FACULDADE E ESCOLA TECNICA ALCIDES MAYA

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

APLICAÇÃO MOBILE PARA CONTROLE DE FLUXO DE CAIXA

Desenvolvimento de um sistema mobile multiplataforma para auxílio no controle de fluxo de caixa para microempresa

RODRIGO RODRIGUES SCOTTI

Orientador: Prof ME. João Padilha Moreira

Porto Alegre - RS, 2020

RODRIGO RODRIGUES SCOTTI[1]

APLICAÇÃO MOBILE PARA CONTROLE DE FLUXO DE CAIXA

Desenvolvimento de um sistema mobile multiplataforma para auxílio no controle de fluxo de caixa para microempresa

Projeto de Pesquisa para conclusão da disciplina de Projeto II do **Curso superior de Tecnologia Em Sistemas para Internet.**

Orientador: Prof. Me. João Padilha Moreira[2]

Porto Alegre - RS

2020

[1] Acadêmico do Superior em Sistemas para Internet - email: rodrigo.scotti@alcidesmaya.edu.br

[2] Professor do Superior em Sistemas para Internet - email: joao_moreira@alcidesmaya.edu.br

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIAÇÕES

Android - Sistema operacional móvel da Google

API - Application Programming Interface.

B2C - Business to consumer.

Backend - Estrutura lógica de uma aplicação.

BloC - Business Logic Component.

CLI - Command Line Interface.

CRUD - Create, Read, Update, Delete.

Desktop - Computador de mesa.

Framework - Conjunto de bibliotecas para auxiliar na construção de uma aplicação.

Frontend - Estrutura visual de uma aplicação.

Github - Repositório de código online.

IOS - Sistema operacional móvel da Apple.

macOS - Sistema operacional para computadores da Apple.

MVP - Mínimo Produto Viável.

Software - Programa.

UI - User Interface.

UX - User Experience.

View - Tela de visualização do usuário.

WEB - World Wide Web

Windows - Sistema operacional para computadores da Google.

SUMÁRIO

RESUMO

A presente proposta objetiva a criação de uma aplicação mobile para controlar o caixa, com a necessidade do funcionamento nos dois sistemas operacionais mais utilizados nos smartphones atualmente (Android, IOS). Para o desenvolvimento do aplicativo, o cronograma foi dividido em 8 etapas. A primeira etapa foi realizado o planejamento, iniciando pela pesquisa comparativa com as aplicações semelhantes a deste projeto, em seguida foi realizado o planejamento de tecnologias a utilizar. Com o objetivo de entender melhor a necessidade da empresa, foi criado um briefing onde os funcionários puderam responder dúvidas sobre o projeto, em seguida foi realizado a delimitação do escopo e criado o mockup do layout com base nas respostas respondidas no briefing. Após a validação das telas da aplicação, foi iniciado o desenvolvimento das funcionalidades para que em seguida seja realizado os testes e a homologação com a empresa, tendo assim a primeira versão entregue do aplicativo. Com a automatização deste processo utilizando uma aplicação mobile, irá auxiliar e facilitar o cotidiano desta empresa, podendo realizar a gerência do caixa da empresa, tendo o controle de tudo que entra e saí para contabilizar os lucros e gastos de uma forma simples. Administração do estoque como forma de auxílio para identificar de forma prática quais produtos a empresa possui. Base de dados com todos os clientes da empresa para melhor atendimento e facilitar no registro de novos pedidos.

Palavras-chave: desenvolvimento mobile, automação, fluxo de caixa.

ABSTRACT

This proposal aims to create a mobile application to control the cashier, with the need to operate on the two most used operating systems in smartphones today (Android, IOS). For the development of the application, the schedule was divided into 8 stages. The first stage was the planning, starting with comparative research with applications similar to this project, then the planning of technologies to be used. In order to better understand the company's needs, a briefing was created where employees could answer questions about the project, then the scope was delimited and the layout mockup was created based on the responses answered in the briefing. After validation of the application screens, the development of functionalities was started so that tests and homologation with the company can be carried out, thus having the first version of the application delivered. With the automation of this process using a mobile application, it will assist and facilitate the daily life of this company, being able to manage the company's cash, having control of everything that enters and leaves to account the profits and expenses in a simple way. Inventory management as an aid to identify in a practical way which products the company has. Database with all customers of the company for better service and facilitate the registration of new orders.

Palavras-chave: mobile development, automation, cash flow.

RESUMEN

Esta propuesta tiene como objetivo crear una aplicación móvil para controlar el cajero, con la necesidad de operar sobre los dos sistemas operativos más utilizados en los teléfonos inteligentes en la actualidad (Android, IOS). Para el desarrollo de la aplicación, el cronograma se dividió en 8 etapas. La primera etapa fue la planificación, comenzando con la investigación comparativa con aplicaciones similares a este proyecto, luego la planificación de las tecnologías a utilizar. Con el fin de entender mejor las necesidades de la empresa, se creó un briefing donde los empleados podían responder preguntas sobre el proyecto, luego se delimitó el alcance y se creó la maqueta de diseño en base a las respuestas respondidas en el briefing. Luego de la validación de las pantallas de la aplicación, se inició el desarrollo de funcionalidades para que se puedan realizar pruebas y homologaciones con la empresa, teniendo así entregada la primera versión de la aplicación. Con la automatización de este proceso mediante una aplicación móvil, se asistirá y facilitará el día a día de esta empresa, pudiendo administrar el efectivo de la empresa, teniendo el control de todo lo que entra y sale a contabilizar las ganancias y gastos de una manera sencilla. La gestión de inventarios como ayuda para identificar de forma práctica qué productos tiene la empresa. Base de datos con todos los clientes de la empresa para un mejor servicio y facilitar el registro de nuevos pedidos.

Palavras-chave: desarrollo móvil, automatización, flujo de caja.

1. INTRODUÇÃO

Com o avanço da tecnologia, cada vez mais as empresas buscam se adequar no mercado, criando aplicativos como facilidade para o cotidiano das pessoas que utilizam smartphones. Segundo pesquisa realizada no período de dezembro de 2009 a março de 2020 do site Statista, existe 2,87 milhões de aplicativos disponíveis na Google Play Store (CLEMENT, 2020), esta estatística comprova que cada vez mais temos facilidade para encontrar aplicativos que possam resolver a problemática de uma pessoa física tanto quanto pessoa jurídica, possibilitando utilizar diversas funcionalidades e até mesmo criar integrações com outros softwares.

Cada ferramenta possui a sua particularidade como por exemplo: softwares com versões gratuitas por um curto período, ferramentas que liberam poucas funcionalidades de graça e para ter acesso a todas as funcionalidades precisa pagar mensalidades, entre outros. Entre planos específicos e estratégias que as empresas utilizam para que o software gere lucros, existem também barreiras como forma de dificuldade na utilização de suas ferramentas, como por exemplo: dificuldade com a usabilidade pela grande complexidade, aplicativos de índoles duvidosas, entre outros que por muitas vezes geram mais problemas do que soluções.

A criação de um aplicativo próprio faz com que a pessoa ou empresa tenha suas funções específicas de forma mais clara, tendo funcionalidades que serão utilizadas. Além da praticidade de conter somente o necessário, é possível validar e acompanhar o andamento da criação de seu aplicativo para poder aprovar ou apontar possíveis melhorias do layout ou até mesmo do funcionamento da aplicação.

1.1. JUSTIFICATIVA

Uma microempresa de distribuição e revenda de perfumes necessita de uma ferramenta que automatize o processo de anotações de novos pedidos e saídas com gastos na compra de novos produtos, pois atualmente a empresa realiza sua gestão financeira em um bloco de anotações. Por se tratar de um papel, acaba sendo muito fácil de ser danificado, podendo gerar a perda de uma venda e talvez até a perda de clientes.

Com a automatização do fluxo de caixa utilizando uma aplicação mobile, se abre um leque de oportunidades futuras para a melhor utilização destes dados, como a administração do estoque como forma de auxílio para identificar de forma prática quais

produtos a empresa possui. Uma base de dados com todos os clientes da empresa para melhor atendimento e facilidade no registro de novos pedidos. Podendo também ser integrado um e commerce, onde a ferramenta consiga atender diretamente o consumidor final, trabalhando com o modelo de negócio B2C. Todas essas funcionalidades são oportunidades futuras que poderão ser desenvolvidas com o tempo, dependendo da necessidade. No momento, o foco é o desenvolvimento de um MVP para facilitar a gestão financeira através do fluxo de caixa.

A diversidade de aplicativos como objetivo da utilização de anotações pelo celular é extensa, porém trazendo para a área de vendas e finanças, acaba se tornando desnecessário por não ser possível criar métricas a partir das vendas realizadas.

Dos diversos aplicativos existentes no mercado atualmente, os mais conhecidos na área de finanças acabam sendo muito complexos, tendo diversas funcionalidades que não auxiliam em nada para o controle de caixa necessário pela empresa, e muitos desses aplicativos não são gratuitos, é necessário pagar assinaturas mensalmente para utilizá-los.

Por esses motivos, será criado um aplicativo mobile para que a microempresa de distribuição e revenda de perfumes, tenha uma aplicação que contemple todas as funcionalidades necessárias e somente o que precisar.

1.2. OBJETIVOS

O escopo do projeto tem como objetivo o desenvolvimento de um aplicativo mobile, visando a criação de um MVP para auxiliar especificamente uma microempresa de distribuição e revenda de perfumes, situada na cidade de Porto Alegre - RS.

1.2.1. OBJETIVO GERAL

A presente proposta objetiva a criação de uma aplicação mobile para controle de fluxo de caixa, com a necessidade do funcionamento nos dois sistemas operacionais mais utilizados nos smartphones atualmente (Android, IOS).

O projeto almeja alcançar as expectativas da empresa para que o aplicativo auxilie com a problemática.

1.2.2. OBJETIVOS ESPECÍFICOS

- Pesquisa comparativa com as aplicações semelhantes a deste projeto;
- Planejamento de tecnologias a utilizar;
- Briefing e delimitação do escopo;
- Mockup do layout aprovado;
- Desenvolvimento do MVP;

1.3. MOTIVAÇÃO

Diante da atual realidade que vivenciamos com a pandemia por causa do novo coronavírus, micro empresas novas e outras que já estão no mercado, encontram dificuldades para realizar o atendimento diretamente ao consumidor. Somente em abril, 70% das empresas no Rio Grande do Sul foram impactadas no andamento do seu negócio (MÜLLER, 2020).

Com todas as medidas como forma de prevenção do governo, obriga as empresas a buscarem novas alternativas, como a venda online.

O compartilhamento de conhecimentos entre as áreas, possibilita a inovação dentro da empresa dando oportunidade no desenvolvimento de novos projetos que pode auxiliar a manter uma empresa viva, mesmo em um momento tão complicado.

2. REFERENCIAL TEÓRICO

Um dos motivos que acabam levando às empresas a falência é a má gestão financeira, que por muitas vezes os(as) empresários(as) acabam levando muito tempo para perceber a importância de aplicar os mecanismos financeiros, se tornando incapaz na tomada de decisões estratégicas para o seu negócio (SERASA, 2020).

O fluxo de caixa é um dos principais mecanismos, auxiliando na análise e gerenciamento da liquidez que a empresa possui, sendo fundamental na organização financeira de uma empresa, pois visa a apuração e projeção do saldo disponível, possibilitando a existência de capital de giro para eventuais gastos (SEBRAE, 2019).

Segundo Maria de Paula (2018, p. 4):

> Fluxo de caixa é uma ferramenta com finalidade de controle das movimentações financeiras, débito e crédito de receitas, com tempo e período determinado, assim evidenciando as alterações efetuadas na conta caixa. Trazendo para a empresa facilidade e controle, apurando informações quais indicaram exatamente o valor de suas obrigações ou lucro do determinado período.

Realizando a gestão financeira através do fluxo de caixa, possibilita à empresa obter segurança diante as suas necessidades. Alguns dos exemplos apontados pelo fluxo de caixa são os prazos de retorno do investimento, quantia de receita futura, informativo de gastos que poderiam ser evitados e até mesmo o detalhamento apurado da rentabilidade da empresa.

Uma das formas para se aplicar o fluxo de caixa dentro de uma empresa que está iniciando é na criação de uma planilha, possibilitando que o funcionário consiga listar todos os gastos da empresa, porém pensando em médio a longo prazo pode acabar sendo trabalhoso e difícil trabalhar com uma planilha extensa e pesada. Neste caso, é possível também utilizar este mecanismo financeiro através de um aplicativo. A automatização de todo o processo de fluxo de caixa em um software, favorece a geração de recursos humanos e a economia de tempo. Auxiliando também no controle de despesas, no acompanhamento dos lucros e gastos da empresa através de relatórios (VASQUE e SGOTI, 2019).

Para automatizar este processo e desenvolver uma aplicação para a microempresa, foi realizada uma pesquisa dentro da empresa com foco na descoberta do dispositivo mais utilizado pelos funcionários. Por ser uma empresa pequena, realizando apenas uma conversa com a gerente foi possível identificar que o smartphone com sistema operacional Android é o mais utilizado dentre os funcionários. A partir desta análise simples, foi escolhido o desenvolvimento mobile para automatização do fluxo de caixa.

A subseção 2.1 detalha sobre a arquitetura de software utilizada para o desenvolvimento do aplicativo mobile.

2.1. ARQUITETURA DE SOFTWARE

Por muitos anos esta definição foi debatida na área de tecnologia, atualmente podemos definir como o processo responsável para a definição de soluções viáveis para uma tecnologia, levando em consideração pontos como viabilidade, escalabilidade, performance, interoperabilidade e compatibilidade (IGTV, 2019). Para Fowler, a arquitetura de software é definida como (2003, p.2):

> Nos projetos de software mais bem-sucedidos, os desenvolvedores especialistas que trabalham nesse projeto têm um entendimento compartilhado do design do sistema. Esse entendimento compartilhado é chamado de 'arquitetura'. Esse entendimento inclui como o sistema é dividido em componentes e como os componentes interagem por meio de interfaces. Esses componentes geralmente são compostos de componentes menores, mas a arquitetura inclui apenas os componentes e interfaces que são entendidos por todos os desenvolvedores.

A falta de organização e controle de todo o processo de desenvolvimento de software, pode resultar em um conjunto de riscos aumentando a possibilidade do cancelamento do projeto. A arquitetura de software tem como objetivo a redução de riscos relacionados a processos e atividades de um projeto, auxiliando também na construção de aplicações flexíveis (NHIMI, 2016).

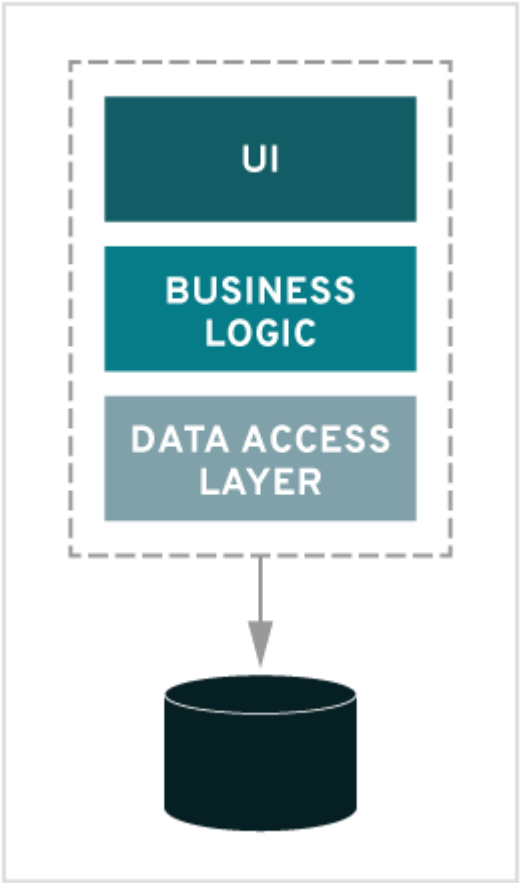
Dentre vários padrões de arquitetura, existe uma em específico muito utilizada e debatida sobre seus casos e necessidades de utilização, monolito, que para este projeto foi tomada a decisão da utilização desta arquitetura por se tratar de uma arquitetura adequada para aplicações pequenas.

A seção a seguir realiza o detalhamento sobre a arquitetura monolito e a arquitetura modular.

2.1.1. Monolito

A arquitetura monolítica possui uma aplicação responsável por todo o processo, abaixo uma imagem representando de forma breve esta arquitetura:

Figura 1 - Exemplo de um sistema monolito



Fonte: Red Hat[3]. (2020)

Essa arquitetura é baseada em uma estruturação de aplicativos únicos, contemplando todos os processos necessários para a construção de um programa web. Um software monolítico deve ser auto-suficiente, tendo uma base de dados única, centralizando todos os dados da aplicação, o backend é mantido em apenas um serviço, sendo mantido toda a regra de negócio em apenas uma aplicação backend.

A camada de visualização do projeto, também conhecida como frontend, pode ser separada em uma aplicação a parte (utilizando frameworks como react, vue, entre outros) se comunicando com o backend através de uma API ou ser estruturado na mesma aplicação do backend. Dessas duas formas é possível trabalhar com a arquitetura monolítica no frontend.

Segundo Marina Medeiros Lima (2019), faz uma breve descrição sobre a abordagem monolítica:

> O termo “monolítico” é utilizado quando o sistema é construído como uma única unidade: todos seus componentes estão juntos em uma única plataforma, por

[3] Disponível em: <https://www.redhat.com/pt-br/topics/microservices/what-are-microservices>. Acessado dia 12 Out. 2020.

> exemplo um único cliente com sua UI (User Interface, interface do usuário), suas regras de negócio e sua camada de acesso à dados, e arquivo Java WAR.

Podemos citar os maiores benefícios de uma aplicação monolítica como a facilidade para desenvolver pois as ferramentas atualmente disponíveis geralmente possuem o foco para esta arquitetura, tendo também como benefício a entrega e a praticidade para escalar réplicas, pois se trata de apenas uma aplicação para todo o projeto. No entanto, esta abordagem pode acabar sendo difícil de se trabalhar ao decorrer do projeto com novas versões e com equipes maiores, podendo ter maiores dificuldades no entendimento do código. A entrega contínua que no início era rápida e prática, pode acabar se tornando demorada pela quantidade de novas versões adicionadas ao projeto (RICHARDSON, 2020).

2.1.3. Modular

A arquitetura de projeto orientado a módulos, sua estrutura auxilia na escalabilidade e manutenibilidade de uma aplicação por permitir trabalhar com recursos de forma separado e independente (BECAPTAIN, 2020).

Uma das grandes vantagens do Flutter é a praticidade na decisão na arquitetura de projeto possível ao iniciar um projeto, e arquitetura modular acaba se tornando a melhor decisão ao pensar que o escopo do projeto aqui proposto, que terá a possibilidade da criação de novas funcionalidades.

Para utilizar esta arquitetura em um projeto mobile no Flutter, é possível utilizar a biblioteca flutter_modular. Segundo a Flutterando (2020), comunidade brasileira de Flutter:

> A Modular oferece várias soluções Flutter-adequadas para lidar com esse problema, como injeção de dependência, sistema de roteamento e o sistema de "singleton descartável" (ou seja, a Modular descarta o módulo injetado automaticamente, pois está fora do escopo).

Além do sistema de roteamento, a injeção de dependência da Modular suporta qualquer sistema de gerenciamento de estado (FLUTTERANDO, 2020).

2.2. FERRAMENTAS ABORDADAS

No capítulo anterior, foi possível introduzir sobre a arquitetura utilizada para a construção da aplicação, podendo manter um padrão de organização para o projeto. Nesta seção, será abordado sobre as ferramentas utilizadas para o desenvolvimento do aplicativo mobile.

Os frameworks vem ganhando cada vez mais uma maior popularidade na comunidade de desenvolvimento pela sua maior aproximação a performance de um aplicativo nativo, contendo apenas uma base de código para multiplataformas.

Na subseção 2.2.1 será realizada uma breve introdução sobre o framework Flutter, a linguagem Dart desenvolvida pela Google, a ferramenta MobX e o Slidy. Na subseção 2.2.2 serão abordados as ferramentas do Firebase, como Cloud Firebase para armazenamento dos dados e Authentication Firebase para controle de acesso de usuário na ferramenta.

2.2.1 Flutter

Flutter é um framework de código aberto criado pela Google. Foi desenvolvido na linguagem Dart e atualmente é mantido pela comunidade. Esta ferramenta possibilita a criação de aplicações mobile, web e desktop utilizando uma única base de código. O Flutter teve sua primeira versão (1.0) lançada em 4 de dezembro de 2018 e possui milhares de aplicativos em produção atualmente utilizando esta tecnologia (FLUTTER, 2020). Atualmente se encontra na versão 1.22.4 publicada em 13 de novembro de 2020 em seu repositório do Github (Sosinski, 2020).

O objetivo do framework Flutter é a construção da interface para o usuário a partir de Widgets. Diferente dos demais frameworks no mercado atualmente, o Flutter possui seu próprio mecanismo de renderização de widgets, contendo estrutura do aplicativo e de navegação, botões, inputs, widgets de alerta e painéis de diálogos, displays informativos e layouts (Flutter, 2020).

Em 2019, Robson Rosa em seu projeto de conclusão do curso de Sistemas para Internet, realizou um estudo comparativo entre dois frameworks mais utilizados no desenvolvimento mobile multiplataformas, Flutter e React Native. Em sua pesquisa, foi utilizado 5 critérios para chegar na conclusão de qual framework possui mais benefícios dentro do escopo definido de seu projeto. No qual se teve a conclusão de que o Flutter possui uma melhor entrega na maior parte dos pontos comparado com o React Native. Segundo Rosa:

> A conclusão que pudemos obter de acordo com as avaliações realizadas é que, no contexto deste projeto, a ferramenta que melhor auxiliou o desenvolvedor durante o período de desenvolvimento foi o framework Flutter, devido principalmente à consistência de emulação da aplicação e os diversos componentes já construídos que a ferramenta dispõe para seu utilizador.

Tendo em vista o resultado contido na pesquisa realizada entre as tecnologias, será utilizado o framework Flutter para a automatização do processo proposto neste projeto.

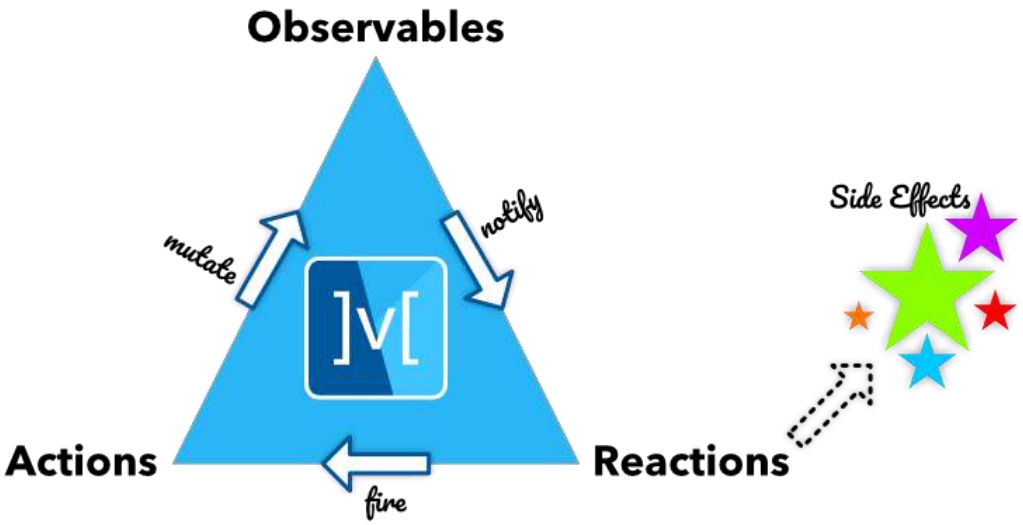
2.2.1.1 Slidy

Slidy é um gerenciador de pacotes CLI e gerador de módulos (FLUTTERANDO, 2020). Com a utilização do Slidy se torna prático e rápido a criação de um projeto implementando uma arquitetura Modular ou BloC, pois o Slidy se responsabiliza de criar todos os módulos e configurações iniciais. Além disso, é possível também criar novos módulos, views, BLoCs (caso a arquitetura escolhida seja BLoC), controllers e testes automatizados.

2.2.1.2 MobX

MobX é uma biblioteca com o objetivo de tornar o gerenciamento de estado simples e escalável. MobX possui três conceitos básicos (MOBX, 2020):

Figura 2 - Conceitos básicos do MobX



Fonte: mobx[4]. (2020)

[4] Disponível em: <https://pub.dev/packages/mobx>. Acessado dia 02 Dez. 2020.

- Observables: Os observables são simples e escaláveis, são representados como o reactive-state da aplicação, podendo ser consumidas pela UI ou por outros Observables (MOBX, 2020).
- Actions: Actions é a forma para modificar os observables de forma não direta. Também, realiza o agrupamento de todas as notificações, garantindo que as alterações sejam notificadas somente ao concluir a ação (MOBX, 2020).
- Reactions: Reactions é responsável por observar o sistema e notificar novas modificações do observable (MOBX, 2020).

2.2.2 Cloud Firebase

O Cloud Firebase é um banco de dados que mantém os dados em sincronia por meio de listeners em tempo real. Criado pela Google, o Cloud Firebase é flexível e escalonável para desenvolvimentos móveis, web e servidores a partir do Firebase e do Google Cloud Platform. Oferece também suporte off-line, auxiliando na criação de aplicativos mais responsivos (FIREBASE, 2020).

Os dados são armazenados em documentos mapeados de campos para valores. Os documentos permanecerão gravados dentro de coleções, que basicamente são containers de documentos que auxiliam na organização e na criação de consultas. Cada campo em um documento pode conter apenas um tipo, como por exemplo strings (para armazenar texto), números e timestamp (para armazenar data e hora). Os documentos podem conter mais de um campo. É possível realizar a criação de subcoleções nos documentos (FIREBASE, 2020).

2.2.3 Firebase Authentication

Firebase authentication é uma ferramenta da Google que fornece serviços de autenticação de fácil utilização, tanto para aplicações web quanto mobile. O Firebase Authentication dá suporte em autenticações com email e senha, número de telefone, redes sociais como Google, Facebook, Twitter, autenticação anônima, entre outros (FIREBASE, 2020).

Para que o usuário possa acessar o aplicativo, deve ser necessário que o aplicativo contenha as credenciais, no caso, que o usuário possua uma conta já cadastrada, podendo ser endereço de e-mail (para autenticação via e-mail e senha) ou um token do

OAuth (para autenticação via redes sociais). Contendo as informações, deve ser passado os dados para o Firebase que irá se encarregar de validar as credenciais, e logo após será retornado uma resposta para o cliente, podendo ser uma mensagem de sucesso ou de erro (FIREBASE, 2020).

3. LAYOUT DA APLICAÇÃO

Nesta etapa, foi realizado uma pesquisa em conjunto com a microempresa através de briefing para melhor entendimento da visão do cliente sobre o aplicativo, estudo de cases semelhantes visando adicionar uma boa usabilidade, desenvolvimento do wireframe da aplicação através de pesquisas U.X e do briefing, podendo validar de forma prática e simples o layout do projeto e assim tendo a possibilidade de realizar a estruturação do mockup contendo todo projeto em uma escala real para validação do layout.

Nas seções a seguir, detalha sobre cada etapa realizada neste processo.

3.1. Briefing

O briefing é um documento que reúne todas as ideias que o cliente tem para a realização de determinado projeto, seja um site, uma campanha de marketing, uma peça publicitária, uma identidade visual, entre outros. Com esta ferramenta, podemos agrupar diversas informações importantes sobre o cliente e sobre a expectativa para o projeto, mantendo uma forma organizada e simples para poder consultar sempre que necessário (SOUZA, 2019).

Para esta etapa, separei algumas perguntas em dois tópicos: o primeiro realiza algumas perguntas sobre a empresa, e no segundo tópico é esclarecido alguns pontos sobre o projeto.

3.1.1 Sobre a empresa

Nesta etapa foi levantado algumas dúvidas sobre a empresa a fim de entender melhor sobre a história, processo de venda e potenciais concorrentes que possam existir. Foi feito um questionário de 5 perguntas para melhor esclarecimento. Abaixo, as perguntas e respostas realizadas:

Qual o histórico da empresa? Empresa aberta dia 1/06 visando venda de perfumes de 15 ml, 100 ml, hidratante corporal, produto de apoio para venda do executivo como flaconetes, catálogos, fitas olfativas, canetas, sacolas (caso o cliente solicite) e produtos para cabelo na linha home care (não na linha profissional).

Qual o processo de venda? Suprir material para o executivo e venda ao público em geral. Vendas online por mídias sociais e lojas físicas. A empresa também vai até o cliente.

Qual o ramo de negócio? Perfumaria, creme capilar, hidratante e futuramente maquiagens.

Existe concorrência? Existem concorrentes diretos que vendem perfumes de 15 ml por um preço mais elevado, e outros concorrentes indiretos mas que já são lojas muito bem consolidadas atualmente no mercado de perfumaria.

Quais são os pontos positivos e negativos da concorrência? Como ponto positivo das empresas concorrentes é a notável eficiência na entrega dos produtos para as filiais. Já como ponto negativo, podemos citar que a qualidade dos perfumes, como por exemplo a com pouca fixação deixa a desejar, tendo valores mais elevados.

3.1.2 Esclarecimento do projeto

Como forma de entender a visão da empresa sobre o projeto, foi realizado um questionário com sete perguntas. Abaixo podemos acompanhar todas as questões respondidas:

Qual o objetivo do projeto? Facilitar a entrada e saída das finanças.

Problemática do projeto? Nossa empresa não realiza uma gestão financeira.

Região atingida? Região de Porto Alegre e grande Porto Alegre.

Público alvo? Entre os 20 até os 50 anos.

Qual será a forma e frequência de uso do aplicativo? O aplicativo será utilizado diariamente.

Posicionamento no mercado? Projeto interno da empresa.

Qual é a expectativa do cliente? Unificação da informação, tendo todos os registros de entrada e saída. Contendo todas as informações sobre o tipo de pagamento (dinheiro, cartão) e podendo visualizar todos os registros mensais.

3.2. Trabalhos relacionados

Dentre as etapas de desenvolvimento de uma aplicação, está a etapa de análise de mercado, onde é feito pesquisas para identificar possíveis aplicativos que já possuem a mesma características, assim, amadurecendo a real necessidade em um início de projeto.

Esta seção apresenta a pesquisa realizada que tem como objetivo identificar trabalhos relacionados ao tema trazido neste projeto. Nesta pesquisa realizada através do

site Google Acadêmico utilizando as frases software controle de caixa e software fluxo de caixa, os trabalhos e artigos publicados foram filtrados por datas, limitando para publicações realizadas de 2015 até 2020. Depois da análise do título, foi feita a leitura do resumo, da introdução e por fim, da conclusão do trabalho, para poder ter o entendimento do que era o projeto, o que se pretendia alcançar realizando o estudo e se esta meta foi alcançada.

Foi possível identificar três trabalhos relacionados:

Desenvolvimento de Software para Controle de Fluxo de Caixa para Pequena Empresa: Foi desenvolvido um software nativo desktop para o controle do fluxo de caixa em uma pequena empresa, auxiliando nas decisões no processo de gestão financeira e melhor controle de despesa (VASQUE, SGOTI, 2019).

Tecnologia Java no desenvolvimento de Sistema de Vendas para Controle de Caixa e Estoque: Este artigo teve como objetivo apresentar as etapas de desenvolvimento de um software nativo desktop, utilizando a automação do controle de fluxo de caixa de uma empresa do ramo de vendas como escopo da ferramenta (ZENZELUK, PASTERNAK, BINI, 2015).

Análise e Desenvolvimento de um Sistema Offline para Auxílio Gerencial da Pecuária de Corte e Fluxo de Caixa Simples para Pequenas Propriedades Rurais: O projeto descreve a realização do desenvolvimento de uma aplicação web sem a necessidade de conexão com a internet para pequenas propriedades rurais. Onde o software tem como objetivo o controle de fluxo de caixa e auxílio no gerenciamento da pecuária (CARVALHO, 2019).

Tabela 1: Descrição do documento money transaction

	Qual o objetivo do projeto?	Quais são as funcionalidades?	O projeto foi desenvolvido para dispositivos mobile?
Aplicação mobile para controle de fluxo de caixa	Criação de um aplicativo mobile para automação do processo de fluxo de caixa.	Controle de autenticação de usuário; Controle de fluxo de caixa;	Sim
Desenvolvimento de Software para Controle de Fluxo de Caixa para	Desenvolvimento de um software nativo desktop para o controle do fluxo de	Controle de fluxo de caixa;	Não

Pequena Empresa	caixa em uma pequena empresa.		
Tecnologia Java no desenvolvimento de Sistema de Vendas para Controle de Caixa e Estoque	Objetiva apresentar as etapas de desenvolvimento de um software nativo desktop, criando um sistema de fluxo de caixa de uma empresa do ramo de vendas.	Gerenciamento de vendas, funcionários e produtos;	Não
Análise e Desenvolvimento de um Sistema Offline para Auxílio Gerencial da Pecuária de Corte e Fluxo de Caixa Simples para Pequenas Propriedades Rurais	Desenvolver um sistema para gerenciamento da pecuária de corte e o fluxo de caixa simples sem necessidade de conexão com a Internet para pequenas propriedades rurais.	Controle individual dos animais, das pesagens, dos manejos sanitários, da compra e venda de animais, do fluxo de caixa simples; Emissão de relatório financeiro, relatório de ciclo reprodutivo e relatório de baixas;	Não

Fonte: Autor. (2020)

Em comparação com os objetivos principais deste trabalho, o projeto 2 possui uma série de funcionalidades como cadastro de fornecedores, funcionários e transportadora, são requisitos que não serão necessários para o desenvolvimento do projeto aqui descrito. O projeto 3 é voltado especificamente para o gerenciamento rural. Por fim, todos os 3 trabalhos comparados aqui não são voltados para o desenvolvimento mobile, a utilização dos softwares nos trabalhos são voltados diretamente para desktops.

3.3. Wireframe

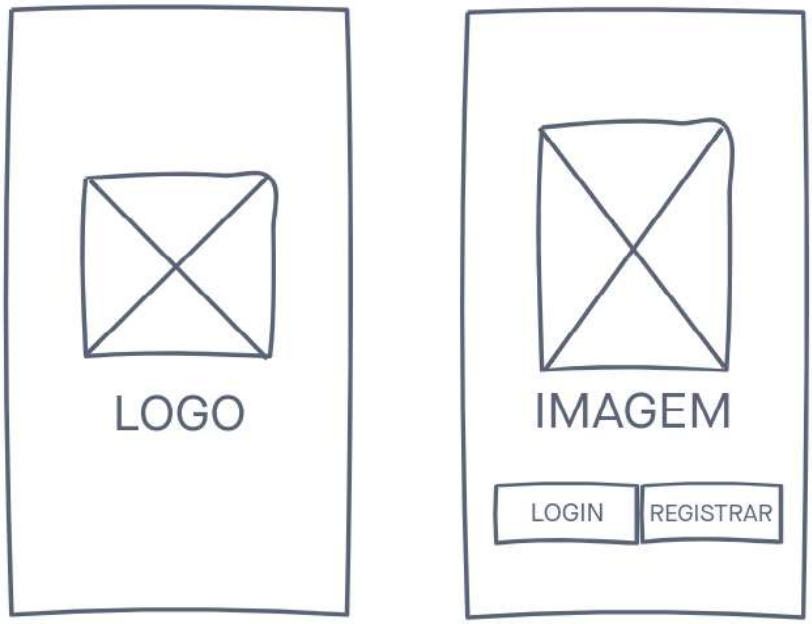
Para que seja feito um esboço das telas que estarão presentes dentro do aplicativo, foi criado um wireframe contendo as telas principais do projeto. PINHEIRO (2016, p. 54), explica que:

> Um wireframe é a representação visual de uma interface de usuário, desprovido de qualquer design visual. Ele é usado por UX designers para definir a hierarquia dos itens e definir o que os itens nesta página devem comunicar baseado nas necessidades do usuário.

É possível criar o esboço das telas de um aplicativo utilizando ferramentas online que auxiliam para a criação do wireframe e facilitam na validação com a equipe e nas possíveis modificações. A escolha pela ferramenta InVision para a criação do wireframe foi especialmente pela facilidade de criação de telas pela ferramenta online, tendo a possibilidade de compartilhar com outras pessoas e salvar os dados na nuvem e exportar todas as telas como imagem. O InVision permite que você sincronize todas as telas do seu aplicativo em uma única ferramenta online, possibilitando a prototipação da mesma e auxiliando na validação e realização de testes de usabilidade (SILVA, 2020).

Abaixo, imagem com todas as telas desenvolvidas através da ferramenta InVision.

Figura 3 - Telas iniciais do aplicativo



Fonte: Autor. (2020)

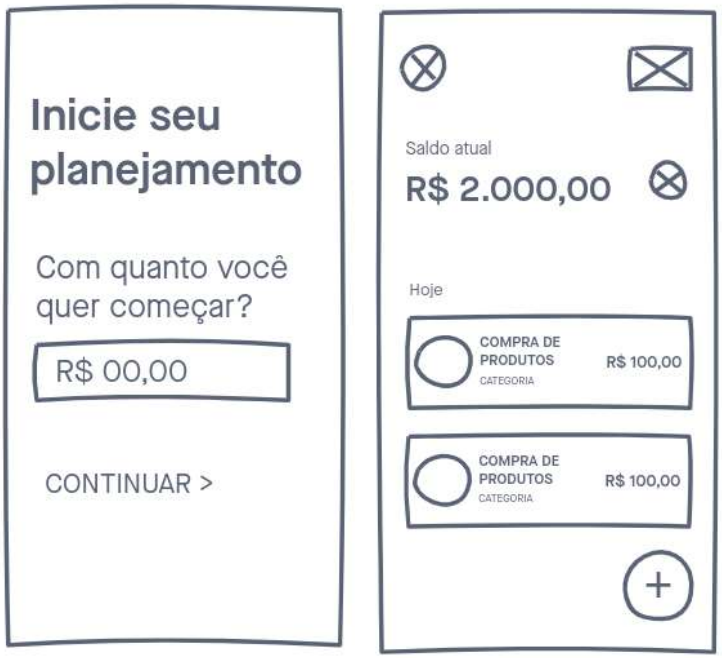
As duas telas acimas representam as telas iniciais do aplicativo, sendo a primeira, posicionada a esquerda, uma splash screen (tela que o usuário visualiza quando o aplicativo está carregando) e a segunda tela, posicionada a direita, a tela inicial quando o usuário não estiver logado, contendo uma imagem e dois botões para que o usuário faça o login ou se registre no aplicativo.

Figura 4 - Tela para o usuário se registrar e fazer o login

Fonte: Autor. (2020)

Para que o usuário consiga utilizar a ferramenta de uma forma mais segura, será criado um sistema de cadastro de usuário e login, para isso, foi criado esses dois esboços dando um exemplo de como deve ficar as telas onde o usuário deve fazer o login e se cadastrar no aplicativo.

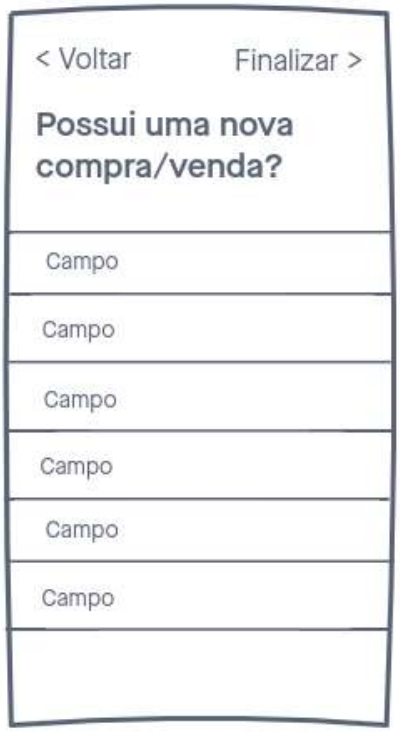
Figura 5 - Tela quando usuário estiver logado



Fonte: Autor. (2020)

Ao realizar o login, o usuário deve informar para a ferramenta toda sua quantia atual em dinheiro para que ao decorrer dos cadastros de seus gastos, o valor seja atualizado automaticamente e se espelhando na realidade. Para que o usuário consiga visualizar seu saldo e suas últimas transações, deve ser mostrado essas informações na página inicial, também para facilitar a utilização do aplicativo, é importante ter o botão de cadastro de novas entradas e gastos na posição inferior direita da tela.

Figura 6 - Tela para registrar nova transação



Fonte: Autor. (2020)

Para que o usuário possa cadastrar seus novos registros, tanto de entrada quanto de saída, será necessário ter uma tela na qual tenha todos os campos necessários como o valor da transação, descrição e a data de quando foi efetuado o pagamento. No final, deve ser possível salvar os dados preenchidos através de um clique em um botão, na tela acima, o usuário consegue clicando no botão “Finalizar”.

3.4. Mockup

Com o recurso da ferramenta Adobe XD, foi possível criar as telas do aplicativo mobile com base em mockups, que é a representação final do produto de forma estática. Esta ferramenta auxilia na apresentação do projeto, tendo a facilidade de realizar ajustes nas telas a partir de feedbacks antes de ser iniciado o desenvolvimento do produto.

Adobe XD é uma ferramenta da Adobe compatível com o sistema operacional Windows e macOS que ajuda na criação de telas de uma forma fácil e da prototipação, podendo representar o fluxo de uso de um usuário para, possibilitando a navegação entre telas.

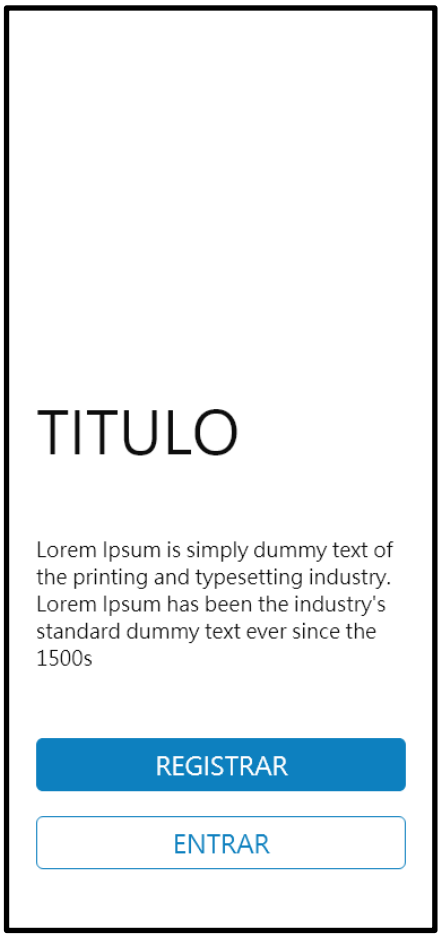
Abaixo, as telas do aplicativo aprovado pelo cliente. As bordas ao redor de cada figura são somente para demonstrar a dimensão final de cada imagem.

Figura 7 - Splash Screen e Tela inicial

Fonte: Autor. (2020)

A imagem número 7 demonstra a splash screen da aplicação, tela exibida ao abrir o aplicativo enquanto é carregado o mesmo.

Figura 8 - Tela de acesso



Fonte: Autor. (2020)

Quando o usuário abrir o aplicativo e não estiver logado, a tela, conforme demonstrado na imagem 8, será exibida para que o usuário realize o login ou se cadastre caso não possua acesso. Ao clicar no botão “REGISTRAR”, será redirecionado para a tela de registro de usuário no sistema e ao clicar no botão “ENTRAR”, será redirecionado para a tela de login, na qual deverá preencher os seus dados corretamente.

Figura 9 - Tela de Login

Fonte: Autor. (2020)

A figura 9 representa a tela de login do usuário. O campo “Usuário” será onde o usuário deve preencher com o seu usuário de e-mail e no campo “Senha” deve ser preenchido com a senha correta cadastrada no sistema. O teclado do campo “Usuário” deverá ser diferente, contendo o formato de e-mail. O campo senha conforme preenchido, não deverá ser substituído por caracteres indefinidos para que não seja possível ser visualizado por outras pessoas.

Ao clicar no botão “ENTRAR”, deve ser feita uma validação dos campos antes de efetuar o login do usuário. Em caso de o usuário ou senha não estiverem corretos, deve ser informada uma mensagem para o usuário dizendo que o usuário não existe (em caso do e-mail estiver incorreto) ou que a senha está errada (em caso do usuário existir mas somente a senha estiver errada). Caso um ou todos os campos não estejam preenchidos ao clicar no botão, deve ser informada uma mensagem para o usuário informando que deve ser preenchidos.

O texto final “Não tenho uma conta. Cadastrar” contém o botão no texto “Cadastrar” com a função de redirecionar o usuário para a tela de cadastro.

Figura 10 - Tela Registre-se

Fonte: Autor. (2020)

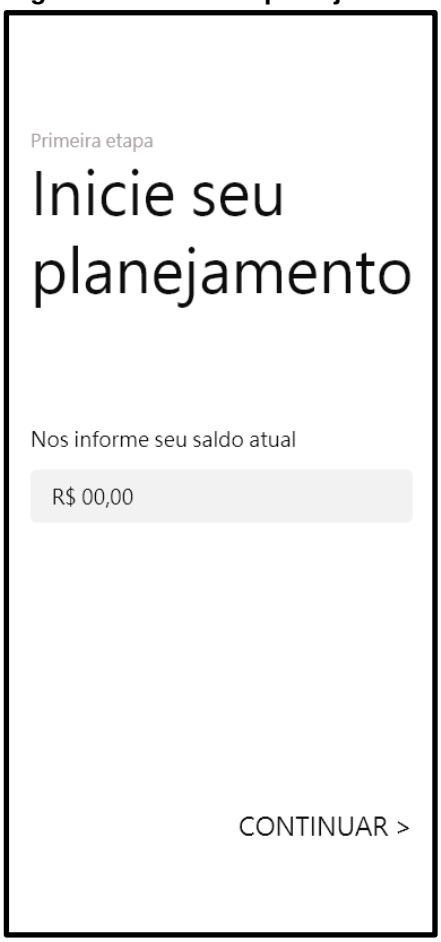
Caso o usuário não possua acesso ao aplicativo, deve ser renderizada a tela de registro de usuário, conforme representada na figura 10. O campo “Usuário” deve ser preenchido com o e-mail. O último campo “Senha” deve ser preenchido com a senha de login, de forma oculta para que outras pessoas não consigam visualizar. Ao clicar no botão “CADASTRAR” o cadastro do usuário deve ser validado e caso todos os campos estejam preenchidos de forma correta, deverá ser registrado os dados na base de dados, realizado o login no sistema e redirecionado para a tela de cadastro do valor inicial.

O campo de e-mail deve ser único no sistema, em caso da tentativa de cadastro de usuário de e-mail já existente, deverá ser mostrado uma mensagem informando que o usuário já existe, impossibilitando o acesso dentro da ferramenta.

No caso da tentativa de cadastro sem o preenchimento dos campos, deverá ser negado o acesso ao aplicativo e o usuário deverá receber uma mensagem solicitando o preenchimento de todos os campos.

O texto final “Já tenho uma conta. Entrar” contém o botão no texto “Entrar” com a função de redirecionar o usuário para a tela de login.

Figura 11 - Início de planejamento



Fonte: Autor. (2020)

Ao criar uma nova conta, o usuário será redirecionado para a tela de início de planejamento, conforme a figura 11. Nesta tela deve ser possível permitir que o usuário preencha no campo a quantidade de dinheiro atualmente.

O botão “CONTINUAR” ao ser clicado deve realizar uma validação no campo. Caso o campo esteja vazio, deve ser informada uma mensagem para o usuário, solicitando que preencha o campo. Em caso do campo ser preenchido, deverá ser armazenado o dado na base de dados e o usuário redirecionado para a tela inicial.

Figura 12 - Tela inicial

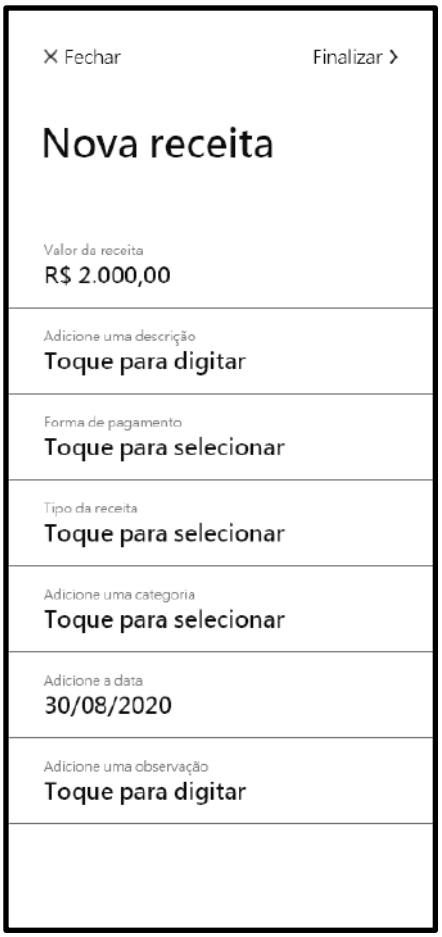
Fonte: Autor. (2020)

A tela inicial, conforme representada na imagem número 12, contém todas as informações do fluxo de caixa como o valor atual do usuário e uma listagem de suas transações ordenada por último registro.

O usuário poderá visualizar detalhes sobre cada entrada e saída em seu caixa clicando nos registros listados na tela.

Na parte inferior direita da tela, contém um botão no qual possui a funcionalidade de adicionar nova entrada ou saída da caixa. Ao clicar no botão, o usuário será redirecionado na tela de cadastro de fluxo de caixa.

Figura 13 - Tela de registro de nova entrada ou saída do caixa



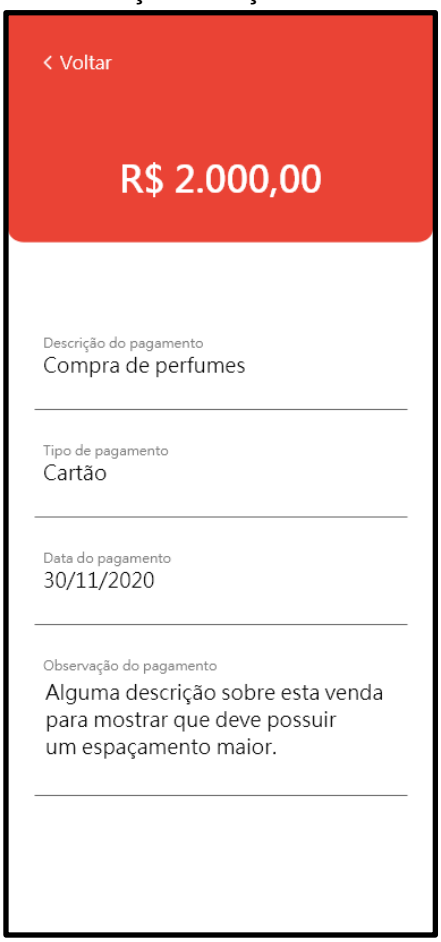
Fonte: Autor. (2020)

A figura número 13 representa a tela de novo cadastro da entrada ou saída do caixa. O campo “Valor da receita” é obrigatório e deverá ser preenchido com o valor da entrada ou saída. O campo “Adicione uma descrição” é um campo obrigatório no qual o usuário deverá preencher com uma breve descrição do pagamento. O campo “Forma de pagamento” é um campo obrigatório do tipo de múltipla escolha, contendo as opções entre crédito, débito e dinheiro. O campo tipo de receita é um campo obrigatório de múltiplas escolhas, contendo as duas opções: entrada e saída. O campo “Adicionar data” é um campo obrigatório que deve ser preenchido com a data em que foi ou será efetuado o pagamento. O campo “Adicione uma observação” não é um campo obrigatório, o usuário pode preencher com uma observação sobre o pagamento.

Existem dois botões funcionais na tela de registro de receita. O botão “Voltar” no canto superior esquerdo da tela possibilita que o usuário cancele todo o processo e volte para a tela inicial. O botão “Finalizar” no canto superior direito da tela realiza uma validação em todos os campos da tela, caso algum dos campos obrigatórios não forem preenchidos, deverá ser informado com uma mensagem de texto. Ao clicar no botão e todos os campos

estiverem preenchidos de forma correta, o sistema deverá gravar todos os dados na base de dados e redirecionar o usuário para a tela inicial.

Figura 14 - Tela de visualização e edição da entrada ou saída do caixa



Fonte: Autor. (2020)

A figura 14 representa a tela de visualização e edição dos dados de cada receita (entrada e saída do caixa). O usuário pode visualizar seus dados e editá-los caso necessário clicando em cada campo separadamente. A cor vermelha representa que o registro é um gasto. A cor verde representa uma entrada.

4. DESENVOLVIMENTO

No capítulo anterior foi possível acompanhar todas as etapas para a criação das telas do projeto, como a realização do briefing, realização do wireframe e prototipação das telas. Este capítulo tem como objetivo descrever toda a etapa de desenvolvimento, dando uma breve introdução sobre a estruturação da base de dados utilizada no projeto, os fluxogramas criados para o desenvolvimento do projeto e sobre a criação das telas e das funcionalidades do aplicativo.

4.1 Base de dados

Para que seja possível armazenar todos os dados necessários do aplicativo, foram criados dois documentos no Cloud Firebase com os nomes “money_transaction” e “cash_flux”. A tabela 1 descreve os campos e os tipos de campo da tabela money_transaction, a tabela 2 detalha a tipagem de cada campo do documento cash_flux.

Tabela 1: Descrição do documento money transaction

Campo	Tipagem de dado	Descrição
userUid	String	ID do usuário logado
value	String	Dinheiro do usuário

Fonte: Autor. (2020)

Tabela 2: Descrição do documento cash money

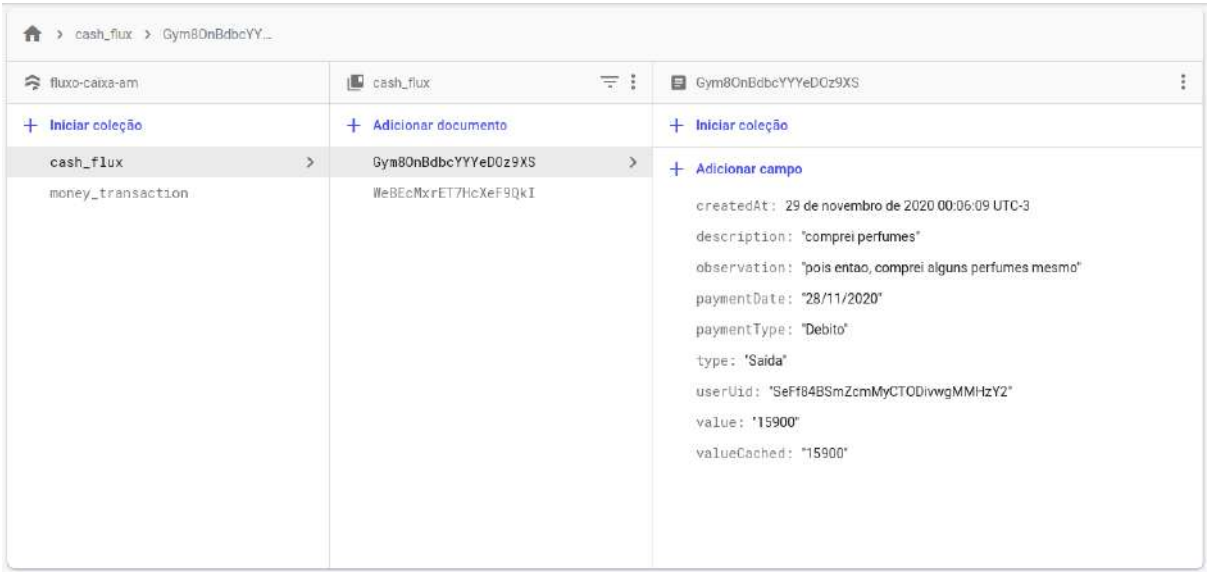
Campo	Tipagem de dado	Descrição
createdAt	Timestamp	Hora e data da criação do registro
description	String	Descrição
observation	String	Observação
paymentDate	String	Data do pagamento
paymentType	String	Tipo do pagamento (débito, crédito ou dinheiro)
type	String	Tipo do registro (entrada ou saída)
userUid	String	ID do usuário logado

value	String	Valor do registro
valueCached	String	Valor do registro para atualização do valor do documento money transaction

Fonte: Autor. (2020)

A figura número 15 representa como os dados do documento cash flux estão armazenados no Cloud Firebase.

Figura 15 - Documento cash flux no Cloud Firebase



Fonte: Autor. (2020)

4.2 Detalhamento do desenvolvimento do projeto

Esta seção tem como objetivo dar uma breve introdução aos requisitos do projeto e detalhar o desenvolvimento das telas e das regras de negócio do projeto.

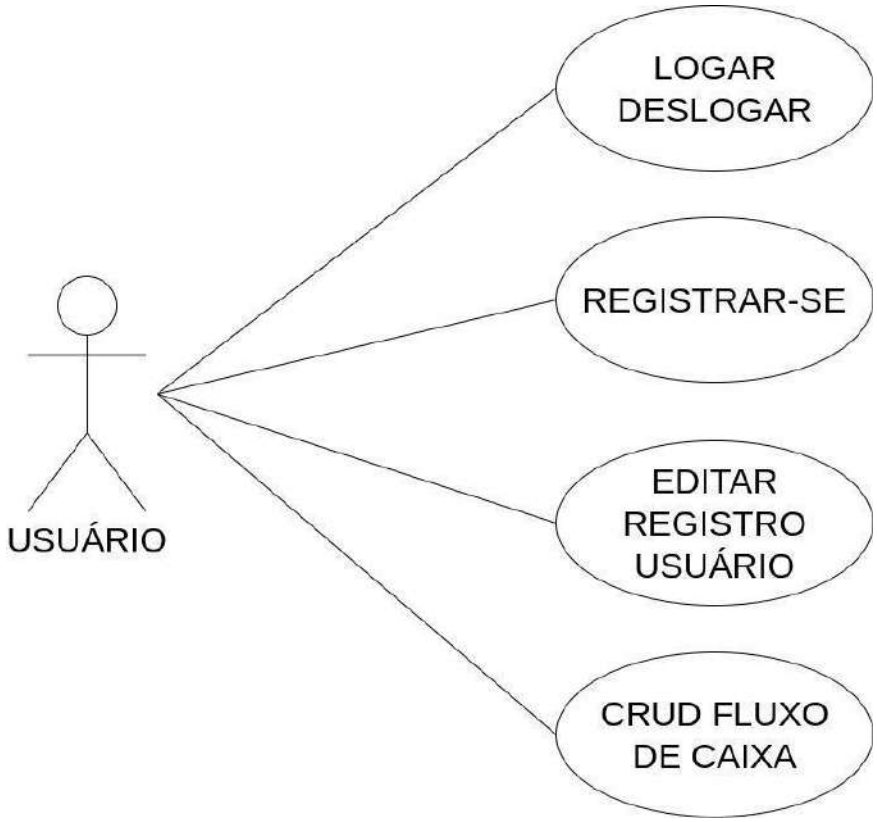
Com o objetivo de criar uma aplicação mobile para a automação do fluxo de caixa, as telas do aplicativo mobile deverão ser criadas conforme descritos na seção 3.4 Mockup contendo suas funcionalidades específicas por tela. Abaixo, listagem dos requisitos funcionais da aplicação mobile:

- RF01: A aplicação deverá privar o acesso ao aplicativo a partir de usuário e senha;
- RF02: A aplicação deverá permitir que o usuário se cadastre para acessar o aplicativo;

- RF03: A aplicação deverá permitir que cada usuário possa visualizar seus próprios dados de fluxo de caixa e somente seus dados. Restringir funcionalidade para usuários que estiverem logados.
- RF04: A aplicação deverá possibilitar que o usuário adicione novas receitas da empresa. Restringir funcionalidade para usuários que estiverem logados.
- RF05: A aplicação deverá possibilitar que o usuário atualize seus dados financeiros. Restringir funcionalidade para usuários que estiverem logados.
- RF06: A aplicação deverá permitir que o usuário remova os dados financeiros. Restringir funcionalidade para usuários que estiverem logados.

A figura 16 contém o diagrama de caso de uso demonstrando as funcionalidades possíveis que o usuário poderá realizar na ferramenta.

Figura 16 - Diagrama de caso de uso



Fonte: Autor (2020)

Para a especificação de cada funcionalidade, será separado em quatro etapas. Primeiro será desenvolvido todo o fluxo de registre-se e login, integrado com o Firebase Authentication, para que o usuário possa ter mais segurança com seus dados registrados na plataforma. A segunda etapa será criada a tela inicial contendo todos os registros de entrada e saída do caixa da empresa. Na terceira etapa do desenvolvimento do aplicativo,

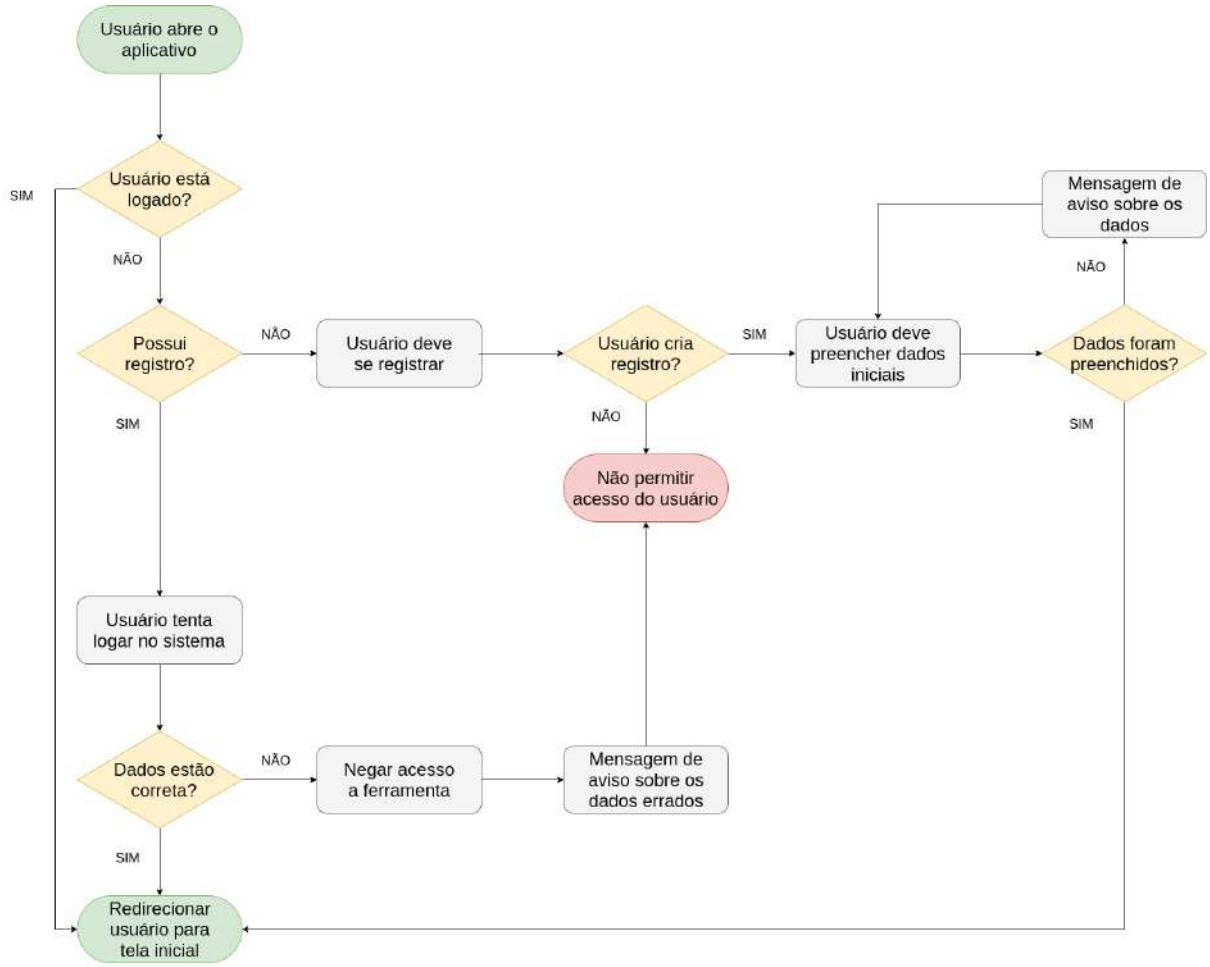
será criado a tela de registro de novas receitas de entrada e saída, descrevendo também a criação da tela de visualização, edição e exclusão da entrada ou saída do caixa. Na quarta etapa será criado a tela de início do planejamento para que o usuário possa informar seu valor inicial.

4.2.1 Fluxo de login

O fluxo de login tem como objetivo realizar a integração com o Firebase Authentication no Flutter para que seja possível implementar a autenticação no aplicativo mobile. Nesta etapa, será desenvolvido 4 telas (conforme detalhado na seção 3.4 Mockup), são elas:

- Splash Screen: Tela inicial de carregamento do aplicativo.
- Tela de acesso: Tela que deve renderizar para o usuário quando não estiver logado para que possa ou efetuar o login ou criar um novo usuário.
- Tela de login: Tela para que o usuário realize o login.
- Tela registre-se: Tela responsável na criação de novos usuários dentro da ferramenta.

Figura 17 - Fluxograma de login



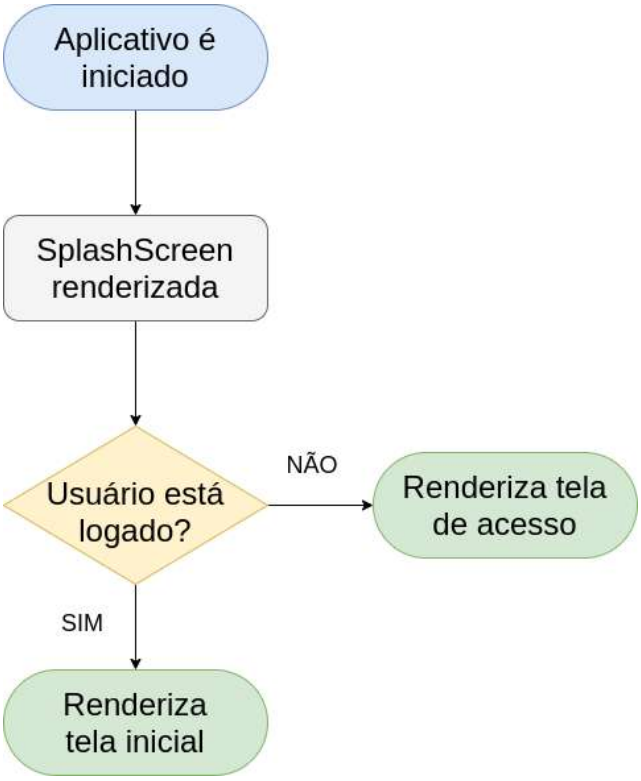
Fonte: Autor. (2020)

Conforme o fluxograma da figura 17, o usuário deverá conter uma conta para se autenticar no aplicativo, caso contrário deverá criar uma conta nova. Nas próximas seções serão especificadas cada tela desenvolvida.

4.2.1.1 Tela Splash Screen

A tela splash screen tem como objetivo mostrar uma tela amigável para o usuário enquanto o aplicativo carrega, para que não fique apenas uma tela preta. Também, deve redirecionar o usuário para outra tela dependendo do estado do usuário. A figura 18 apresenta o fluxograma da inicialização do aplicativo, detalhando o fluxo para cada estado do usuário.

Figura 18 - Fluxograma da renderização do aplicativo



Fonte: Autor. (2020)

Para a criação da widget dentro do projeto, será utilizado o Slidy CLI para facilitar a estruturação e manter a organização modular do projeto. Com o comando no terminal “slidy generate page splash -b”. O parâmetro generate (ou apenas g) possibilita a criação de um novo módulo, página, repositório entre outras opções, foi adicionado o parâmetro page para criar uma nova página. O nome da página é definido em seguida, para esta tela foi definido como splash. No final é passado o comando “-b” para definir que será do tipo bloc, não tendo a necessidade do Slidy criar um controlador para esta página. Com este comando, é criado uma pasta chamada splash e dentro da pasta é criado um arquivo contendo uma classe StatefulWidget.

Figura 19 - Widget Splash Screen

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.blue[600],
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Padding(padding: EdgeInsets.only(bottom: 60.0)),
          Icon(
            Icons.attach_money,
            color: Colors.white,
            size: 120.0,
            semanticLabel: 'Icone com o formato de cifrão'
          ), // Icon
          Padding(padding: EdgeInsets.only(bottom: 60.0)),
          CircularProgressIndicator(
            backgroundColor: Colors.white,
          ) // CircularProgressIndicator
        ], // <Widget>[]
      ), // Column
    ) // Center
  ); // Scaffold
}
```

Fonte: Autor. (2020)

A figura número 19 contém o código de toda a StatefulWidget. Nela, foi inserido o Scaffold Widget com a cor azul. Como filho dessa Widget, foi adicionado a Center Widget e após Column Widget para centralizar todos os itens no centro da tela. No Icon é feita a escolha do ícone cifrão e a Widget CircularProgressIndicator é adicionada para que o usuário consiga visualizar o progresso de carregamento da página.

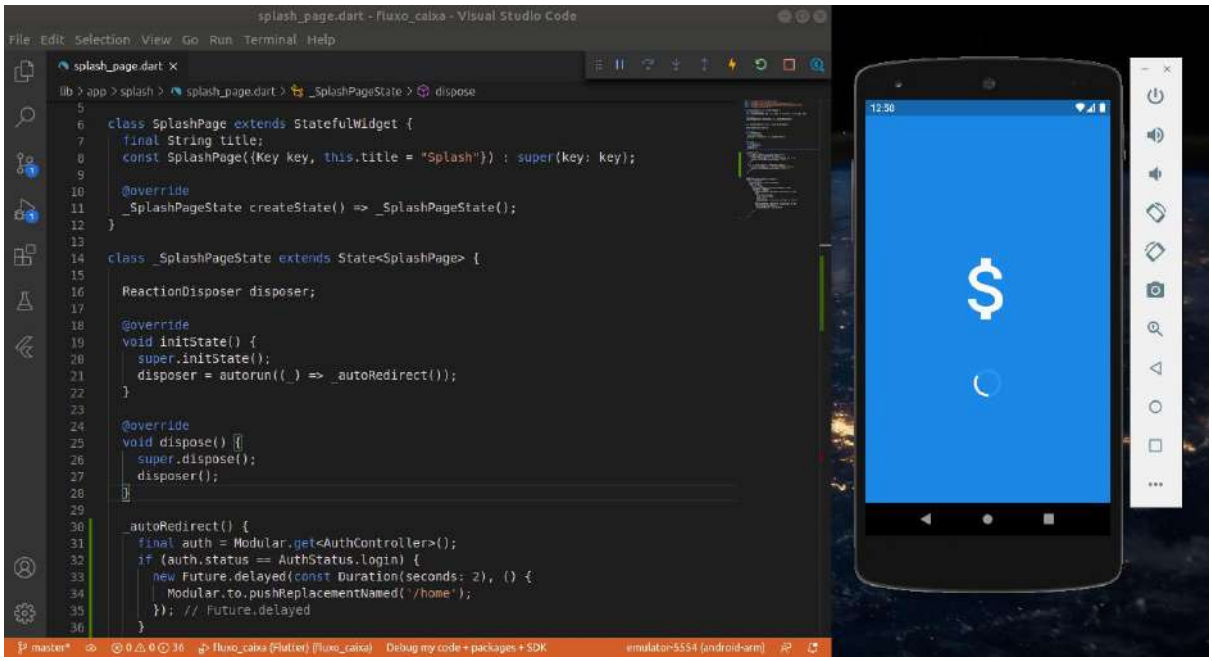
Figura 20 - Redirecionamento de tela

```
ReactionDisposer disposer;

@override
void initState() {
  super.initState();
  disposer = autorun((_) => _autoRedirect());
}

@override
void dispose() {
  super.dispose();
  disposer();
}

_autoRedirect() {
  final auth = Modular.get<AuthController>();
  if (auth.status == AuthStatus.login) {
    new Future.delayed(const Duration(seconds: 2), () {
      Modular.to.pushReplacementNamed('/home');
    }); // Future.delayed
  }
  else if (auth.status == AuthStatus.logoff) {
    new Future.delayed(const Duration(seconds: 2), () {
      Modular.to.pushReplacementNamed('/start');
    }); // Future.delayed
  }
}
```

Fonte: Autor. (2020)

Para o redirecionamento da tela foi criado o método _autoRedirect, conforme demonstrado na figura número 20. Esta função é declarada na função initState para ser executada assim que for iniciada. Como o aplicativo não contém muitas informações para fazer com que a aplicação demore para carregar, foi adicionado a função delayed com duração de 2 segundos, para que a tela splash screen seja mostrada para o usuário por um curto tempo.

Figura 21 - Classe _SplashPageState



Fonte: Autor. (2020)

A figura número 21 demonstra a tela SplashScreen sendo renderizada pelo simulador Nexus 5.

4.2.1.2 Tela de acesso

Na tela de acesso será renderizado uma tela simples contendo dois botões para que o usuário consiga acessar o aplicativo. Caso possua acesso o usuário poderá clicar no botão de login, ou se não tiver um cadastro, poderá clicar no botão de registro de usuário. Ambos botões direcionam para telas específicas.

Para criar a tela de acesso, foi utilizado o Slidy CLI para manter o projeto organizado. Pelo terminal foi executado o comando “slidy g p start -b” que gera uma nova classe StatefulWidget sem a necessidade de um controlador.

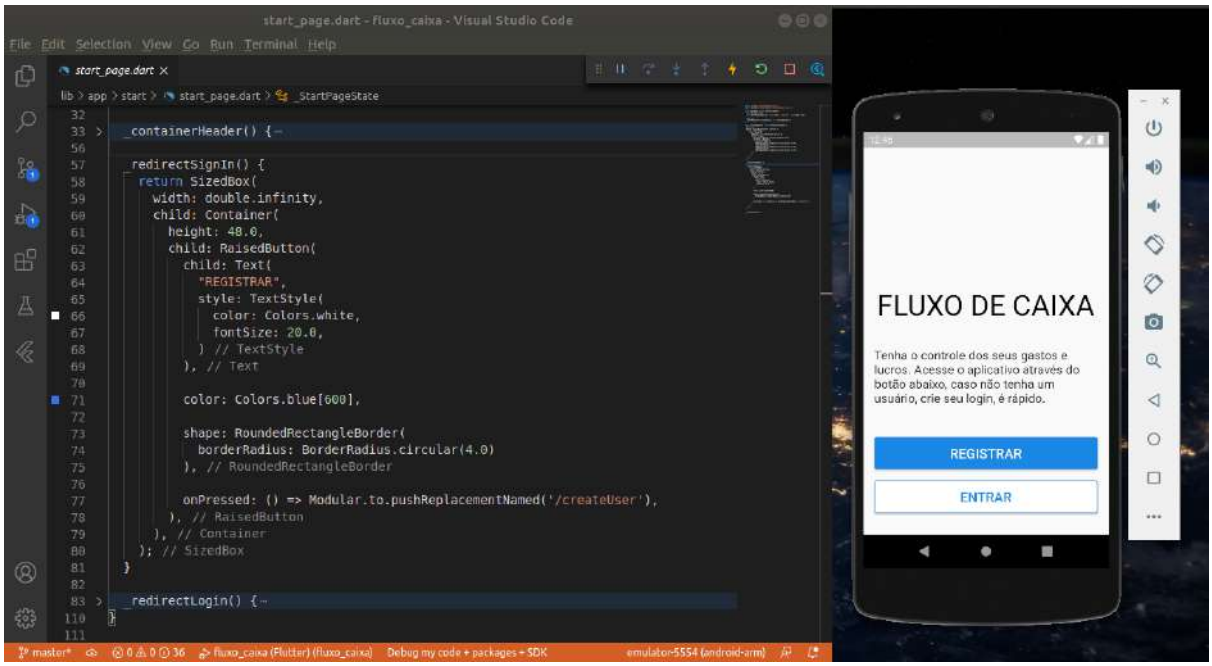
Figura 22 - Classe _StartPageState

```
class _StartPageState extends State<StartPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            _containerHeader(),
            Padding(padding: EdgeInsets.only(bottom: 46.0)),
            _redirectSignIn(),
            Padding(padding: EdgeInsets.only(bottom: 16.0)),
            _redirectLogin(),
            Padding(padding: EdgeInsets.only(bottom: 16.0)),
          ], // <Widget>[]
        ), // Column
      ), // Padding
    ); // Scaffold
  }

  _containerHeader() {…

  _redirectSignIn() {…

  _redirectLogin() {…
}
```

Fonte: Autor. (2020)

A classe _StartPageState, conforme a imagem acima, é responsável por renderizar a tela de acesso. Nesta classe foi criado três funções, a função _containerHeader é a responsável por renderizar o título e o texto da página, a função _redirectSignIn tem a responsabilidade de criar o botão de registro de usuário com a ação de redirecionar o usuário ao clicar no botão para a tela de registre-se, assim como a função _redirectLogin, com a diferença que ela renderiza o botão de login com a ação de redirecionar o usuário para a tela de login ao clicar.

Figura 23 - Tela de acesso no simulador



Fonte: Autor. (2020)

A figura acima demonstra a tela de acesso sendo renderizada em um simulador do Nexus 5. Também é possível visualizar o método _redirectSignIn descrito acima.

4.2.1.3 Tela de login

A tela de login tem como objetivo renderizar uma tela na qual possua um formulário contendo um campo de usuário e outro campo para que o usuário possa preencher a senha. Ao final, deve possuir um botão para que o usuário consiga validar seus dados e realizar o login, assim, realizando o primeiro requisito funcional.

A autenticação em um sistema se torna simples de ser realizada quando integrado com o Firebase Authentication, existindo diversas opções de integração. Para a solução deste projeto foi utilizado o método de login com e-mail e senha.

Tendo a forma de login que será utilizada pelo aplicativo, foi realizado o download da biblioteca firebase_auth para que o aplicativo possua acesso aos métodos do Firebase Authentication. Foi criado uma interface chamada IAuthRepository para que o projeto tenha uma espécie de contrato caso o login com Firebase Auth seja utilizado em outros projetos. A figura 24 demonstra a interface citada. No método getEmailPasswordLogin é o método que implementa o login com o Firebase Authentication e o método getLogout é onde deve ser desenvolvido a função de logout.

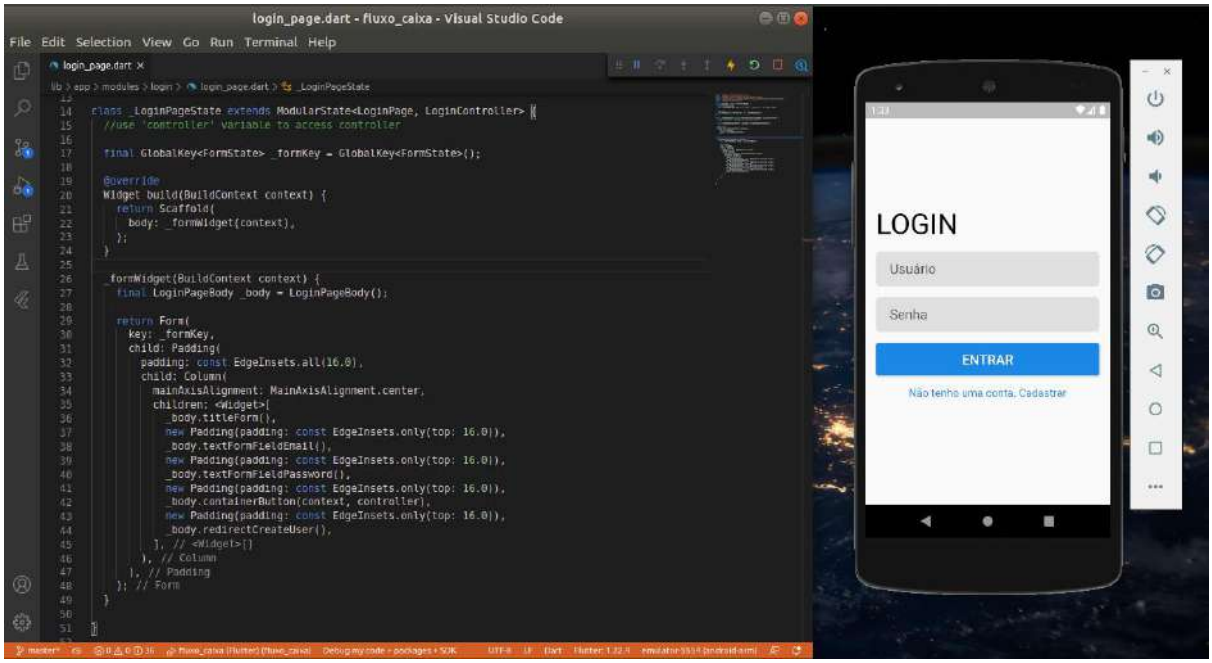
Figura 24 - Interface IAuthRepository

```dart
import 'package:firebase_auth/firebase_auth.dart';

abstract class IAuthRepository {
  Future<User> getUser();
  Future getEmailPasswordLogin(email, password);
  Future getLogout();
}
```

Fonte: Autor. (2020)

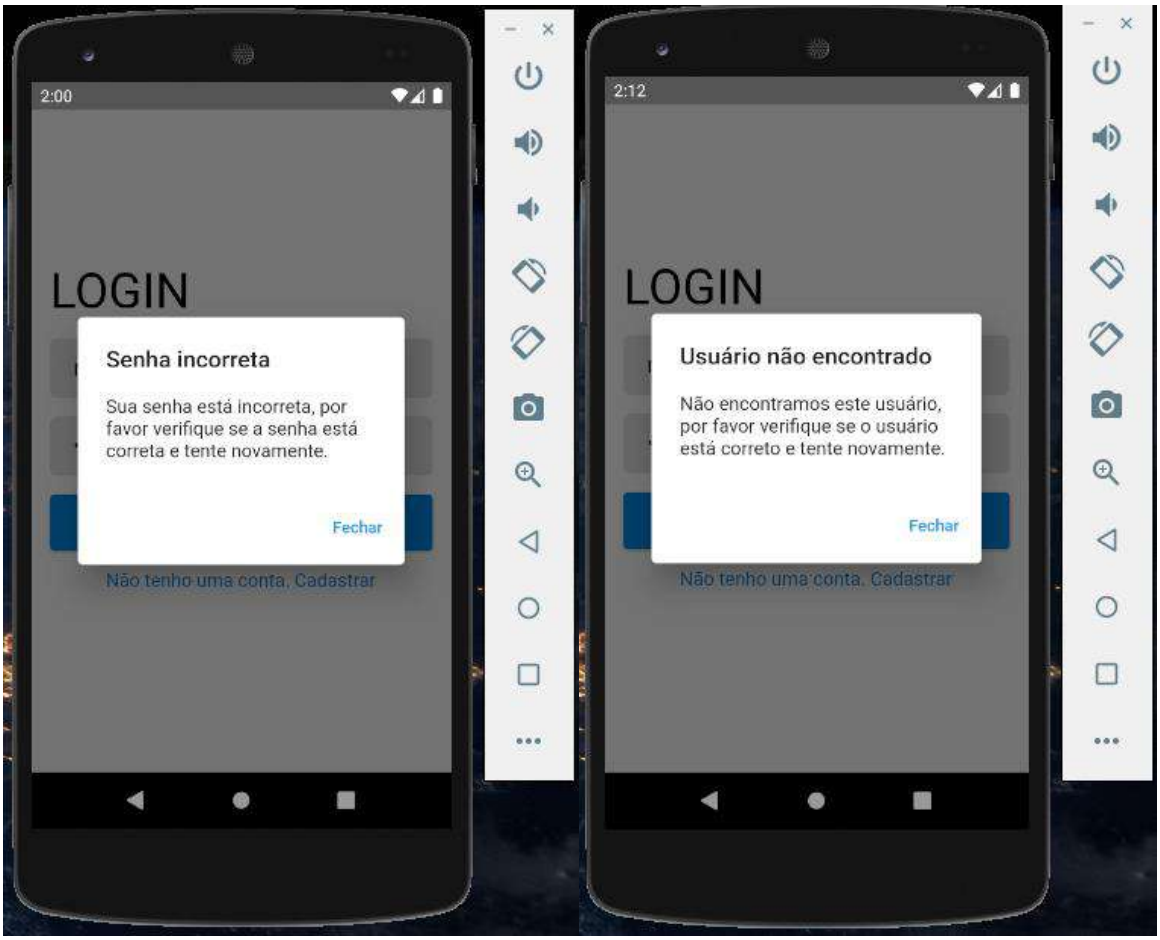
Na classe LoginPage, como na figura 25, foi criado a tela de login conforme criado na seção 3.4 Mockup.

Figura 25 - Classe LoginPage



Fonte: Autor. (2020)

Figura 26 - Erro ao efetuar o login



Fonte: Autor. (2020)

A figura 26 demonstra as mensagens de erros caso o usuário erre o usuário ou a senha.

4.2.1.3 Tela registre-se

A tela registre-se tem como objetivo permitir que o usuário consiga se registrar no aplicativo. A tela deve possuir um formulário com os campos de e-mail e senha. Ao final, deve possuir um botão para que o usuário consiga validar seus dados e criar seu registro, assim, realizando o segundo requisito funcional.

Figura 27 - Classe UserModel

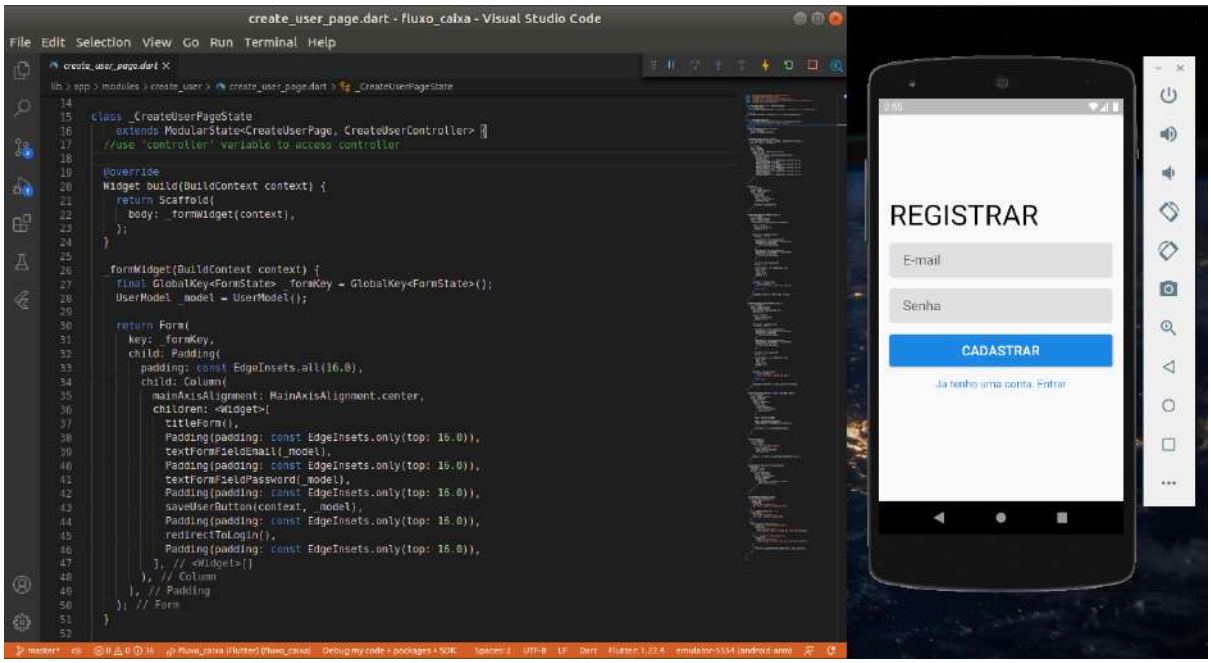
```dart
import 'package:firebase_auth/firebase_auth.dart';

class UserModel {
  String email;
  String password;

  UserModel({this.email = '', this.password = ''});

  Future<String> create() async {
    var result = 'success';
    try {
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: email,
        password: password
      );
    }
    catch(e) {
      print(e.code);
      result = e.code;
    }
    return result;
  }

}
```

Fonte: Autor. (2020)

A classe UserModel, conforme na imagem acima, implementa o método create que realiza a criação de novos usuários no Firebase Authentication, retornando o resultado do processo. Se o usuário for criado com sucesso, será retornado uma string com o conteúdo ‘success’, caso ocorra algum erro, será capturado pela exception e retornado onde o método foi declarado.

O método createUserWithEmailAndPassword realiza toda a validação necessária para a criação de usuário. Quando uma nova conta é gerada, o usuário é autenticado automaticamente. Esta função disponível na biblioteca Firebase Authentication espera um e-mail do tipo string e uma senha do tipo string. Por ser uma função assíncrona, foi feito a chamada dela usando await.

Figura 28 - CreateUserPage

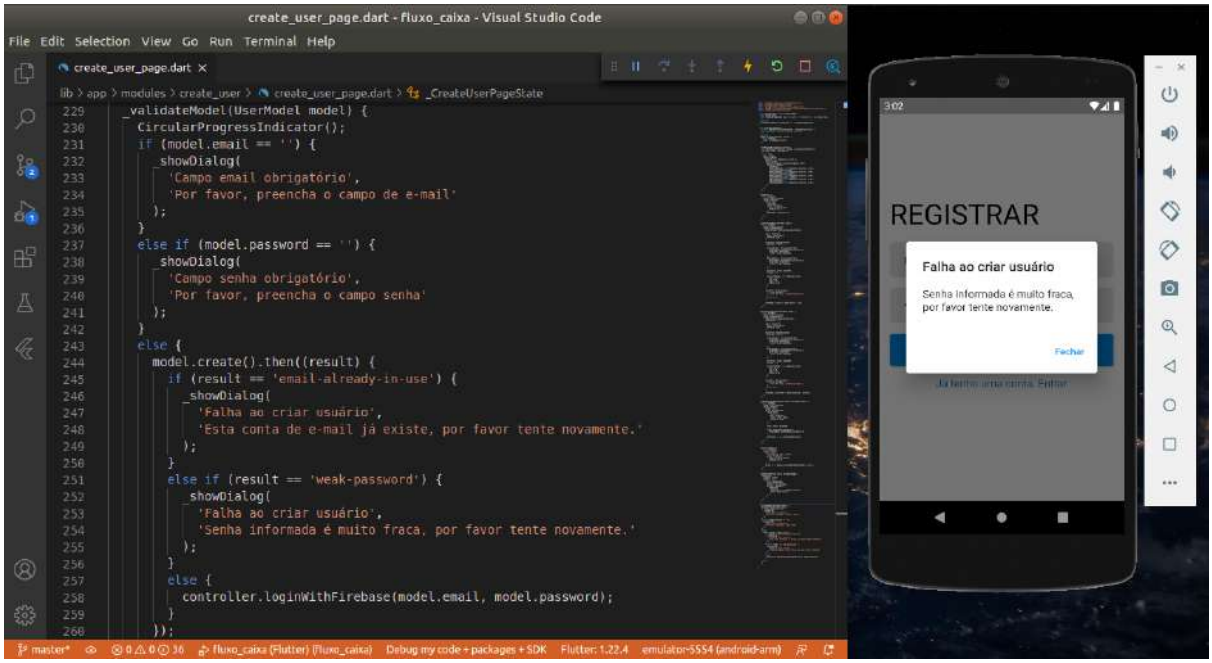


Fonte: Autor. (2020)

A classe CreateUserPage tem a responsabilidade de renderizar a tela de registro de usuário, na qual está representada na figura 28. A função build contém todas as Widgets estilizadas da tela.

A função _validateModel, apresentado na figura 29, realiza uma sequência de validações para que o usuário possua uma mensagem de aviso caso algo ocorra errado.

Figura 29 - Função _validateModel



Fonte: Autor. (2020)

4.2.2 Tela inicial

A tela inicial tem como objetivo mostrar os dados dos documentos cash_flow e money_transaction, restringindo por cada usuário e possibilitando que o usuário realize o logout. Nesta etapa, será desenvolvida a tela inicial, na qual será aberta assim que o usuário efetuar o login com sucesso.

Na primeira etapa, foi realizado o desenvolvimento do cabeçalho, no qual possui apenas o e-mail do usuário e um botão para que o usuário possa realizar o logout. Para isso, foi utilizado a Widget AppBar, nela podemos adicionar outras Widgets como textos, ícones e até mesmo botões para redirecionamento de página. Conforme na figura 30, a Widget Text foi utilizada para inserir o e-mail do usuário que está logado no sistema. Para adicionar novas Widgets dentro da AppBar na qual permaneça na mesma linha e posicionado no lado direito da tela, é necessário utilizar o argumento actions que aguarda uma lista de Widgets. Neste item, foi adicionado o IconButton para termos um botão logout com o evento de clique. Quando o usuário clicar neste botão, será realizado o logout da ferramenta.

Figura 30 - AppBar da tela inicial

```
appBar: AppBar(
  title: Text(Modular.get<AuthController>().user.email),
  actions: [
    IconButton(
      icon: Icon(Icons.logout),
      onPressed: controller.logoff,
    ), // IconButton
  ],
  elevation: 0,
  backgroundColor: Colors.blue[600],
), // AppBar
```

Fonte: Autor. (2020)

A segunda etapa foi contemplada com a criação do corpo da tela. Para que seja possível buscar todos os dados de cada documento que está no Firebase, foi necessário criar dois métodos de consulta em uma interface para que seja possível trabalhar de forma isolado.

Figura 31 - Interface IFirestoreRepository

```
abstract class IFirestoreRepository {
  Stream<List<CashFlowModel>> getCashFlow(AuthController auth);
  Stream<List<MoneyTransactionModel>> getMoney(AuthController auth);
}
```

Fonte: Autor. (2020)

Os métodos getCashFlow e getMoney da interface IFirestoreRepository, representados na imagem 31, tem como responsabilidade retornar uma lista de objetos contendo todos os dados de seus respectivos documentos. Ambos os métodos aguardam um parâmetro do tipo AuthControlller. Na imagem 32 temos a implementação da interface IFirestoreRepository através da classe FirestoreRepository. Esta classe sobrescreve os métodos da interface, fazendo a consulta no Cloud Firebase. Ambas as consultas fazem a pesquisa em seus respectivos documentos, no qual realiza o filtro a partir do userId igual ao do usuário logado. Também, é feita uma ordenação pela data de pagamento.

Figura 32 - Classe FirestoreRepository

```
class FirestoreRepository implements IFirestoreRepository {

  final FirebaseFirestore firestore;

  FirestoreRepository(this.firestore);

  @override
  Stream<List<CashFlowModel>> getCashFlow(AuthController auth) {
    return firestore.collection('cash_flux')
      .where('userUid', isEqualTo: auth.user.uid)
      .orderBy('paymentDate', descending: true)
      .snapshots().map((query) {
      return query.docs.map((doc) {
        return CashFlowModel.fromDocument(doc);
      }).toList();
    });
  }

  @override
  Stream<List<MoneyTransactionModel>> getMoney(AuthController auth) {
    return firestore.collection('money_transaction')
      .where('userUid', isEqualTo: auth.user.uid)
      .snapshots().map((query) {
      return query.docs.map((doc) {
        return MoneyTransactionModel.fromDocument(doc);
      }).toList();
    });
  }
}
```

Fonte: Autor. (2020)

Quando uma consulta no Firebase é composta com muitas clausulas, é preciso criar indices para que possamos ter uma melhor performance na hora de realizar as consultas no Cloud Firebase. A figura 33 demonstra um índice criado para o documento cash_flux.

Figura 33 - Índice do documento cash_flux

Código da coleção	Campos indexados	Escopo da consulta	Status
cash_flux	userUid Crescente paymentDate Decrescente	Coleta	Ativado

Fonte: Autor. (2020)

Figura 34 - Classe HomeController

```
abstract class _HomeControllerBase with Store {
  final IFirestoreRepository firestoreRepository;

  @observable
  ObservableStream<List<CashFlowModel>> cashFlowList;

  @observable
  ObservableStream<List<MoneyTransactionModel>> moneyTransactionList;

  _HomeControllerBase(IFirestoreRepository this.firestoreRepository) {
    getList();
  }

  @action
  getList() {
    cashFlowList = firestoreRepository.getCashFlow(Modular.get<AuthController>()).asObservable();
    moneyTransactionList = firestoreRepository.getMoney(Modular.get<AuthController>()).asObservable();
  }
```

Fonte: Autor. (2020)

No nosso controlador da página inicial, foram criados dois observadores no qual o MobX acaba fazendo toda gerência de estado. O método getList define o resultado da consulta criada na figura 34 para os observadores, para que assim, o MobX consiga atualizar de forma reativa os valores entre o aplicação e o Firebase.

Figura 35 - Body da tela inicial

```
body: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  mainAxisSize: MainAxisSize.max,
  children: <Widget>[
    _header(),
    Padding(padding: EdgeInsets.only(bottom: 32.0)),
    Padding(
      padding: EdgeInsets.all(8.0),
      child: Align(
        alignment: Alignment.centerLeft,
        child: Text(
          'Últimas transações',
          style: TextStyle(
            fontSize: 12.0,
            color: Colors.black,
          ), // TextStyle
          textAlign: TextAlign.left,
        ), // Text
      ), // Align
    ), // Padding
    _sectionObserver(),
  ], // <Widget>[]
), // Column
```

Fonte: Autor. (2020)

No final, para mostrar os dados que agora estão sendo consultados pela aplicação, foram criadas duas funções para manter a nossa classe mais organizada. Na função _header temos a estilização dos dados consultados no documento money_transaction e na função _sectionObserver contém a listagem de todos os dados consultados no documento cash_flow, conforme podemos visualizar na figura 35.

4.2.3 CRUD cash flow

Esta seção tem como objetivo descrever toda a etapa de desenvolvimento realizada para a criação dos métodos de inserção, atualização, visualização e remoção dos dados da coleção cash flow criada no Cloud Firebase.

4.2.3.1 Criando novo registro

A utilização da biblioteca cloud_firestore facilitou muito o desenvolvimento nesta etapa. Para realizar a inserção de novos documentos, foi necessário criar uma instância nova, informando o nome da coleção e declarando o método add, informando todos os campos da coleção, conforme demonstrado na figura 36.

Figura 36 - Criando novo registro cash flow

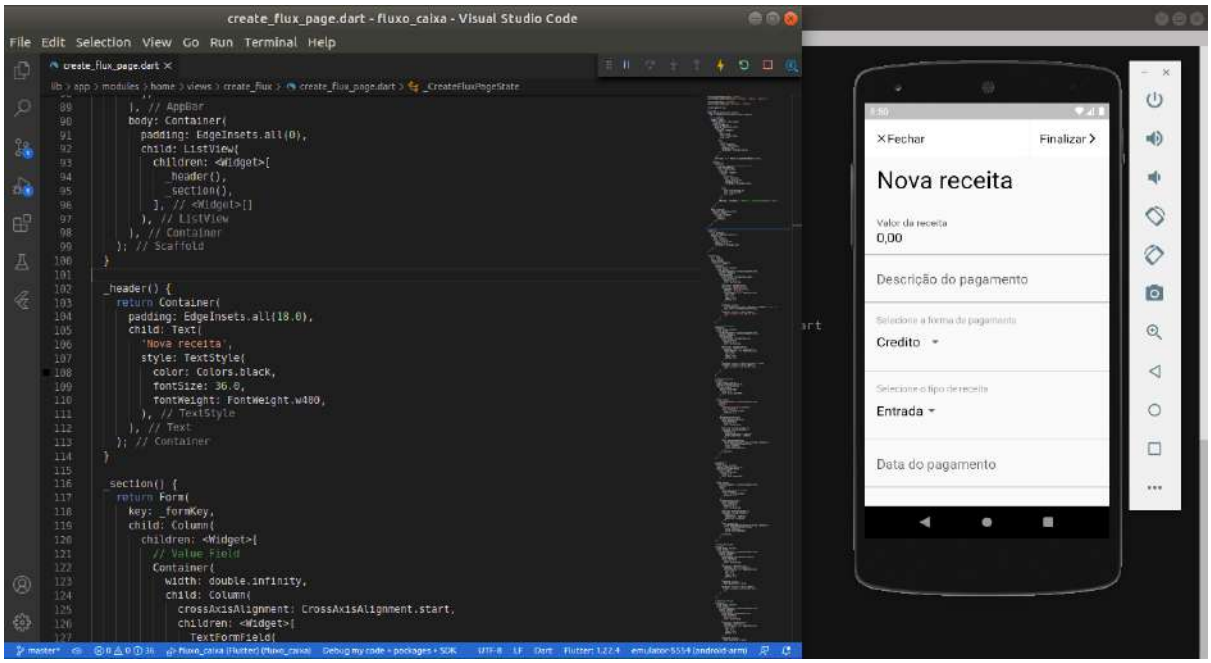
```
reference = await FirebaseFirestore.instance
  .collection('cash_flux')
  .add({
    'description': description,
    'observation': observation,
    'value': value,
    'valueCached': value,
    'type': type,
    'paymentType': paymentType,
    'createdAt': createdAt,
    'paymentDate': paymentDate,
    'userUid': userUid,
  });
```

Fonte: Autor. (2020)

Como podemos visualizar na imagem 37, foram criados os métodos _header e _section que possuem os layouts da tela de criação. Ao lado, temos o simulador que está renderizando a tela. Cada campo deve ser preenchido de forma específica, sendo obrigatórios ou não. Abaixo, listagem dos campos e descrição de cada:

- Valor da receita: Campo obrigatório do tipo texto, deve ser preenchido o valor de quanto o usuário gastou ou recebeu. Para formatação do texto para moeda, foi utilizado a biblioteca flutter_masked_text pois o Flutter não possui esta conversão de forma nativa.
- Descrição do pagamento: Campo obrigatório do tipo texto. Usuário deve preencher uma breve descrição sobre a entrada ou saída do dinheiro.
- Selecionar a forma de pagamento: Campo obrigatório de múltipla escolha, onde o usuário pode escolher entre crédito, débito ou dinheiro.
- Selecione o tipo de receita: Campo obrigatório de múltipla escolha, no qual possibilita que o usuário escolha entre entrada ou saída.
- Data do pagamento: Campo obrigatório do tipo datetime. Deve ser preenchido com a data em que foi realizada a entrada ou saída do dinheiro.
- Observação: Campo não obrigatório do tipo texto, no qual o usuário pode descrever algumas observações sobre a sua transação.

Figura 37 - Criando novo registro cash flow

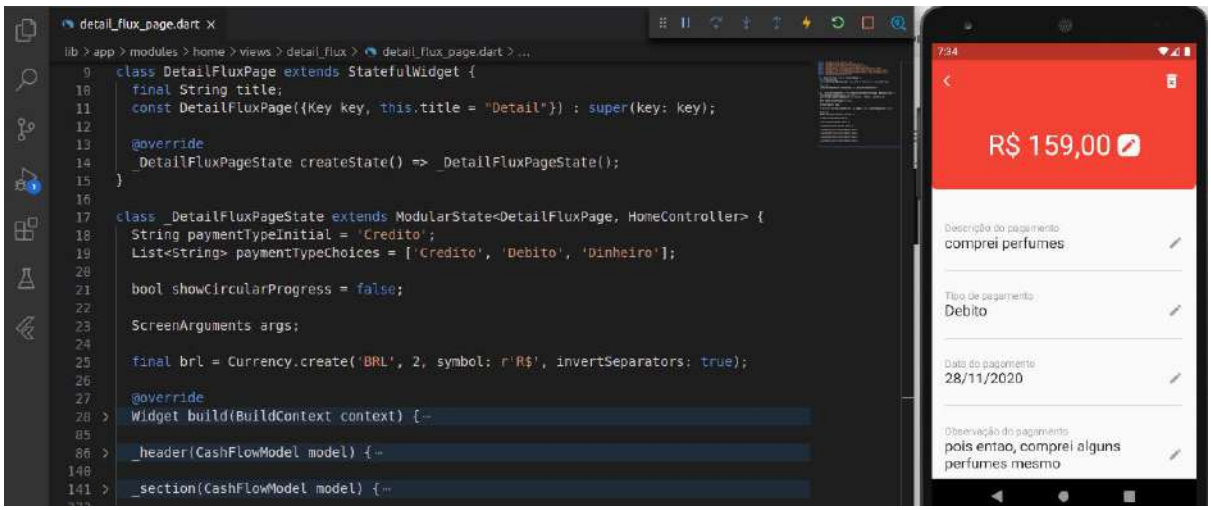


Fonte: Autor. (2020)

4.2.3.2 Visualizando registro

Como podemos visualizar na seção 4.2.2, a tela inicial realiza a listagem de todos os registros. Nesta etapa, será detalhado a visualização específica de cada item para que o usuário consiga visualizar todos os dados pertencentes a um registro.

Figura 38 - Visualização de registro



Fonte: Autor. (2020)

A classe DetailFluxPage renderiza a tela de visualização do registro específico, mostrando todos os dados do documento, como podemos visualizar na figura 38.

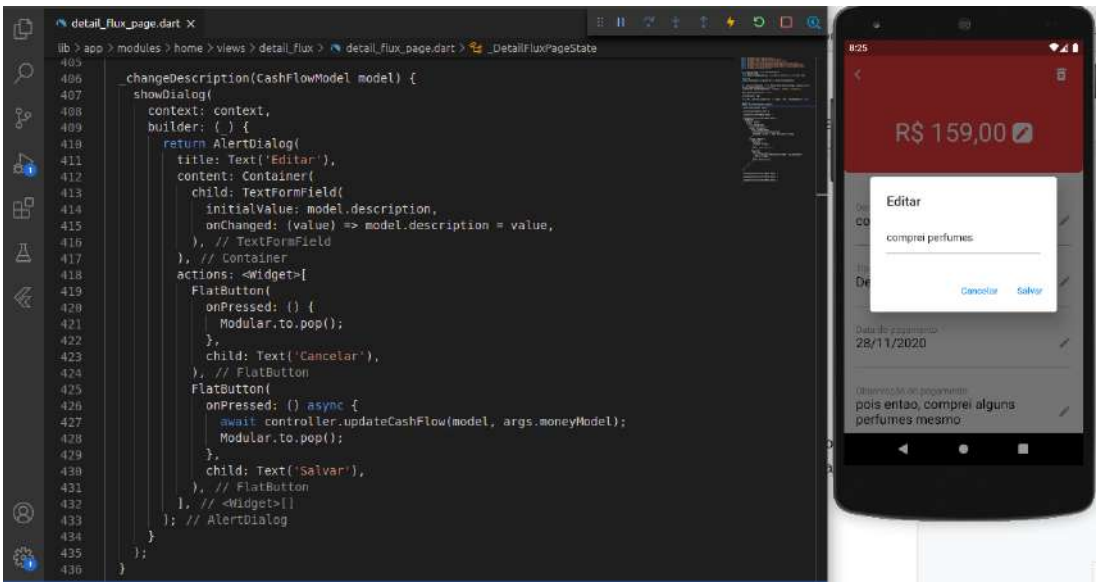
Quando o campo type tiver o valor igual a entrada, o cabeçalho da tela será da cor verde, já quando o mesmo campo for igual a saída, terá a cor vermelha. No qual a cor verde representa o recebimento de um valor, enquanto a cor vermelha simboliza o gasto.

4.2.3.3 Editar registro

Na mesma tela onde o usuário pode visualizar o dado de um registro específico, pode também editar os valores de todos os campos apresentados na tela.

Foi utilizado o campo que contém a descrição do registro como exemplo do funcionamento da atualização do dado. Na figura 39 é possível visualizar o método _changeDescription responsável por renderizar a Widget AlertDialog contendo o texto atual e possibilitando que o usuário troque o valor. Para ativar este modo de alteração, basta o usuário pressionar no ícone que fica ao lado direto de cada campo.

Figura 39 - Atualização do campo description



Fonte: Autor. (2020)

4.2.3.4 Excluir registro

A tela de visualização permite também que o usuário exclua o registro clicando no ícone com um formato de lixeira posicionado na parte superior direita do cabeçalho. Ao ser pressionado, a função deleteCashFlow (conforme a figura 40) é chamada e realiza uma atualização na coleção money transaction antes de deletar o registro. Quando o documento que está sendo removido é do tipo entrada, é removido o valor do campo value do saldo atual do usuário. Caso seja do tipo saída, então este valor é devolvido. Ao final, é chamado o método delete da model CashFlowModel no qual só chama o método delete da referência que está sendo manipulada.

Figura 40 - Método deleteCashFlow

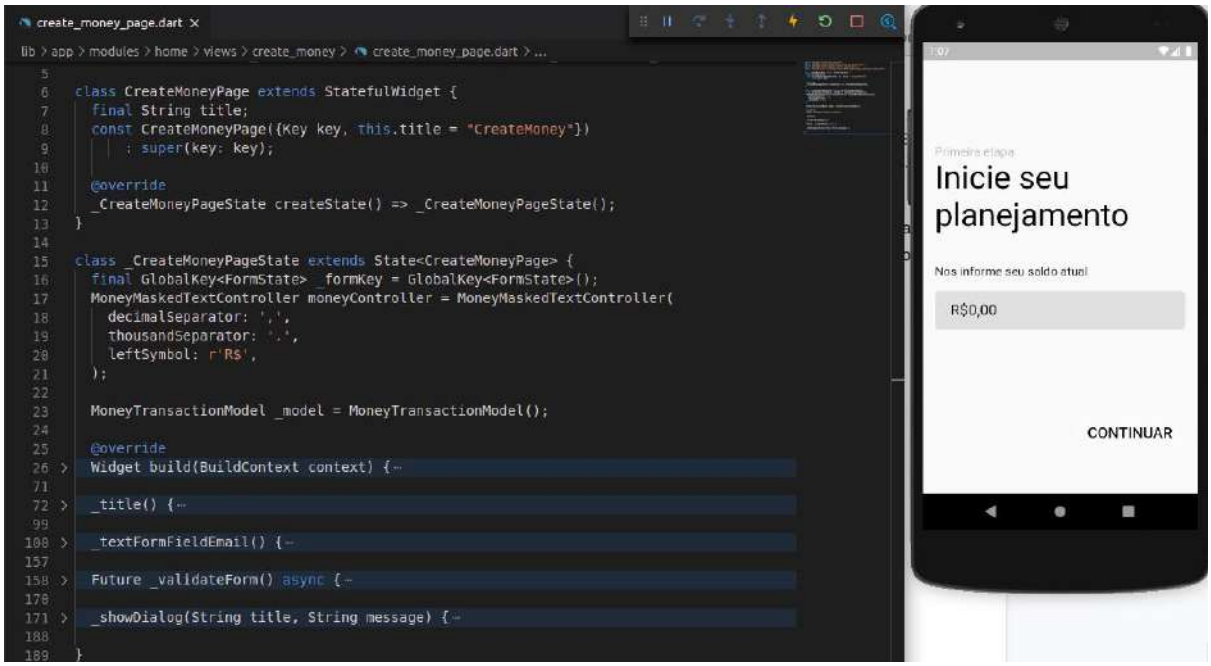
```
deleteCashFlow(CashFlowModel modelCashFlow, MoneyTransactionModel modelMoney) async {
  int money = 0;
  int account = int.parse(modelMoney.value);
  int price = int.parse(modelCashFlow.value);

  money = modelCashFlow.type.toLowerCase() == 'entrada' ?
      account - price : account + price;

  modelMoney.value = money.toString();
  modelMoney.save();

  modelCashFlow.delete();
}
```

Fonte: Autor. (2020)

4.2.4 Tela de início do planejamento

A primeira vez que o usuário acessar o aplicativo, deverá informar ao sistema quanto possui de capital para que a ferramenta consiga controlar e atualizar quanto o usuário está gastando e recebendo.

Figura 41 - Classe _CreateMoneyPageState



Fonte: Autor. (2020)

A classe CreateMoneyPageState renderiza a tela na qual é aberta assim que o usuário se cadastra no aplicativo. Conforme podemos visualizar na figura 41, é solicitado ao usuário o preenchimento do input com o saldo que o usuário gostaria de iniciar sua gestão financeira.

Assim que o botão continuar é pressionado, a função _validateForm realiza uma validação no input verificando se o campo está vazio. Caso esteja, informa para o usuário que o campo é obrigatório e deve ser preenchido, como demonstrado na figura 42. Caso contrário, o dado será salvo no Cloud Firebase e redirecionado para a tela inicial, podendo cadastrar seus gastos semanais.

Figura 42 - Mensagem de erro da tela inicio de planejamento

Fonte: Autor. (2020)

6. CONCLUSÃO

A automatização do processo de controle de fluxo de caixa irá auxiliar muito no dia a dia da microempresa, pois os funcionários poderão começar a realizar a gerência financeira pelo aplicativo de forma simples e prática. A entrega deste MVP nos dará a possibilidade de adicionar novas funcionalidades que poderão ser incluídas futuramente, dando ainda mais utilidade ao aplicativo e automatizando mais processos manuais da empresa.

O primeiro contato com o framework Flutter foi muito satisfatório, a utilização do hot reload do Flutter é muito interessante pois facilita muito na hora de visualizar as modificações do código sem precisar reiniciar toda a aplicação. Fazendo com que o desenvolvimento seja mais produtivo e se torne mais rápido.

A utilização da arquitetura modular vai facilitar para que seja possível escalar o aplicativo mobile, pois só será preciso criar um novo módulo e desenvolver toda a regra de negócio. Trabalhando em um escopo independente sem precisar editar pastas ou arquivos que envolvem outras funcionalidades.

6.2 Trabalhos Futuros

Em busca de otimizar cada vez mais os processos realizados pela microempresa, será realizado o desenvolvimento de novas funcionalidades neste mesmo aplicativo mobile para auxiliar cada vez mais a microempresa. Primeiro, será criado o gerenciamento de estoque, ajudando no controle dos produtos que possuem em loja. Após a integração desta funcionalidade, buscaremos a criação do cadastramento dos clientes para facilitar a visualização dos dados de cada cliente da microempresa.

7. REFERÊNCIA BIBLIOGRÁFICA

BECAPTAIN. **Estruturando um projeto no Flutter com Modular**. 2020. Disponível em: <https://becaptain.com.br/2020/05/23/flutter-modular/>. Acessado dia 01 Dez. 2020.

CARVALHO, Thais. **Análise e Desenvolvimento de um Sistema Offline para Auxílio Gerencial da Pecuária de Corte e Fluxo de Caixa Simples para Pequenas Propriedades Rurais.** 2019. Disponível em <https://repositorio.ifgoiano.edu.br/handle/prefix/992>. Acessado dia 03 Jul. 2020.

CLEMENT. **Number of available applications in the Google Play Store from December 2009 to June 2020.** 2020. Disponível em <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. Acessado em 26 Mai. 2020.

FIREBASE. **Cloud Firestore**. 2020. Disponível em <https://firebase.google.com/docs/firestore/>. Acessado dia 03 Dez. 2020.

FIREBASE. **Firebase Authentication**. 2020. Disponível em <https://firebase.google.com/docs/auth>. Acessado dia 03 Dez. 2020.

FLUTTER. **Documentação Flutter.** 2020. Disponível em: <https://flutter.dev/docs/resources/faq>. Acessado dia 29 Nov. 2020.

FLUTTER. **Introduction to widgets.** 2020. Disponível em: <https://flutter.dev/docs/development/ui/widgets-intro>. Acessado dia 30 Nov. 2020.

FLUTTERANDO. **Flutter Modular**. 2020. Disponível em: <https://pub.dev/packages/flutter_modular>. Acessado dia 01 Dez. 2020.

FOWLER, Martin. **Who Needs an Architect?** 2003. Disponível em <https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>. Acessado dia 04 Jul. 2020.

IGTV. **Arquitetura de software: sua definição e aplicação nos negócios.** 2019. Disponível em <https://www.igti.com.br/blog/arquitetura-de-software-definicao-negocios/>. Acessado dia 04 Jul. 2020.

LIMA, Marina. **Análise do Processo de Migração de Sistemas de Arquitetura Monolítica para Microsserviço.** 2019. Disponível em: <https://www.cin.ufpe.br/~tg/2019-1/TG_EC/tg_mml3.pdf>. Acessado dia 12 Out. 2020.

MARIA DE PAULA, Bianca. **O CONTROLE DE FLUXO DE CAIXA PARA AS PEQUENAS E MICROEMPRESAS.** 2018. Disponível em <http://repositorio.aee.edu.br/bitstream/aee/8206/1/TCC%20-%20Bianca.pdf>. Acessado dia 03 Jul. 2020.

MOBX. **Mobx Documentação**. 2020. Disponível em <https://pub.dev/packages/mobx>. Acessado dia 02 Dez. 2020.

MÜLLER, Julia. **Mais de 70% de pequenas e microempresas sentem os impactos da pandemia.** 2020. Disponível em <https://www.diariopopular.com.br/economia/mais-de-70-de-pequenas-e-microempresas-sentem-os-impactos-da-pandemia-151245/>. Acessado dia 26 Jun. 2020.

NHIMI, Filipe. **Princípios e Práticas em Arquitetura de Software.** 2016. <http://www.machado.mg.gov.br/files/concursos/1cf11cf161fe4eb688dfeec880d6b4d9.pdf>. Acessado dia 04 Jul. 2020.

PINHEIRO, Allan Petterson da Silva. **UX design introduzido no desenvolvimento de interfaces gráficas**. 2016. Disponível em <https://repositorio.uniceub.br/jspui/handle/235/9445>. Acessado dia 14 de out 2020.

RICHARDSON, Chris. **Pattern: Monolithic Architecture.** 2020. Disponível em: <https://microservices.io/patterns/monolithic.html>. Acessado dia 12 Out. 2020.

ROSA, Robson. **TECNOLOGIAS PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIPLATAFORMA: Um estudo sobre os frameworks React Native e Flutter.** 2019. Disponível em <http://raam.alcidesmaya.com.br/index.php/projetos/article/view/54/54>. Acessado dia 04 Jul. 2020.

SEBRAE. **O que é o fluxo de caixa e como aplicá-lo no seu negócio.** 2019. Disponível em <https://www.sebrae.com.br/sites/PortalSebrae/artigos/fluxo-de-caixa-o-que-e-e-

como-implantar,b29e438af1c92410VgnVCM100000b272010aRCRD>. Acessado dia 03 Jul. 2020.

SERASA. **Ir à falência: veja os principais motivos e como evitá-los.** 2020. <https://empresas.serasaexperian.com.br/blog/ir-a-falencia-veja-os-principais-motivos-e-como-evita-los/>. Acessado dia 24 Ago. 2020.

SILVA, João Vitor Chaves. 2016. **Como facilitar o relacionamento de Devs e Designers com o Invision**. Disponível em <https://medium.empreendajunto.com/como-facilitar-o-relacionamento-de-devs-e-designers-com-o-invision-9c1866eaaf33>. Acessado dia 05 Dez. 2020.

SOSINSKI, Patrick. **HOTFIXES TO THE STABLE CHANNEL.** 2020. Disponível em: <https://github.com/flutter/flutter/wiki/Hotfixes-to-the-Stable-Channel>. Acessado dia 29 Nov. 2020.

SOUZA, Ivan. **Entenda o que é briefing e como elaborar um do zero!** 2020. Disponível em <https://rockcontent.com/br/blog/briefing/>. Acessado dia 05 Set. 2020.

VASQUE, Michel. SGOTI, Rogerio. **Desenvolvimento de Software para Controle de Fluxo de Caixa para Pequena Empresa.** 2019. Disponível em <http://www.jornacitec.fatecbt.edu.br/index.php/VIIIJTC/VIIIJTC/paper/viewFile/1739/2588>. Acessado dia 03 Jul. 2020.

ZENZELUK, Jean. PASTERNAK, Wilian. BINI, Elena. **Tecnologia Java no desenvolvimento de Sistema de Vendas para Controle de Caixa e Estoque.** 2015. Disponível em <https://semanaacademica.org.br/system/files/artigos/tecnologia_java_no_desenvolvimento_de_sistema_de_vendas_para_controle_de_caixa_e_estoque_0.pdf>. Acessado dia 03 Jul. 2020.