo, que neste caso está importando o widget *HomeScreen*.

4.2.7. Navegação entre Telas

Para realizar a navegação entre telas, foi necessário também utilizar um Widget, já que Flutter é praticamente orientado a Widgets. O Widget utilizado é chamado de *PageView*, e o que controla as ações é chamado de *PageController*.

O Widget *PageView* gera páginas na tela onde podemos realizar ações, ter uma lista de páginas (rolagem) ou uma função que constrói páginas. Já o *PageController* é utilizado para controlar uma *PageView* via código, onde cada Widget dentro dos filhos terá um código de página específico. Após obter a ação de *click* do usuário, basta utilizar a função **controller.jumpToPage(_pageController).** Na Figura 23 é apresentado como foi estruturado o Widget *PageView* e seus componentes.

Figura 23 - PageView (navegação entre Telas)

```
final _pageController = PageController();

@override
Widget build(BuildContext context) {
  return PageView(
    controller: _pageController,
    physics: NeverScrollableScrollPhysics(), //Não permite arrastar as telas
    children: <Widget>[
    Scaffold(
      body: HomeTab(),
      drawer: CustomMenu(_pageController),
      floatingActionButton: FavButton(),
    ), // Scaffold
    Scaffold(
      appBar: AppBar(
        title: Text("Banners"),
        centerTitle: true,
      ), // AppBar
      drawer: CustomMenu(_pageController),
      body: BannersTab(),
      floatingActionButton: FavButton(),
    ), // Scaffold
```

Fonte: Do autor, 2020

Conforme apresentado na Figura 23, a Classe *HomeScreen* utiliza um controlador dentro da *Widget pageView*, onde define qual vai ser o controlador atual de cada sessão da tela. Este *controller* também está disponível em outras situações, como por exemplo em uma tela que contenha um formulário, onde cada caixa de texto por exemplo, terá seu *controller* que será seu valor, neste caso o *controller* é um objeto que pode ser usado para controlar a posição para a qual a visualização da página é direcionada ou rolada.

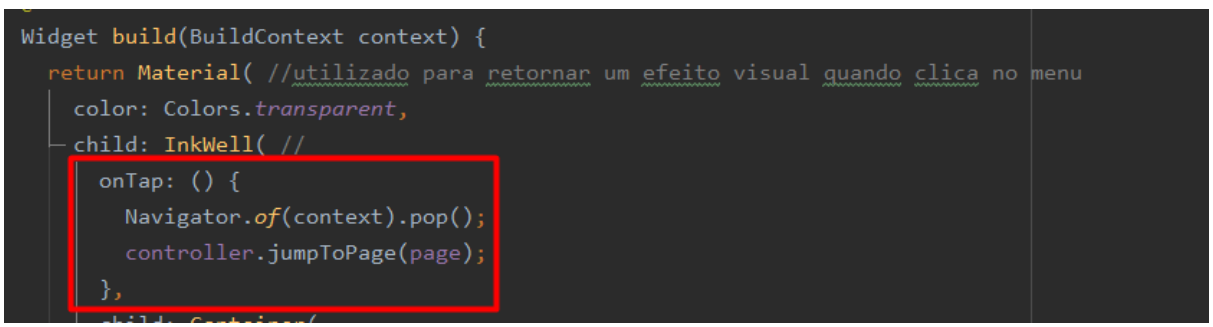
O *widget Scaffold* implementa a estrutura do *layout* visual do *Material Design* básico e permite definir outros *widgets* do *Material Design* no seu interior.

O *widget Scaffold* é utilizado basicamente para criar um aplicativo móvel de uso geral, e, contém quase tudo que você precisa para criar um aplicativo funcional e responsivo.

Em geral é utilizado o *MaterialApp* uma vez na aplicação, e, o widget *Scaffold* para cada tela que desejamos desenhar. Foi desenvolvido então para cada tela do aplicativo, como estrutura pai um *Scaffold*, pois ele possui muitas opções que para este caso da construção do menu lateral e navegação de telas. A opção *body* do *Scaffold*, é o conteúdo que será apresentado ao ser clicado, podendo ser qualquer tipo de Widget, então o *body* nada mais é do que o conteúdo de cada tela.

A opção *drawer*, é o que permite criar uma navegação entre telas, onde neste caso é chamado um novo arquivo com o *Widget* da construção do menu, e é passado por parâmetro seu *pageController*, para referenciar cada item construído, conforme apresentado na Figura 24.

Figura 24 - Controlador de Páginas



Fonte: Do autor, 2020

Analisando a Figura 24, a função *onTap* refere-se à quando o usuário clicar sobre um item no menu, onde ao clicar irá disparar o que foi descrito dentro da função. Neste caso, é realizado a migração para outra página, de acordo com a página que foi passada por parâmetro.

4.2.8. Arquivos de Validação

Para validar os campos preenchidos pelo usuário, foi criado uma classe de validação, para utilizar foi importado o arquivo, e chamada a função de cada campo e assim retornará a inconsistência destacando o campo do formulário em vermelho.

A Figura 25 é apresentado como foi estruturado o arquivo *banner_validator*, onde valida os campos do Banner ao cadastrar ou editar.

Figura 25 - Validador de campos do Banner

```
class BannerValidator{

  String validateImagem(List imagens){
    if(imagens.isEmpty) return "Adicione imagens do Banner";
    return null;
  }

  String validateTitulo(String text){
    if (text.isEmpty) return "Preencha o título do Banner";
    return null;
  }

  String validateDescricao (String text){
    if(text.isEmpty) return "Preencha a Descrição do Banner";
    return null;
  }

  String validatePreco(String text){
    double preco = double.tryParse(text);
    if(preco != null){
      if(!text.contains(".")) || text.split(".")[1].length != 2)
        return "Utilize duas casas decimais";
    }else {
      return "Preço Inválido";
    }
    return null;
  }
}
```

Fonte: Do autor, 2020

Para cada campo do Banner, foi criado um validador. Cada função de validação recebe por parâmetro o campo em questão, onde apenas o *validateImagem* é em outro formato, recebe um *List*, que no flutter seria como um *Array* de posições, os demais recebem um valor do tipo *String*.

Na Figura 26, é exibido como foi implementado em cada campo do formulário do Banner a validação.

Figura 26 - Formulário Banner

```
Form(
  key: _formKey,
  child: StreamBuilder<Map>(
    stream: _bannerBloc.outData,
    builder: (context, snapshot) {
      if(!snapshot.hasData) return Container();
      return ListView(
        padding: EdgeInsets.all(16),
        children: <Widget>[ //Filho para os campos das tela
          Text(
            "Imagens",
            style: TextStyle(
              color: Colors.black,
              fontSize: 12
            ), // TextStyle
          ), // Text
          ImagesWidget(
            context: context,
            initialValue: snapshot.data["fotos"],
            onSaved: _bannerBloc.saveImagens,
            validator: validateImagem,
          ), // ImagesWidget
          TextFormField(
            initialValue: snapshot.data["titulo"],
            style: _fieldStyle,
            decoration: _buildDecoration("Título"),
            onSaved: _bannerBloc.saveTitulo,
            validator: validateTitulo,
          ), // TextFormField
```

Fonte: Do autor, 2020

Para utilizar a validação nos campos/*Widgets*, foi necessário utilizar um *Widget Form* (Figura 26), que atua como contêiner para agrupar e validar os *Widgets*. Cada formulário criado deve conter uma *key,* que vai identificar de forma única o formulário e permitirá assim validar posteriormente.

Agora, ao observar os *Widgets* que contém as ações do usuário para preenchimento (*ImagesWidget* e *TextFormField*), eles possuem uma opção chamada *validator*, que então utiliza a função definida no arquivo de validação do Banner.

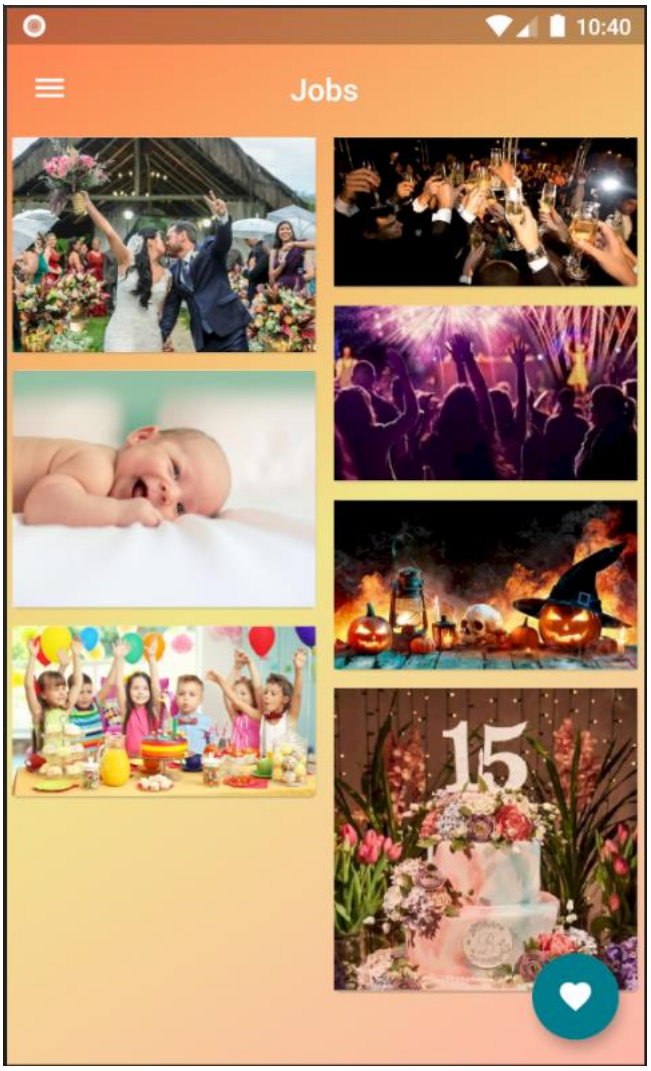
O *onSaved* é utilizado para fazer uma chamada e concluir o processamento do formulário, ou seja, vai salvar em um *Array* os dados do formulário, para realizar a inserção ao clicar no *submit*.

5. RESULTADOS

Após a conclusão do desenvolvimento do aplicativo, tornou-se possível realizar login, criar um novo usuário, recuperar senha via e-mail, registrar Banners/Publicações do fotógrafo, adicionar um banner aos favoritos do usuário, adicionar um banner a tela de contratação e listar fotógrafos. Logo, neste tópico será demonstrado os processos que o aplicativo está realizando.

Ao acessar o Aplicativo, o usuário não é obrigatório estar logado, porém para fazer determinadas ações dentro do aplicativo, o sistema irá permitir apenas se verificar que o mesmo está autenticado. A Figura 27 apresenta a tela inicial do aplicativo.

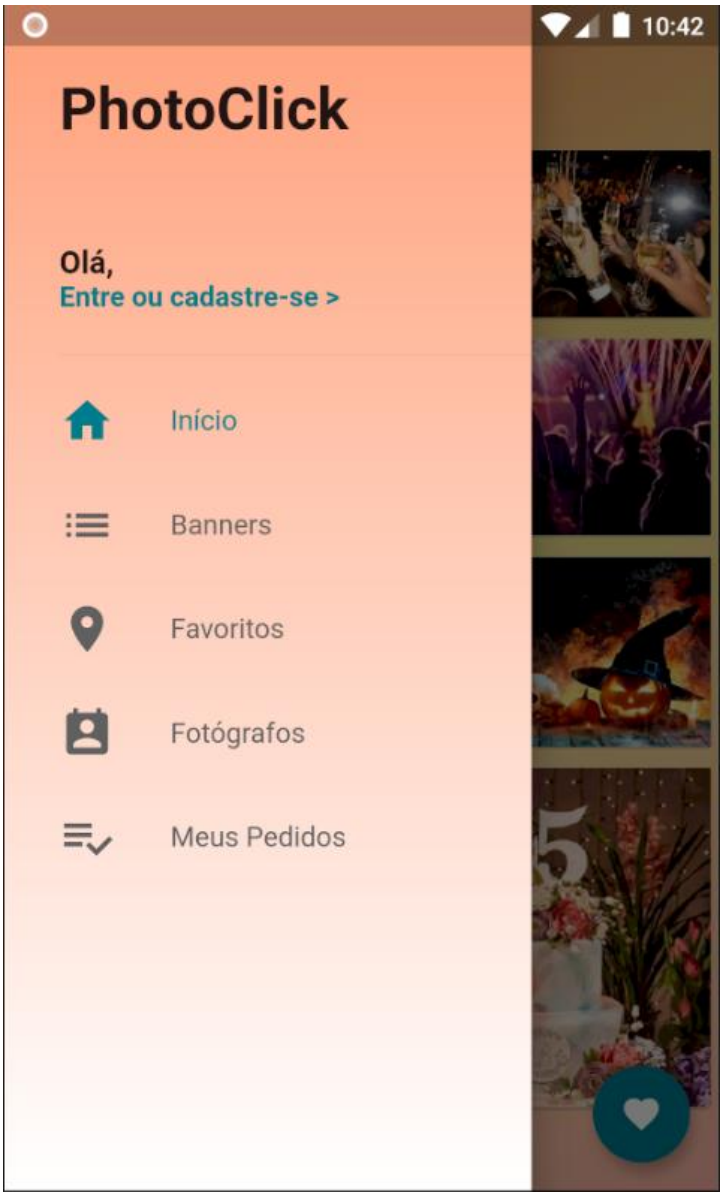
Figura 27 - Tela Home do Aplicativo



Fonte: Do autor, 2020

O aplicativo conta com um Menu de fácil acesso, onde possui a opção de fazer o *Login*, sair, acessar a tela *Home*, tela das categorias, tela dos favoritos, tela dos itens marcados para contratação e a tela da listagem dos fotógrafos cadastrados no aplicativo. A Figura 28 apresenta o menu do aplicativo.

Figura 28 - Menu do Aplicativo

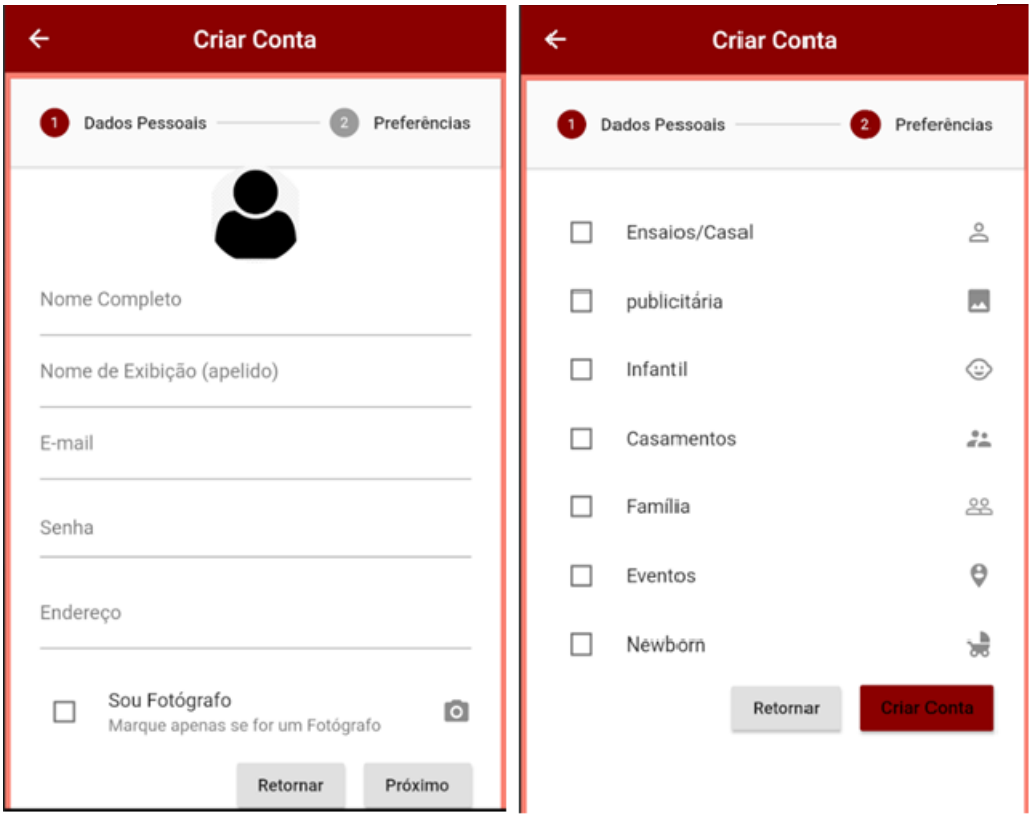


Fonte: Do autor, 2020

Para realizar o *Login*, pode ser realizado através do Menu – Entrar ou Cadastrar, ou então ao tentar fazer uma ação, como por exemplo adicionar um Banner aos favoritos, desta forma irá apresentar um modal solicitando o *login* do usuário.

A tela de *Sign Up* (cadastro), é dividia em etapas, onde a etapa um entrega a possibilidade do usuário carregar uma foto de perfil, informar dados pessoais, solicita se o mesmo é um fotógrafo (para fazer validações em outras telas onde apenas o usuário fotógrafo poderá ter acesso), e a etapa dois serve para o usuário marcar suas preferências de busca ou suas qualidades (se for fotógrafo), como ensaios de casal, casamentos, festas, entre outras. A Figura 29 apresenta o a tela de *Sigin Up* do aplicativo.

Figura 29 - Tela de Sigin Up



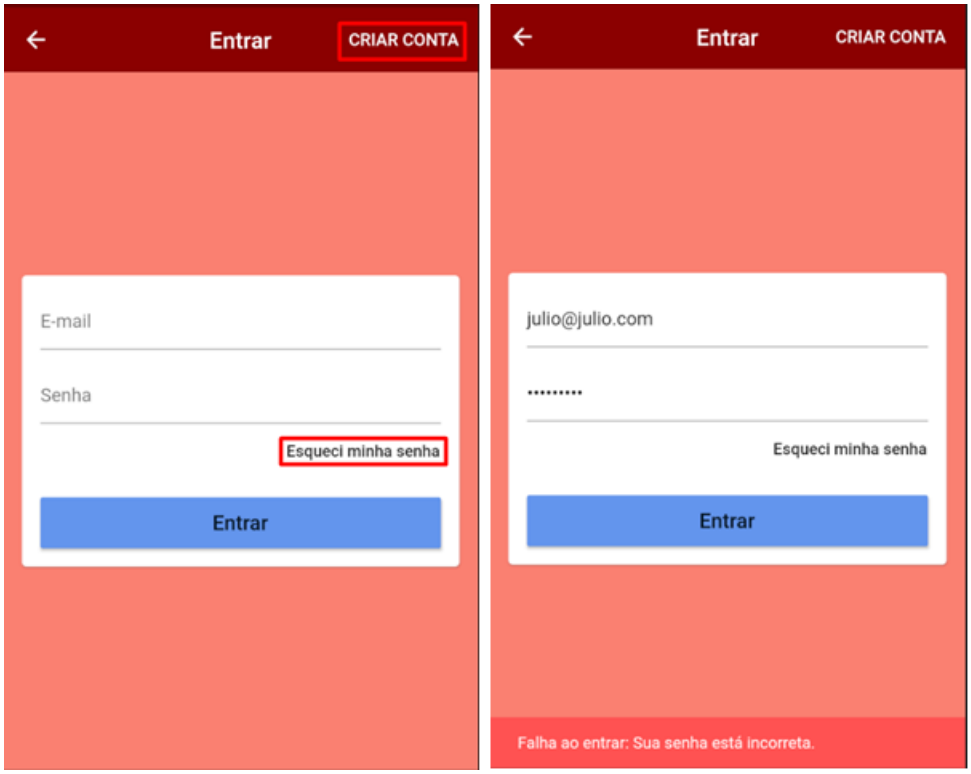
Fonte: Do autor, 2020

Após realizar o cadastro, automaticamente o aplicativo faz o *Login* de acordo com os dados preenchidos no cadastro. A outra opção de se autenticar com o sistema, é fazendo o *Login*, caso o usuário já possua uma conta. Neste caso apresenta uma tela onde solicita o e-mail e senha, um botão para criar conta, e uma opção “esqueci minha senha”.

Se clicar em “esqueci minha senha”, o aplicativo valida se o e-mail foi preenchido, para então validar se o e-mail existe na base de dados, se existir envia

um e-mail com a nova senha. A Figura 30 apresenta a tela de login que foi desenvolvida.

Figura 30 - Tela de Login/Validações

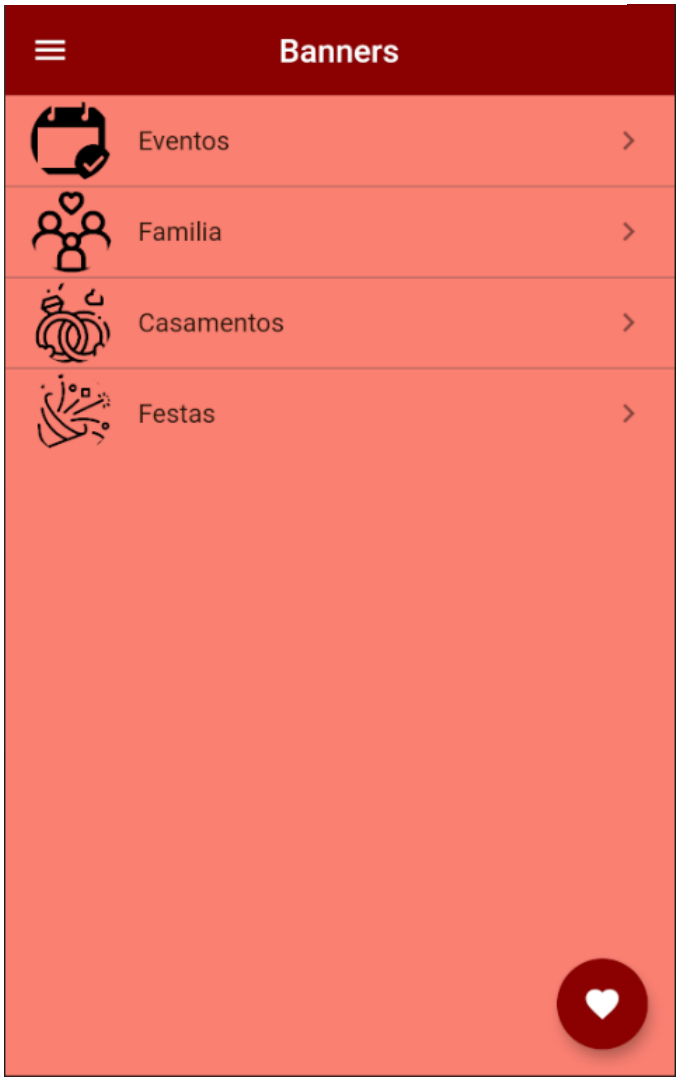


Fonte: Do autor, 2020

Caso ocorra alguma inconsistência no login, o Firebase *Autentication* retorna um código de erro, de acordo com o erro. Para traduzir e apresentar ao usuário este erro, foi criado uma classe de validação com o erro em português de acordo com o código que é recebido, e então é apresentado por quatro segundos na tela.

Após a autenticação, o usuário terá acesso ao sistema de acordo com sua categoria (Fotógrafo ou Não Fotógrafo). Um usuário Fotógrafo poderá registrar um novo Banner escolhendo em qual categoria deseja cadastrar, editar o Banner abrindo o mesmo, onde o aplicativo verifica se o usuário logado que está abrindo o Banner é o mesmo usuário que criou, se for o mesmo irá apresentar um botão flutuante para editar, também poderá acessar todos os banners cadastrados e adicionar aos favoritos. Já o usuário normal, poderá acessar a listagem dos banners por categoria, adicionar à lista de favoritos, solicitar uma contratação para o fotógrafo e acessar a listagem dos fotógrafos. A Figura 31 exibe a tela de Listagem das Categorias.

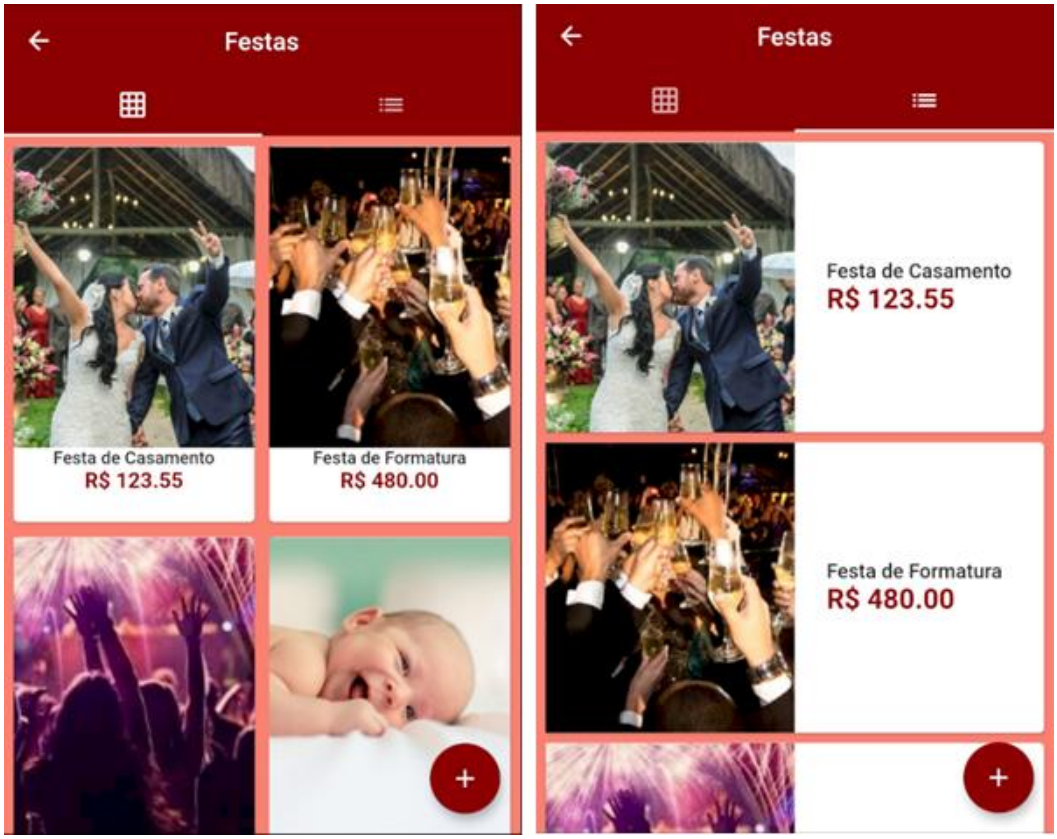
Figura 31 - Listagem de Categorias



Fonte: Do autor, 2020

Para selecionar em qual categoria cadastrar a publicação, o usuário deve acessar a lista de categorias e clicar no botão flutuante com o símbolo “+”, conforme a Figura 32.

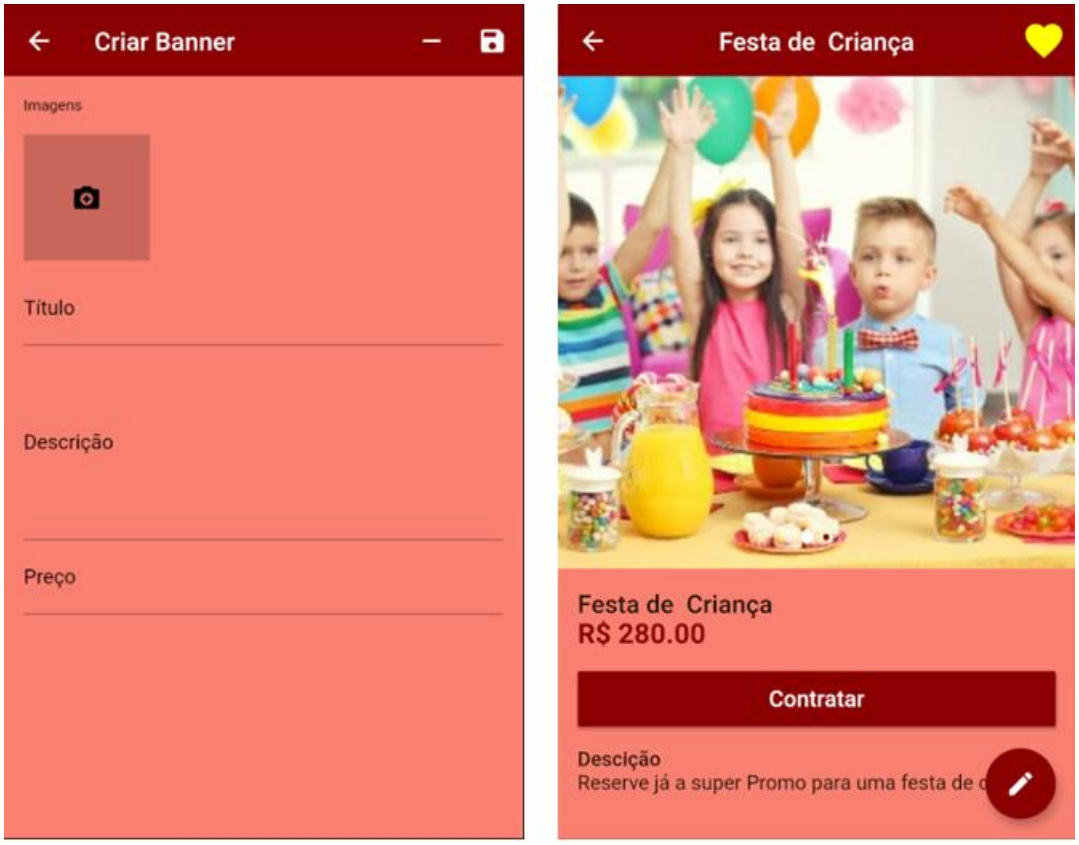
Figura 32 - Publicações/Banners Cadastrados



Fonte: Do autor, 2020

Para realizar o cadastro de um novo Banner/Publicação, basta clicar no botão flutuante “+”, onde irá direcionar para a tela de criação. Para editar é exibida a mesma tela, porém, já trazendo os dados da publicação selecionada. Basta abrir a publicação e clicar no botão flutuante com o símbolo de um lápis, e irá direcionar para a tela de manutenção, conforme as duas ilustrações da Figura 33.

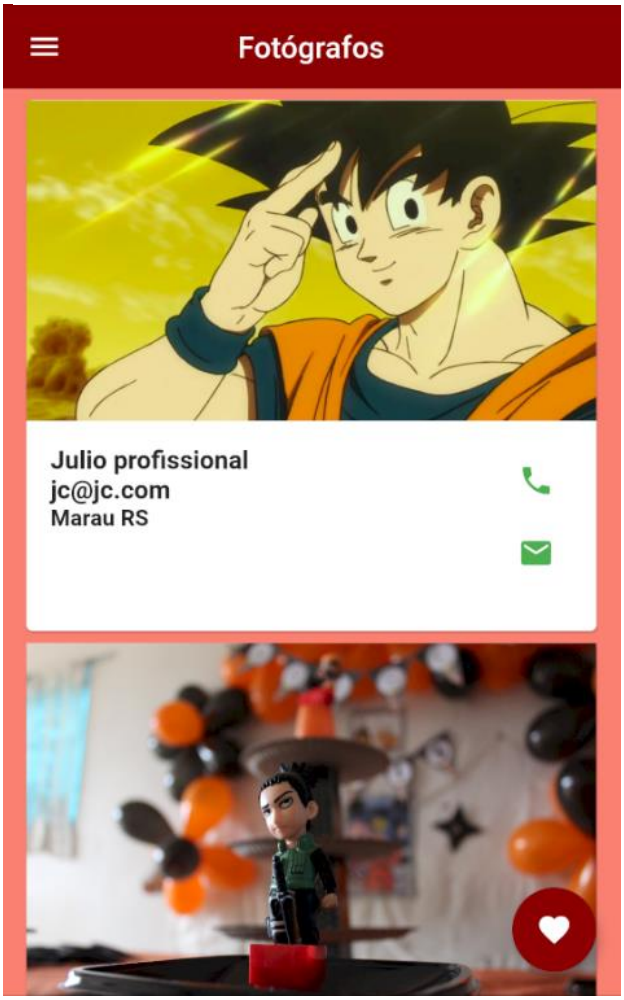
Figura 33 - Tela de Manutenção



Fonte: Do autor, 2020

O aplicativo conta também com o usuário que não é fotógrafo, onde o mesmo é restrito as ações perante a manutenção e criação de Banner/Publicação, porém tem acessos exclusivos como acesso a solicitação de contrato e a listagem dos fotógrafos cadastrados no aplicativo. A Figura 34 mostra a listagem dos fotógrafos, com a foto que foi seleciona ao criar o perfil, endereço e contato.

Figura 34 - Listagem de Fotógrafos



Fonte: Do autor, 2020

Para adicionar um item aos favoritos, basta o usuário acessar um Banner/Publicação que tenha gostado, e então clicar no ícone do coração no canto superior direito conforme na Figura 33, desta forma irá adicionar aos favoritos, já direcionando o usuário para a listagem dos favoritos. A Figura 35 apresenta a listagem dos itens adicionados aos Favoritos.

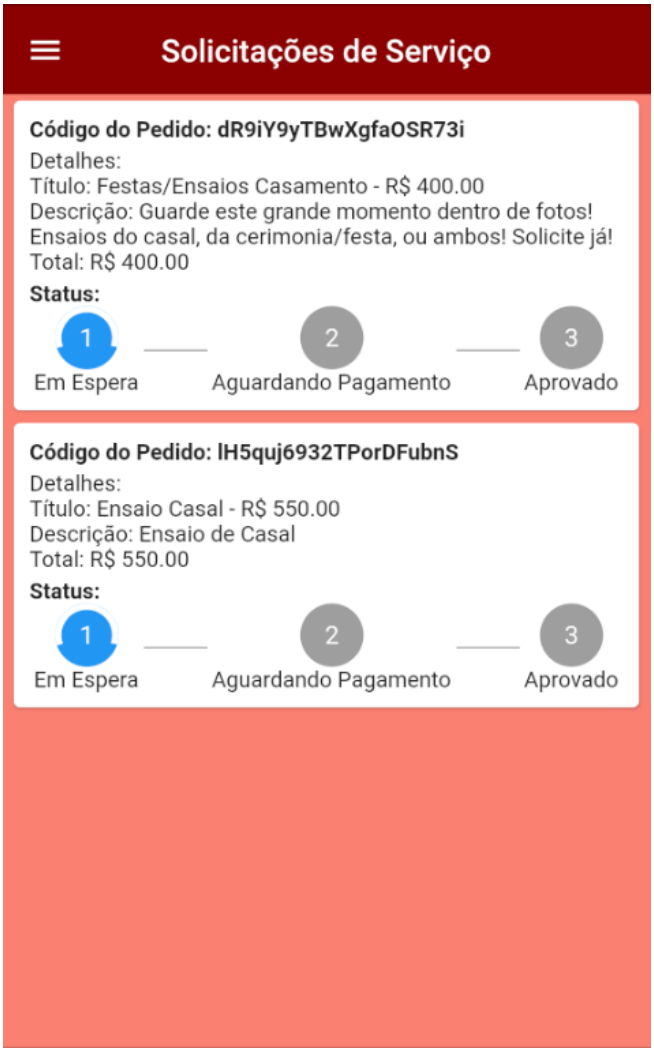
Figura 35 - Listagem dos Favoritos

Fonte: Do autor, 2020

Para realizar um contato de contratação de um pacote disponibilizado pelo fotógrafo, basta o usuário acessar a publicação, e clicar no botão “Contratar”. Ao fazer isso, o aplicativo irá gerar uma chave única para identificar o seu “pedido”, após poderá acompanhar através da listagem do menu “minhas contratações”. Neste, poderá observar os itens adicionados e o Status, onde até então não é possível concretizar de fato uma contratação, mas as informações estão sendo armazenadas em uma coligação para uma futura implementação.

A Figura 36 apresenta a opção de adicionar para as solicitações de contrato, assim como a listagem dos já adicionados.

Figura 36 - tela Solicitações de Serviço



Fonte: Do autor, 2020

Essas foram as funções desenvolvidas para o aplicativo, onde buscou-se desenvolver algo de fácil entendimento para o usuário, que ao visualizar possivelmente já entende o que pode ser feito em cada botão ou ícone da tela, onde foi obtido o resultado esperado. O aplicativo passou por um breve teste através do próprio Emulador do Android Studio por um usuário, mas apenas para averiguar se o usuário conseguiria se localizar nas ações e telas do aplicativo.

6. CONSIDERAÇÕES FINAIS

O presente estudo teve por objetivo desenvolver um aplicativo para facilitar o contato entre um fotógrafo e um cliente, no qual sua finalidade é de realizar a divulgação e gerenciamento de promoções tudo por só um lugar – Através do Photoclick, tornando mais fácil o acesso a informações como valores, descrições e promoções.

O aplicativo apresentado teve seu levantamento de requisitos através de conversas com fotógrafos, para saber como fazem a divulgação do trabalho hoje em dia, após foi realizada a modelagem de dados de acordo com o estudo realizado e por fim o desenvolvimento do aplicativo, utilizando os recursos disponíveis e necessários para implementar a maioria das funções levantadas no estudo de caso.

O desenvolvimento foi realizado através do Framework Flutter da google, uma nova tecnologia que está ganhando seu espaço no mercado de desenvolvimento para aplicativos, devido a seu fácil desenvolvimento para *Front-End*, não exigindo muito da parte do *Back-End*. O Flutter usa a linguagem Dart como base para programação, sendo ela muito parecida com outras do mercado, como por exemplo PHP. Para emular a aplicação foi utilizado o Android Studio, onde o mesmo contém suporte ao Flutter, auxiliando assim no desenvolvimento e também possui o seu próprio emulador, para então testar o aplicativo, e para salvar os dados foram utilizados o Firebase Storage e o Cloud Firestore.

No decorrer do desenvolvimento a maior dificuldade encontrada foi se adaptar ao tipo de estrutura de desenvolvimento do Flutter, já que o mesmo foge um pouco dos conhecimentos obtidos no curso. Desta forma exigiu muita pesquisa para entender como funciona como um tudo, onde também foi necessário atualizar a versão do Framework para conseguir utilizar o *Upload* de imagens. Neste momento ocorreu muito retrabalho, já que foi necessário alterar outros métodos que não eram mais aceitos na última versão. Como o Flutter é muito voltado ao Front-End, também foi necessário ampliar meus estudos a respeito de UI, já que meu foco é trabalhar com a parte da lógica.

Para trabalhos futuros pretende-se implementar novas funcionalidades para o sistema, o que não foi possível devido a disponibilidade de tempo para a realização do trabalho, como por exemplo a opção de busca por localização de fotógrafos e a

finalização de uma contratação do fotógrafo. Outro trabalho futuro é a utilização de pagamento no aplicativo, onde poderá finalizar um contrato dentro do mesmo, conforme pesquisado por APIs do PagSeguro, por exemplo.

A utilização do SERVQUAL não foi possível, já que para aplicar no aplicativo desenvolvido, deve estar por um tempo sendo utilizado por usuários, para depois realizar a pesquisa de satisfação e avaliação do mesmo, desta forma também ficará para ser implementado em um trabalho futuro.

7. REFERÊNCIAS

APP STORE. **Uber**. Disponível em: <https://apps.apple.com/br/app/uber/id368677368>. Acesso em: 09 Nov. 2019.

APP STORE. **iFood - Delivery de Comida**. Disponível em: < https://apps.apple.com/br/app/ifood-delivery-de-comida/id483017239>. Acesso em: 08 Nov. 2019.

AVRAM, Abel. Infoq: 2016**. Google Firebase: back-end completo para aplicações web e mobile**. Disponível em: <https://www.infoq.com/br/news/2016/07/google-firebase>. Acesso em: 17 de out. de 2019.

FIREBASE. Firebase: 2019. **Documentação**. Disponível em: <https://firebase.google.com/?hl=pt-BR>. Acesso em: 17 de out. de 2019.

FLUTTER. Flutter: 2019. **Documentação**. Disponível em: <https://flutter.dev/docs>. Acesso em: 18 de out. de 2019.

MAGALHÃES, Túlio. Zup: 2019. **Flutter: tudo sobre o queridinho do google**. Disponível em: <https://www.zup.com.br/blog/flutter>. Acesso em: 18 de out. de 2019.

MEDIUM. Disponível em: < https://miro.medium.com/max/1024/1*iarfkgGGRItfb_GD7Vbn6g.jpeg>. Acesso em: 20 nov. 2020***.***

ORLANDI, Claudio. rocketseat: 2018. **Firebase: serviços, vantagens, quando utilizar e integrações**. Disponível em: < https://blog.rocketseat.com.br/firebase>. Acesso em: 17 de out. de 2019.

OGAWA, Marcia. Deloitte: 2018. **Global mobile consumer survey 2018**. Disponível em:

<https://www2.deloitte.com/content/dam/Deloitte/br/Documents/technology-media-telecommunications/Global-Mobile-Consumer-Survey-2018-Deloitte-Brasil.pdf>. Acesso em: 13 de set. de 2019.

RAMOS, Davidson. Blog da Qualidade: 2017. **SERVQUAL: um método para avaliar a qualidade do serviço**. Disponível em: <https://blogdaqualidade.com.br/servqual-um-metodo-para-avaliar-a-qualidade-do-servico>. Acesso em: 23 de out. de 2019.

RIBEIRO, Leandro. Devmedia: 2012. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML**. Disponível em: < https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acesso em: 16 de dez. de 2020.

SILVA, Paulo Cesar Barreto. Devmedia: 2009. **Diagrama de Classes UML**. Disponível em: <https://www.devmedia.com.br/diagrama-de-classes-uml/12251>. Acesso em: 08 de nov. de 2019.

WIKIPEDIA. Wikipedia: 2013. **Diagrama de Objetos**. Disponível em: <https://pt.wikipedia.org/wiki/Diagrama_de_objetos>. Acesso em: 09 de nov. de 2019.

WIKIPEDIA. Wikipedia: 2013. **Diagrama de Sequência**. Disponível em: <https://pt.wikipedia.org/wiki/Diagrama_de_sequ%C3%AAncia>. Acesso em: 15 de nov. de 2019.