UNIVERSIDADE FEDERAL DO PARANÁ

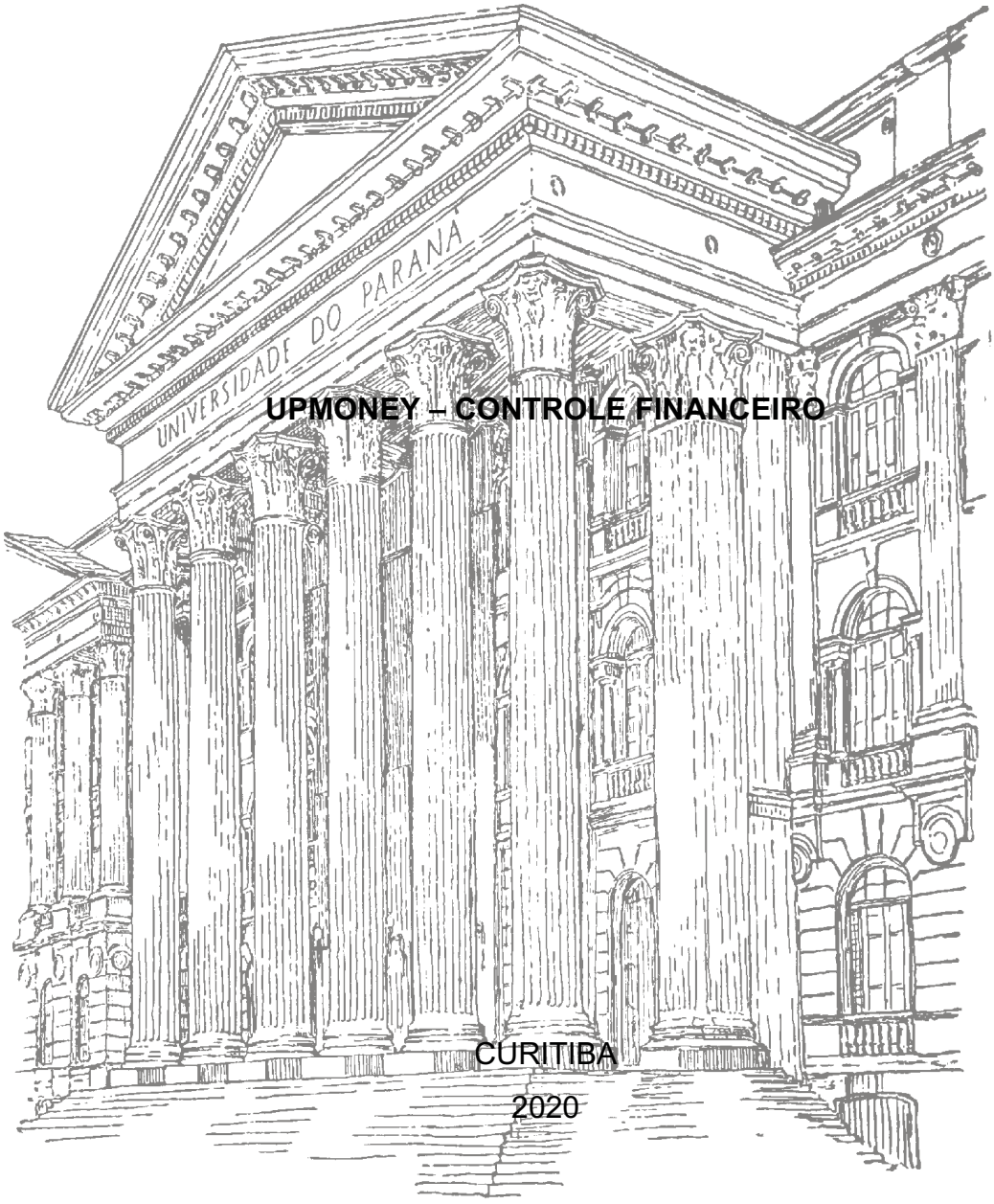
AMANDA DOS SANTOS ZANON

BRUNA JENIFER FONSECA

CLEVERSON DIEGO VIEIRA

FERNANDO AUGUSTO PERIN JUNIOR

LÍGIA KAORI IHA NAKAZATO



UPMONEY – CONTROLE FINANCEIRO

CURITIBA

2020

AMANDA ZANON DOS SANTOS
BRUNA JENIFER FONSECA
CLEVERSON DIEGO VIEIRA
FERNANDO AUGUSTO PERIN JUNIOR
LÍGIA KAORI IHA NAKAZATO

UPMONEY – CONTROLE FINANCEIRO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo, Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná

Professora Doutora Rafaela Mantovani Fontana

Curitiba
2020

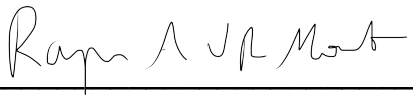
TERMO DE APROVAÇÃO

Amanda Zanon
Bruna Jenifer Fonseca
Cleverson Diego
Fernando Júnior Perin
Lígia Nakazato

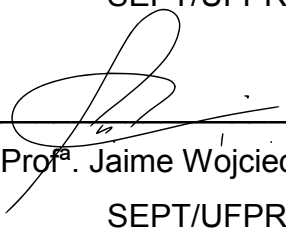
Upmoney

Monografia aprovada como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Setor de Educação Profissional e Tecnológica da Universidade Federal do Paraná.

Profa. Rafaela Mantovani Fontana
Orientadora – SEPT/UFPR



Prof. Razer A. N. R. Montaño
SEPT/UFPR



Profª. Jaime Wojciechowski
SEPT/UFPR

Curitiba, 03 de Setembro de 2020.

RESUMO

Por meio de uma pesquisa realizada pela ANBIMA, constatou-se que a maioria da população brasileira não possui a cultura de guardar dinheiro, alguns por má administração de seus recursos, outros por não acharem necessário ou até mesmo por desconhecerem o mercado financeiro. No entanto, também, observou-se que muitas das pessoas que não guardam dinheiro, se incentivadas, passariam a poupar, pois, muitas vezes, não possuem um efetivo controle sobre seus gastos. Neste contexto, o aplicativo UpMoney visa facilitar o controle financeiro de seus usuários, por meio de um método organizado de registro de despesas e receitas, o qual o usuário poderá registrar suas finanças de forma imediata, com registros manuais. Além de possibilitar uma visão analítica do momento e, também, histórica da vida financeira do usuário por meio de gráficos e filtros. Além disso, esse aplicativo viabiliza a criação de planejamentos a curto e longo prazo baseado na metodologia 50/35/15, proporcionando ao usuário meios para atingir metas, por meio de um planejamento mais factível e personalizado. Desta forma, o desenvolvimento deste projeto foi realizado utilizando-se de um processo de modelagem que aplicou a UML (*Unified Modeling Language*), por meio dos diagramas de caso de usos, classes e sequência. Além desses, utilizou-se da modelagem de dados para conceber o banco de dados. Assim, prosseguiu-se com a programação de um aplicativo *Android* utilizando o framework *React Native*, o qual permite o controle financeiro por meio do cadastro de despesas e receitas aplicando a metodologia 50/35/15 para tal. Os dados deste aplicativo são armazenados no sistema de gerenciamento de banco de dado *MySQL* e o acesso deste é realizado através de um *webservice* desenvolvido na linguagem orientada a objetos Java. Sendo o projeto realizado de forma incremental com o auxílio do Scrum e Kanban, permitindo assim, o desenvolvimento deste trabalho sobre os conceitos e práticas de metodologias ágeis.

Palavras Chaves: Controle Financeiro. Projeto. Aplicativo. Modelagem. *Webservice*.

ABSTRACT

Through a survey conducted by ANBIMA, we've found that most of the Brazilian population has no culture about saving money, some for bad managing their resources, others for not being allowed or even unaware of the financial market. However, people who do not save money encourage themselves to spend money because they often do not know where they spent it. In this context, the UpMoney app aims to facilitate the financial control of their users through an organized method of recording expenses and revenues. The users will be able to register their finances immediately, with manual records. In addition to providing both current and historical analytical view of the user's financial life through graphs and charts. The app also enables the creation of short and long term planning based on 50/35/15 methodology, which provides the user ways to achieve goals, through a more feasible and personalized plan. Therefore this project's development was carried out through a modeling process who used the Unified Modeling Language (UML) standards, including the use case diagram, the classes diagram and the sequence diagrams. Therefore, and Android application was built using the React native framework, which allows financial control through the registration of expenses and revenues applying the 50/35/15 methodology for this. The application data is stored in a MySQL database management system and is accesses through a web service developed in Java object-oriented language. In addition to these, data modeling was applied to obtain the database. The project was carried out incrementally, practices such as Kanban and Scrum were used, which provided the application concepts about agile methodology during the entire project.

Keywords: Financial Control. Design. Application. Modeling.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ANBIMA	–	Associação Brasileira das Entidades dos Mercados Financeiros e de Capitais
API	–	*Application Programming Interface*
APK	–	*Android Application Pack*
BD	–	Banco de Dados
CNDL	–	Confederação Nacional de Dirigente Lojistas
DAO	–	*Data Access Object*
DDNS	–	*Dynamic Domain Name System*
DNS	–	*Domain Name System*
ENEF	–	Estratégia Nacional de Educação Financeira
IDE	–	*Integrated Development Environment*
IP	–	*Internet Protocol*
JAX-RS	–	*Java API for RESTful Web Services*
JWT	–	*JSON Web Token*
MVC	–	*Model, View, Controller*
OCDE	–	Organização para a Cooperação e o Desenvolvimento Econômico
PoC	–	*Proof of Concept*
POJO	–	*Plain Old Java Object*
REST	–	*Representational State Transfer*
SDK	–	*Software Development Kit*
SGBD	–	*Database Management System*
SPC	–	Serviço de Proteção ao Crédito
SOAP	–	*Simple Object Access Protocol*
SQL	–	*Structured Query Language*
UML	–	*Unified Modeling Language*
UFPR	–	Universidade Federal do Paraná
URL	–	*Uniform Resource Locator*
WIP	–	*Work in Progress*

SUMÁRIO

1. INTRODUÇÃO

Durante um grande período de sua história, o Brasil teve uma economia inflacionária, sendo um dos piores cenários a hiperinflação nos anos 80/90. Neste período a população era obrigada a gastar todo o seu dinheiro, pois este ficava totalmente desvalorizado em questão de horas (BASSOTTO, 2018).

Assim, o Brasil passou pelo menos por três trocas de padrões monetários e, também, houve o confisco de poupanças durante o Governo de Fernando Affonso Collor de Mello (1990 – 1992). Desta forma, três gerações de brasileiros conviveram com um alto grau de incerteza sobre as ações do governo no plano econômico. E o hábito de guardar dinheiro, mesmo que em conta corrente, era um risco para poucos (BASSOTTO, 2018).

Como consequência, o povo brasileiro tornou-se desestimulado e descuidado com relação a suas finanças (BASSOTTO, 2018; RIBEIRO, 2016). Desta forma, o termo “educação financeira” tornou-se um assunto, para a maioria dos brasileiros, ignorado.

O termo “educação financeira”, segundo a OCDE (2005, citado pela ENEF, 2017), é o processo no qual os indivíduos melhoram a sua compreensão em relação ao dinheiro e produtos com informação, formação e orientação.

Segundo Domingos (2016, citado por RIBEIRO, 2016), o Brasil tem uma combinação de fatores que faz com que a gestão do orçamento familiar seja colocada em segundo plano. Além de grande parte dos programas de educação financeira ser voltada para questões de como e onde investir; poucos abordam o consumo consciente e o planejamento para futuro.

Isto pode ser comprovado por meio do relatório nacional da ANBIMA (Associação Brasileira das Entidades dos Mercados Financeiros e de Capitais), publicado em 2017. Neste relatório, realizou-se uma pesquisa com 3374 brasileiros, em 152 municípios, dos quais 40% não economizaram sob a justificativa de que todo o valor ganho é destinado a pagar as contas do mês (AMBIMA, 2017).

Além disso, é possível observar a falta de conhecimento dos brasileiros quando o assunto é relacionado a finanças, uma vez que dos entrevistados, apenas 42% já teve algum tipo de saldo aplicado nestes (ANBIMA, 2017).

Também, segundo ANBIMA (2017), o brasileiro não se preocupa em obter lucro com investimentos, a maioria (82%) dos que guardam optam pela poupança, pois se preocupam apenas com a segurança financeira no futuro. O segundo colocado, entre as opções dos investidores, está a previdência privada, com apenas 6% dos investimentos brasileiros, o que reforça que o intuito do brasileiro é a segurança financeira e não melhorar sua rentabilidade com investimentos.

1.1. CONTEXTO DO PROBLEMA

Ao longo do tempo, o mundo financeiro tornou-se mais complexo se comparado às gerações anteriores. No entanto, o nível de educação financeira da população, em especial do brasileiro, não acompanhou o ritmo desta complexidade.

Com isso, o déficit de educação financeira aliada com a facilidade de compra do mundo atual, contribuem para o aumento da inadimplência no Brasil. Segundo um estudo realizado em janeiro de 2019 pelo SPC Brasil (Serviço de Proteção ao Crédito) em conjunto com o CNDL (Confederação Nacional de Dirigente Lojistas), cerca de 62,08 milhões de brasileiros estão negativados. O número equivale a 40,2% da população adulta (CNDL; SPC, 2019).

Frente a esta estatística, o desemprego é um agravante que compõe o resultado da pesquisa, pois no segundo trimestre de 2019, haviam cerca de 12,8 milhões de brasileiros desempregados de acordo com Instituto Brasileiro de Geografia e Estatística (IBGE, 2019). Entretanto, o universo de inadimplência no Brasil é bem maior que o levantamento realizado por este. Ou seja, pode-se dizer que uma grande fatia dessa população está inadimplente devido ao descontrole financeiro.

Ainda de acordo com o estudo realizado pelo SPC Brasil e o CNDL, 45% dos brasileiros não realizam um controle efetivo do próprio orçamento. Entre os 55% que controlam os gastos de alguma forma, mais da metade diz sentir dificuldade na tarefa. Os principais motivos são falta de disciplina em anotar os gastos e rendimentos com regularidade (26%), falta de tempo (12%), dificuldade em encontrar um mecanismo simples de controle (11%) e dificuldade em fazer

cálculos (5%) (CNDL, 2018). Diante aos dados expostos, é possível facilitar e estimular o hábito ao controle financeiro?

Outro fator revelador do estudo é que mesmo entre os que se dizem saber pelo menos um pouco sobre suas finanças (seja um controle total ou parcial), uma parcela significativa de 28% não utiliza um método organizado para fazê-lo, confiam na própria memória para gerir seus recursos financeiros. Como atingir essa parcela e as demais mostrando a real importância de registrar e analisar para possuir um efetivo controle de seu orçamento?

Ao analisar mais sobre a saúde financeira da população, o estudo mostra que seis em cada dez, (56%) brasileiros entrevistados, chegaram ao fim do último mês sem ter conseguido poupar. Um dos fatores contribuintes para esse número, é reflexo da “cultura do imediatismo”, pois 30% dos entrevistados reconhecem ter comprado, nos últimos três meses, algum bem não essencial com que fez com que extrapolasse o seu limite financeiro. Isto evidencia o déficit de educação financeira, uma vez que educação financeira não se restringe ao simples ato de economizar dinheiro. Trata-se também de adotar atitudes conscientes ao impor critérios na utilização dos recursos financeiros e saber planejar as próprias contas para um período de longo prazo. Diante disto, como facilitar o planejamento para uma correta utilização dos recursos financeiros a longo prazo?

Existem metodologias financeiras que ajudam no controle de gastos, uma delas, chamada "50/35/15" que consiste em um método simples de organização orçamentária. Nesta metodologia, os gastos devem ser divididos em três categorias: gastos essenciais (50%), estilo de vida (35%) e investimento (15%) (NUBANK, 2019).

Desta forma, busca-se simplificar a forma de realizar um controle financeiro aplicando uma metodologia simples, de modo a facilitar o acesso às informações complexas e tentar contornar o déficit de educação financeira apresentado pela maioria da população brasileira. Procura-se tornar o planejamento de gastos algo mais atraente e de fácil acesso, assim como o controle de despesas e receitas, auxiliando e guiando as pessoas à uma correta utilização de seus recursos financeiros.

1.2. OBJETIVOS

Nesta seção serão apresentados os objetivos pretendidos neste trabalho.

1.2.1. Objetivo Geral

Desenvolver um aplicativo voltado para o planejamento financeiro pessoal capaz de proporcionar um controle sobre o próprio orçamento e, também, auxiliar na criação de metas para controle de despesas baseado na metodologia “50/35/15”.

1.2.2. Objetivos Específicos

Os objetivos específicos propostos para este trabalho são:

- Permitir a visualização de informações financeiras cadastradas pelo usuário através de diferentes relatórios do sistema;
- Permitir o gerenciamento de dados referentes receitas e despesas;
- Permitir o gerenciamento de diferentes tipos contas (cartão de crédito, conta bancária e investimentos) para organizar e categorizar os dados de receita e despesa.
- Permitir o controle de despesas por meio da imposição de limite de gastos.

1.3. JUSTIFICATIVA

Segundo a Organização para a Cooperação e Desenvolvimento Econômico (OCDE, 2005), a educação financeira é um processo em que o indivíduo faz escolhas conscientes e se mantém informado a respeito da economia para, assim, elaborar a melhor forma de lidar com seu dinheiro a curto e longo prazo. Entretanto, com base na problemática apresentada, muitos brasileiros sentem dificuldade em possuir um completo domínio sobre sua vida financeira.

Nesse contexto, o aplicativo *UpMoney* visa facilitar o controle financeiro através de um aplicativo para celular, o qual facilitará o registro da vida financeira de seu usuário uma vez que o uso de celular está presente no cotidiano de muitos. Dessa forma, o aplicativo proporcionará um método organizado para registrar ganhos e gastos, além de possibilitar uma visão analítica atual e histórica da vida financeira do usuário. Adicionalmente, o trabalho proposto viabiliza também um controle de gastos baseado na metodologia "50/35/15", sendo capaz de analisar e sugerir um planejamento mais factível e personalizável.

1.4. ESTRUTURA DO DOCUMENTO

Além deste, o documento é dividido em mais quatro capítulos.

O Capítulo 2 contém conceitos teóricos relacionados ao método de economia aplicado ao aplicativo *UpMoney*. Além disso, apresenta uma análise de *softwares* que possuem um propósito semelhante ao empregado no aplicativo deste projeto.

Já o Capítulo 3 enfatiza o desenvolvimento do trabalho, ou seja, apresenta as ferramentas utilizadas no projeto e a maneira como este foi desenvolvido. O Capítulo 4 apresenta os resultados atingidos no desenvolvimento do aplicativo. Por fim, no Capítulo 5 são abordadas as considerações finais sobre este trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

A fim de situar o leitor no contexto a que este trabalho se refere, apresenta-se uma breve revisão teórica a respeito do tema planejamento financeiro e suas diferentes metodologias.

Inicialmente, são apresentados conceitos de finanças pessoais, expondo uma abordagem em relação a controle orçamentário pessoal, seguindo para a regra que o aplicativo *UpMoney* utiliza como base de cálculos nas suas diretrizes.

2.1. FINANÇAS PESSOAIS

Ao estudar a história da educação financeira no Brasil retrospectivamente até o fim dos anos 1990, Araújo e Calife (2014, p.1) observaram que esse período foi marcado por altos índices de inflação, associados a baixa bancarização, crédito escasso e pouco acesso à informação. Todos esses fatores construíram um cenário no qual a população brasileira não conseguia planejar sua vida financeira, seja a curto ou a longo prazo. Após a estabilização da moeda com o Plano Real em 1994, alguns conceitos relacionados à gestão financeira pessoal começaram a mudar, a possibilidade de prever o valor do dinheiro ao final de alguns meses ou anos, e planejar um fim para esse valor acumulado, passaram a ser uma realidade viável.

Conseguir alocar as despesas dentro das receitas pessoais, criar e seguir um planejamento financeiro evitando gastos desnecessários dos recursos é um feito que poucos conseguem atingir. Um delineamento adequado possibilita o desenvolvimento de hábitos de poupança,

> [...] poupar exige a avaliação das despesas, a fixação de metas e, principalmente, muita persistência, a fim de manter-se economizando pelo tempo necessário até que sejam alcançados os objetivos que motivaram a poupança (MALMANN, 2008, p.13).

É importante que, cada vez mais, as pessoas se conscientizem sobre a importância de planejamento financeiro, que as informações estejam mais acessíveis e que existam facilitadores para o controle das suas rendas pessoais.

Somente dessa forma, a população se tornará capaz de tomar suas próprias decisões e gerenciar seu próprio patrimônio de maneira consciente e saudável.

De acordo com Bodie e Merton (1999, p.28), as decisões financeiras pessoais podem ser classificadas em quatro diferentes tipos:

a) Decisões de consumo e economia - dividir a renda em duas linhas, quanto gastar em consumo e quanto poupar para o futuro;
b) Decisões de investimento - como investir o dinheiro economizado;
c) Decisões de financiamento - como e quando usar o dinheiro de terceiros para alcançar planos de consumo;
d) Decisões de administração de risco - procurar formas de reduzir as incertezas financeiras, e tomadas de decisão em relação ao quanto aumentar a possibilidade de riscos para alcançar os objetivos.

Todas essas diferentes vertentes apresentadas, se relacionam intimamente. No entanto, o enfoque principal deste trabalho se dará no ato de poupar, estimulado pelo estabelecimento de metas a fim de gerir as finanças pessoais de maneira adequada.

2.2. PLANEJAMENTO FINANCEIRO

Em um planejamento financeiro, algumas características são importantes para que exista uma gestão financeira eficaz. O sucesso depende de autocontrole, força de vontade, flexibilidade, disciplina e foco.

> [...] nenhuma orientação financeira é milagrosa o suficiente (...) para torná-lo bem-sucedido, caso você mesmo não entenda o padrão de comportamento que move seus desejos, consumos, gastos e objetivos (MACEDO JÚNIOR, 2010, p.4).

Para Busetti (2012, p.33), é importante que a flexibilidade esteja presente no planejamento e na administração financeira. A automatização de controle de metas para receitas e gastos – mesmo os de valores irrisórios – é complicada, os imprevistos acontecem e nesses momentos é necessário que exista autocontrole e disciplina para que sejam feitos remanejamentos de gastos de maneira consciente, de modo que não impacte o objetivo final.

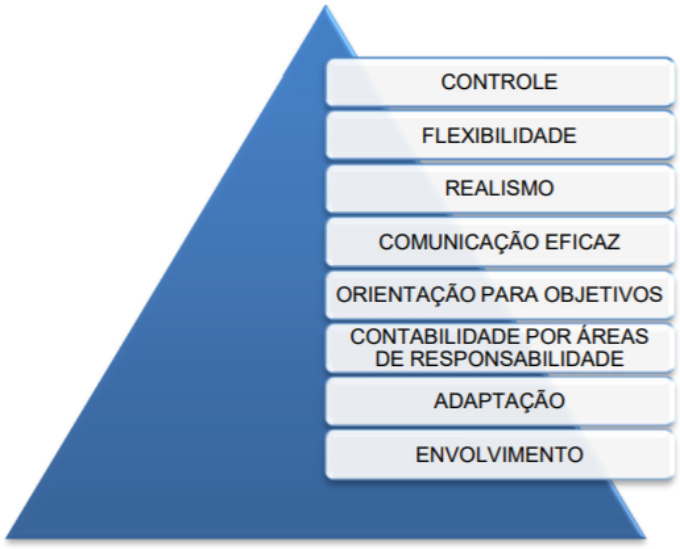
O foco se faz necessário, pois em planos a longo prazo, o acúmulo de riqueza se torna uma tentação para gastos não planejados. Busetti (2012, p.33)

concorda com a ideia de Macedo Júnior (2010, p.4) de que na sociedade em que se vive atualmente, a cultura do consumismo está enraizada no ser humano, tornando difícil a criação do hábito de economizar. Portanto, para que haja um fluxo confortável no objetivo de se criar um planejamento financeiro pessoal é preciso que alguns princípios fundamentais sejam ressaltados nessa jornada.

Busetti (2012, p.34) adapta alguns preceitos de âmbito empresarial para o pessoal, como pode se observar na FIGURA 1 e descreve-os da seguinte maneira para que uma boa gestão financeira seja alcançada:

- Envolvimento: é fundamental que exista uma proximidade pessoal na comunicação e objetivos;
- Adaptação: é importante que o orçamento seja adaptado a nível pessoal e particular, tendo compatibilidade com a situação econômico-financeira de quem o manipula;
- Contabilidade por áreas responsáveis: é necessário que para que a gestão financeira obtenha sucesso, exista uma orientação profissional no envolvimento das divisões de receitas e despesas, separando-os em grupos de afinidade;
- Orientação para objetivos: os objetivos e metas devem ser claros, realistas e mensuráveis para que o controle seja eficiente e eficaz;
- Comunicação eficaz: a didática é de suma importância e deve ser comunicada de maneira apropriada a quem a utiliza;
- Realismo: Os objetivos e metas devem ser difíceis de ser alcançados para que seja incitado à motivação, no entanto devem ser possíveis e atingíveis, o equilíbrio é fundamental;
- Flexibilidade: O orçamento não deve ser forçado, deve apresentar características flexíveis para servir como um guia, uma ferramenta de apoio para que decisões sejam dotadas de racionalidade;
- Controle: deve ser diário, permitindo um monitoramento eficaz, de maneira que o processo orçamentário seja alimentado com informações que possam ser úteis para o próximo ciclo.

FIGURA 1 – PRINCÍPIOS DO PLANEJAMENTO FINANCEIRO



FONTE: Adaptado de BUSETTI (2012, p.35).

2.3. MÉTODOS PARA CONTROLE FINANCEIRO

Um controle adequado sobre os gastos permite ao cidadão ajustar, a partir de sua renda, suas necessidades. Para isso, é essencial um amplo conhecimento sobre sua vida financeira, no quesito de conhecer seu potencial econômico, para conseguir estabelecer metas, prioridades e prazos palpáveis para a realização.

Nesse sentido, um conjunto de controle e procedimentos, torna-se imprescindível para criar um orçamento, acompanhar gastos, possíveis sobras ou falta de recursos e também para se precaver de possíveis mudanças econômicas.

O controle de finanças é algo constante e mutável, cada indivíduo pode se organizar de maneira diferente. Entretanto, algumas premissas são primordiais, como para Cerbasi (2013), deve-se sempre ter inicialmente o conhecimento de todos os fatores relevantes para se fazer um bom planejamento financeiro; são eles: i) os valores de todas as receitas e despesas; ii) seus objetivos e metas; e iii) suas prioridades.

A partir da ideia inicial de Cerbasi (2013), é possível identificar similaridade de conceito nas premissas em estudo de outros autores como de Blanco (2014),

> [...]Quando o fluxo de caixa estiver bem detalhado, é possível fazer estimativas e previsões do que se vai receber, gastar e investir nos próximos meses e anos. Com isso, você estará elaborando um orçamento, processo de estimar e controlar as despesas e gastos, buscando um equilíbrio com as receitas. É instrumento básico para melhorar a sua vida financeira, seja para aumentar os investimentos ou se livrar das dívidas. Ajuda a definir os gastos e monitorar o seu desempenho nesta tarefa (BLANCO, 2014, p.12).

Apesar das premissas serem semelhantes entre vários autores, os métodos a serem aplicados para alcançar um controle financeiro eficaz podem ser distintos. Por exemplo, para Ewald (2003, p. 13) um fidedigno orçamento se dá por três fases distintas: i) Avaliação aleatória do valor das despesas que o indivíduo acha que estão sendo feitas durante um mês; ii) Acompanhamento e apuração no mês seguinte das despesas realmente efetuadas; e iii) Avaliação, programação de possíveis cortes e previsão dos valores que poderão ser gastos no mês seguinte.

Para Warren (2005), um controle de orçamento deve seguir uma regra de segmentação. Na qual, 50% do valor deve ser destinado com gastos essenciais, são despesas necessárias para viver, 35% estilo de vida, gastos com atividades rotineiras e 15% para poupar.

Diante de vários autores sobre o assunto em questão, é possível notar semelhanças nos conceitos básicos para iniciar um controle financeiro adequado e também diferenças nas formas como se atinge o mesmo. Entretanto, todas as formas visam obter uma qualidade financeira, através de métodos específicos. Essa característica é identificada também em aplicativos que possuem a proposta para se obter um controle financeiro, pois possuem métodos para auxiliar o usuário.

2.4. SOFTWARES DE PLANEJAMENTO FINANCEIRO

A tarefa de organizar as finanças, cortar alguns gastos e saber qual a melhor maneira de investir o dinheiro é uma necessidade que está crescendo cada vez mais. Apesar de necessário, essa tarefa pode aparentar ser complexa

e trabalhosa para pessoas que não têm tempo de registrar suas despesas e gastos de maneira rotineira. Com a popularização dos *smartphones*, os aplicativos passaram a ser fortes aliados na área de gestão financeira, a facilidade de acesso que os aplicativos financeiros proporcionam, não apenas dão uma perspectiva melhor sobre finanças, como também ajudam a entender quais tipos de gastos podem ser evitados. Na próxima seção analisaremos as principais aplicações disponíveis no mercado.

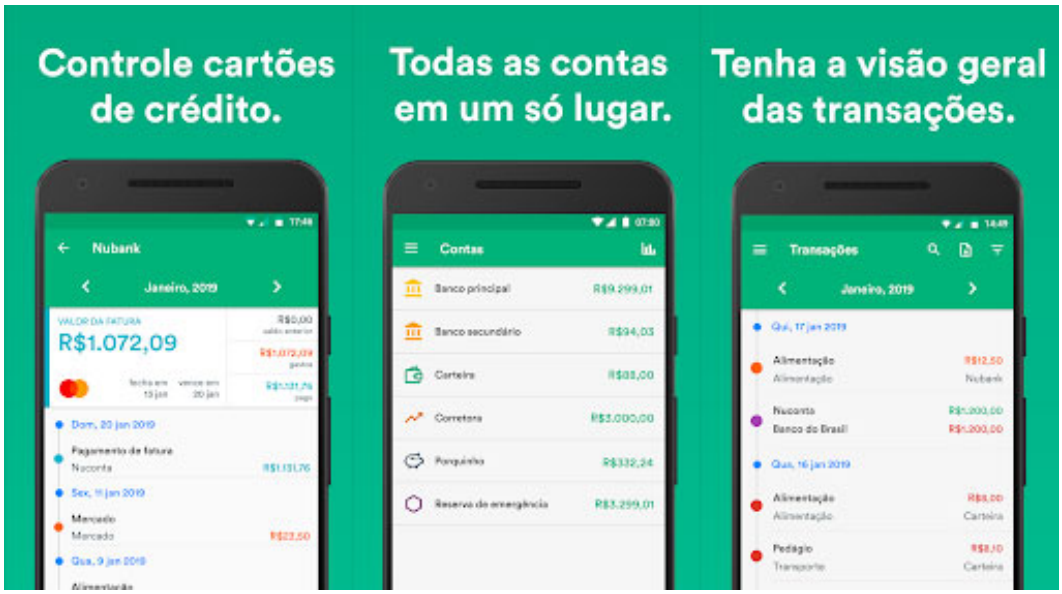
2.4.1. Aplicativos Similares

Nessa seção serão apresentados os aplicativos de controle financeiro analisados com um resumo das suas principais funcionalidades.

2.4.1.1. Fortuno

Disponível apenas para a plataforma *Android*, o Fortuno é um aplicativo que permite o agrupamento de contas para determinar quanto dinheiro o usuário dispõe para gastos, conforme ilustrado na FIGURA 2. Esta aplicação trabalha com relatórios e gráficos, possui opções de notificações de vencimento de pendências financeiras e mantém todas as informações salvas na nuvem. Fornece diferentes temas, com opções de personalização (FORTUNO, 2019).

FIGURA 2 – APLICATIVO FORTUNO



FONTE: GOOGLE PLAY (2009).

2.4.1.2. Guia Bolso

O Guia Bolso teve início como um site em 2014 e desde então veio sofrendo melhorias tanto visuais quanto de funcionalidades, possui a possibilidade de integração com os cartões de créditos e contas bancárias. A aplicação também permite interações com pessoas de mesmo perfil, o que auxilia na organização do usuário em parametrizar a sua média de despesas, permite cadastro e gerenciamento de contas e receitas, cadastro de sonhos, entre outras funcionalidades. Por fim, o Guia Bolso dispõe da possibilidade de contratação de empréstimos e comparação de taxas, conforme ilustrado na FIGURA 3 (GUIABOLSO, 2019).

FIGURA 3 – APLICATIVO GUIA BOLSO

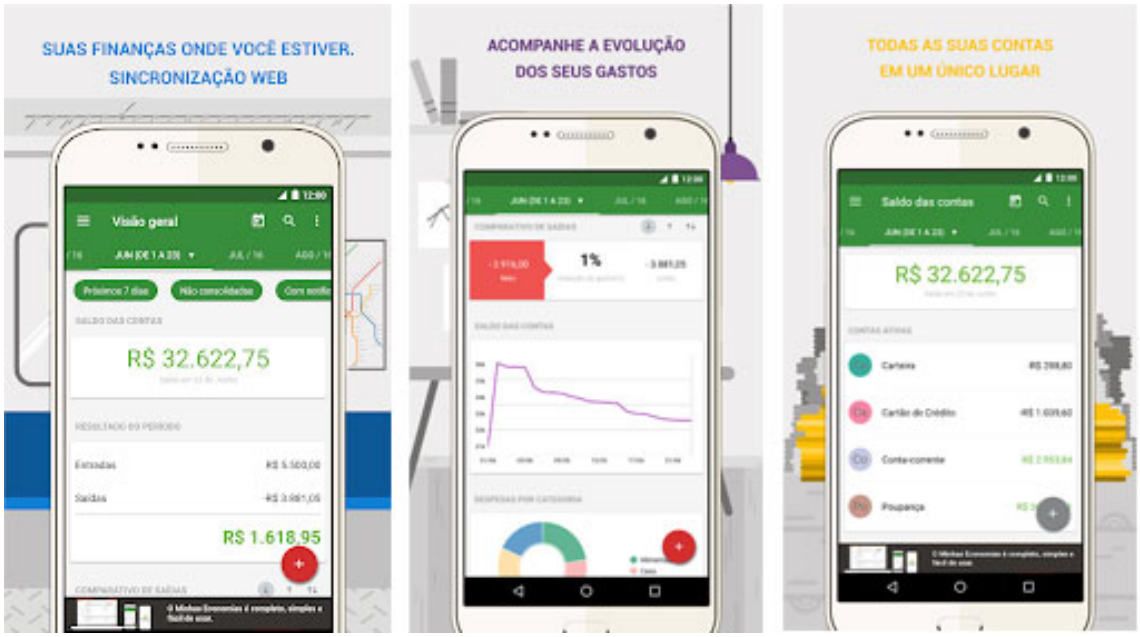
FONTE: GOOGLE PLAY (2009).

2.4.1.3. Minhas Economias

O aplicativo Minhas Economias está disponível para *iOS* e *Android*, e possui um objetivo mais voltado para contas a pagar e receber. Permite o gerenciamento de gastos, cadastro de contas, cartões de crédito e investimento. Com o cadastro feito no aplicativo, o usuário também ganha acesso ao site do aplicativo que é um gerenciador mais completo, no qual é possível gerenciar sonhos, gráficos e relatórios, conforme ilustrado na FIGURA 4. Está disponível

também a categorização de transações, importação de arquivos em formato .ofx, programação de repetição de transações e cadastro de lembrete para não atrasar os pagamentos (MINHAS ECONOMIAS, 2019).

FIGURA 4 – APLICATIVO MINHAS ECONOMIAS

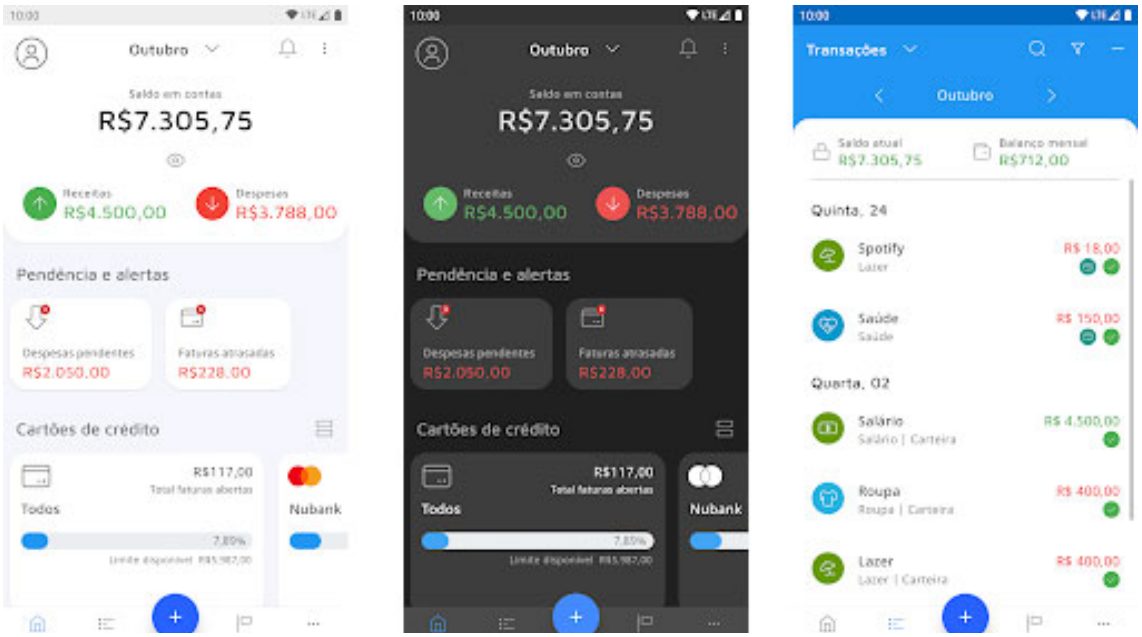


FONTE: GOOGLE PLAY (2009).

2.4.1.4. Mobills

No Mobills é possível gerenciar as despesas por geolocalização, o que ajuda aos usuários compreender os lugares físicos específicos que gastam mais. Possui sincronização dos dados na nuvem, parceria com o programa de pontos *Multiplus* e está disponível para todas as plataformas (*Android*, *iOS* e *Web*). Disponibiliza relatórios em PDF (*Portable Document Format)* e Excel, ideal para usuários que estão mais acostumados com o controle feito em papel. A inclusão de contas, despesas, cartões e empréstimos também são funcionalidades presentes no aplicativo. O diferencial em relação ao *layout* é a possibilidade de mudar os temas da aplicação para modo noturno, conforme ilustrado na FIGURA 5 (MOBILLS, 2019).

FIGURA 5 – APLICATIVO MOBILLS



FONTE: GOOGLE PLAY (2009).

2.4.1.5. Monefy

Monefy é um aplicativo cujo principal objetivo é o acompanhamento de despesas, conforme ilustrado na FIGURA 6, sua interface é baseada no uso de ícones para categorizar os gastos do usuário. Dispõe um *widget* que pode ser habilitado para a tela do celular, facilitando ao usuário o cadastro das despesas realizadas, e o controle do saldo total das contas cadastradas. Exporta dados no formato .csv e permite a sincronização com conta na nuvem Dropbox (MONEFY, 2019).

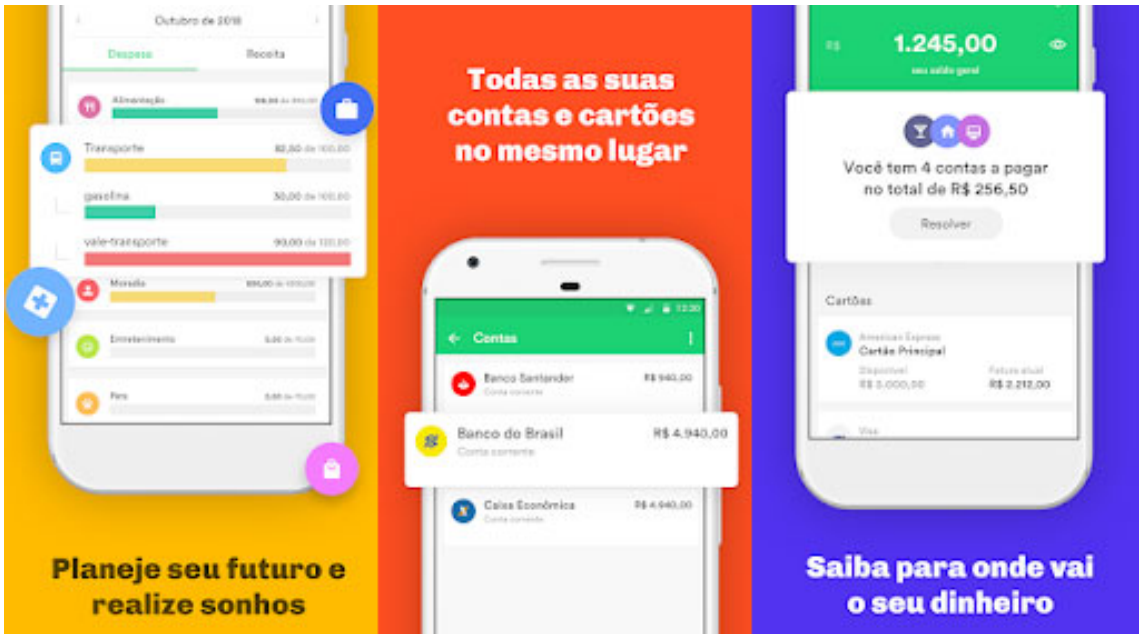
FIGURA 6 – APLICATIVO MONEFY

FONTE: GOOGLE PLAY (2009).

2.4.1.6. Organizze

O Organizze é um aplicativo voltado para o estabelecimento de metas pessoais, e limite de gastos e receitas para cada uma das categorias cadastradas, conforme ilustrado na FIGURA 7. Também, possui a opção de salvar os dados na nuvem, conferência de relatórios, importação de arquivos .ofx, vínculo de contas bancárias e cartões de crédito e configuração de alertas, está disponível para *Android* e *iOS* (ORGANIZZE, 2019).

FIGURA 7 – APLICATIVO ORGANIZZE



FONTE: GOOGLE PLAY (2009).

2.4.2. Metodologia de Análise

Por meio do levantamento e análise de similares, buscam-se produtos existentes e carências de mercado. Segundo Bono (2000), a técnica PNI propõe o levantamento de pontos positivos, negativos e interessantes de um produto ou ideia. Seguindo como base os aspectos abordados na técnica do PNI, aplicamos a análise de similares nos aplicativos apresentados na seção 2.4.1. Os pontos interessantes foram convertidos em uma análise das aplicações como um todo, como pode-se observar na QUADRO 1.

QUADRO 1 - DEMONSTRATIVO DA COMPARAÇÃO DAS PRINCIPAIS FUNCIONALIDADES DOS APLICATIVOS APRESETADOS NA SEÇÃO 1.4.1

Funcionalidades	Monefy	Guia Bolso	Minhas Economias	Fortuno	Mobils	Organizze
Visualizar Gastos por Categoria	x					
Criar categorias	x		x	x		
Logar sem cadastro	x					
Integrar com bancos		x				
Tutorial de utilização		x				
Filtrar relatório mensal		x	x			
Logar via redes sociais			x	x	x	
Cadastrar contas bancárias			x			
Cadastrar cartão de crédito			x	x	x	
Categorizar gastos	x		x		x	
Filtrar relatório anual, período			x	x		
Cadastrar receitas			x	x	x	
Cadastrar despesas			x	x	x	
Cadastrar contas parceladas			x	x		
Cadastrar anotações em uma despesa			x			
Consultar saldos futuros			x			
Programar notificações por e-mail / notificação			x	x	x	
Visualizar relatórios			x		x	x
Gerenciar metas			x		x	x
Arquivar despesa			x			
Visualizar saldo por diferentes análises			x			
Configurar visualização inicial			x		x	x
Importar extrato bancário			x	x	x	x
Exportar relatórios			x	x	x	
Cadastrar transições offline			x			
Alterar foto de perfil usuário				x		
Recomeçar dados cadastrados				x		
Cadastrar periodicidade despesa				x		
Cadastrar transferência			x	x	x	
Agendar receitas a receber				x		
Cadastrar Despesa por QRCode					x	
Inserir receita ou despesa em diversas moedas					x	
Cadastrar despesa por notificação					x	
Anexar arquivos						x

FONTE: Os Autores (2019).

Dentre as funcionalidades analisadas na QUADRO 1, selecionou-se as principais tomando como critérios de análise os aspectos descritos abaixo:

- Usabilidade/ Design: nos similares analisou-se a questão de usabilidade dos aplicativos. Esse item envolve questões como fluxo entre telas, facilidade de acesso às informações, ou seja, se as informações mais importantes estavam disponibilizadas de maneira acessível durante toda a interação do usuário com a aplicação. Interface amigável e intuitiva, se existia uso de ícones, qual o estilo adotado e se eram autoexplicativos, entre outras funcionalidades relacionadas ao design e usabilidade do aplicativo;
- Acompanhamento e gerenciamento de metas: a análise de acompanhamento e gerenciamento de metas foi feita levando em consideração a maneira que o progresso desse objetivo é disponibilizado ao usuário. Quais as formas de alterações e ajustes disponíveis e se o detalhamento de situações como por exemplo cumprimento ou descumprimento de metas foram dispostos de forma clara e objetiva;
- Gerenciamento de gastos: este tópico abordou uma análise sobre a forma de gerenciar as despesas cadastradas e como foi trabalhado a informação para mostrar a realidade da vida financeira do usuário;
- Visualização de relatórios: nesse item analisou-se a forma como foi apresentado os dados da vida financeira do usuário, para isso foi levado em consideração a facilidade em compreender as informações expostas, o formato de gráfico escolhido de acordo com os dados apresentados e relevância da informação analisada;
- Tela inicial (*Dashboard*) - informações principais: para avaliação deste item, foi levado em consideração alguns critérios de usabilidade como engajamento e facilidade em aprender a usar o produto, além de ter sido avaliado a capacidade de proporcionar ações rápidas no aplicativo.

2.4.2.1. Usabilidade/Design

O visual limpo e atrativo ao usuário é um ponto comum entre todos, assim como o design moderno e um bom uso de cores. Apesar do objetivo de os aplicativos serem relacionados a finanças que é tido como um assunto maçante e complicado, a disponibilização das informações é feita de forma simples e direta, fazendo uso de cores vibrantes e complementares, retirando essa ideia sóbria que esse assunto carrega.

Os aplicativos foram capazes de trazer leveza e agilidade ao planejamento financeiro do usuário. Por terem características visuais atrativas e o cadastro de despesas e receitas ser facilitado pelo uso de tecnologias disponíveis, como leitura de *QrCode*, essa manutenção do aplicativo que é obrigatória ao usuário se tornou uma atividade simples.

O lado negativo que se notou, sendo o mais prejudicial em alguns casos, foi o excesso de disponibilização de informações na mesma tela. Isso tornou o visual poluído causando a desistência da leitura por parte do usuário.

Pontos positivos:

- *Dashboard* personalizável;
- Design limpo (ícones arredondados);
- Possibilidade de adicionar despesa, receita, conta durante todo o fluxo de interação;
- Leitura de despesas por *QrCode*;
- Informações principais disponibilizadas na tela inicial.

Pontos negativos:

- Algumas opções possuíam área de clique muito pequena;
- Excesso de funcionalidades disponibilizadas em uma única tela, causando confusão ao usuário;
- Design estilo *desktop* (linhas muito retas e falta de atratividade visual).

2.4.2.2. Acompanhamento e Gerenciamento de Metas

A maioria dos aplicativos apresenta uma visualização boa sobre o progresso da meta, tanto para metas de despesas quanto para uma meta de

longo prazo. A barra de progressão auxilia o usuário a ter uma visão rápida sobre o seu andamento. Em alguns casos as metas podem apenas receber depósitos, ou seja, não permitem ajustes nos valores que foram inseridos, a única forma de atingir uma meta é adicionando depósitos manualmente.

Pontos positivos:

- Visualizar progresso da meta, com barra progressiva e porcentagem;
- Criar metas para despesas;
- Relatório de depósitos na meta.

Pontos negativos:

- O gerenciamento de metas na maioria dos aplicativos é disponibilizado apenas na versão *Premium*;
- Não é possível retirar depósitos ou adicionar valores negativos;
- Não é possível ajustar o tempo da meta.

2.4.2.3. Cadastro de Despesas e Receitas

A questão de cadastro em todos os aplicativos é simples e intuitiva, os campos são claros e autoexplicativos. Em nenhum momento houve dúvida no preenchimento do cadastro, em alguns casos são utilizados facilitadores visuais como ícones, ou logo na primeira utilização do aplicativo é apresentado um tutorial que ensina a cadastrar receitas e despesas, dessa forma o usuário já inicia o uso com informações cadastradas. A importação de arquivos no formato .ofx também facilita o cadastro das despesas e receitas presentes no extrato das contas de usuários.

Alguns aplicativos apresentaram um problema no cadastro de compras parceladas, estas precisavam ser inseridas mês a mês o que gera retrabalho ao usuário. No entanto, alguns já disponibilizavam a programação para lançamentos futuros automáticos de despesas e receitas.

Pontos positivos:

- Categorização de despesas e receitas;
- Utilização de ícones ou cores;
- Opção de repetição;

- Salvar uma receita ou despesa como favorita;
- Adicionar uma notificação;
- Programar lançamento automático de receitas;
- Importação de arquivos formato .ofx.

Pontos negativos:

- Compras parceladas precisam ser inseridas mês a mês.

2.4.2.4. Gerenciamento de Gastos

O gerenciamento é realizado através de informações disponíveis em gráficos como a visualização de quanto já foi gasto no mês, com o que foi gasto. É possível realizar o cadastro de um valor planejado a se gastar com cada categoria de despesa e ao decorrer do período planejado pode-se acompanhar se os gastos ficaram dentro do esperado. A possibilidade de configurar o recebimento de notificações quando o usuário irá estourar um limite estabelecido de gastos em determinada categoria, facilita esse gerenciamento.

Pontos positivos:

- Facilidade em visualizar o que já foi gasto;
- Possibilidade de comparações entre categorias com maiores gastos;
- Planejar gastos futuros;
- Programar alertas para pagamento de despesas;
- Determinar um limite de gastos para as despesas e acompanhamentos dessas metas.

Pontos negativos:

- Não foram identificados pontos negativos.

2.4.2.5. Visualização de Relatórios

Todos os aplicativos analisados dispõem de informações sobre o registro financeiro do usuário em formato de relatórios. Apesar da pouca variedade de gráficos para expor os números, a grande maioria utiliza gráficos comparativos para demonstrar a situação.

Pontos positivos:

- Fácil entendimento sobre os dados expostos;
- Uso adequado de cores para segmentar a visualização;
- É possível visualizar relatórios de meses históricos;
- Variedade de informações distribuídas em relatórios.

Pontos negativos:

- Pouca variedade nas opções de visualização da informação.

2.4.2.6. Dashboard (Tela Inicial)

Em todos os aplicativos analisados, a interface inicial é intuitiva de modo a fornecer rápido acesso a outras funcionalidades, sempre há uma consolidação da situação atual do usuário facilitando o controle.

Pontos positivos:

- *Dashboard* personalizável;
- Facilidade em registrar despesas e receitas;
- Informações principais consolidadas;
- Diferentes informações segmentadas em diferentes *cards*.

Pontos negativos:

- Não foram identificados pontos negativos.

Após esse levantamento de características realizado entre os aplicativos similares, foi possível captar qualitativamente os pontos positivos e negativos dos concorrentes, identificou-se boas práticas que deverão ser levados em consideração no desenvolvimento do produto apresentado neste trabalho.

No próximo capítulo, serão apresentados os materiais e métodos propostos no desenvolvimento deste projeto.

3. MATERIAIS E MÉTODOS

O processo de desenvolvimento de software pode ser entendido, segundo Pressman e Maxim (2016), como uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um software de alta qualidade. Desta forma, cada uma das atividades, ações e tarefas alocam-se dentro de uma metodologia ou modelo que determina sua relação com o processo e uma com as outras (PRESSMAN; MAXIM, 2016).

Ainda segundo Pressman e Maxim (2016), uma metodologia de processo genérica para a engenharia de software estabelece cinco atividades metodológicas: comunicação, planejamento, modelagem, construção e entrega. Além destas, um conjunto de atividades de apoio é aplicado ao longo do projeto, como, por exemplo, acompanhamento e controle do projeto, a administração de riscos, a garantia da qualidade, o gerenciamento das configurações, as revisões técnicas, entre outras (PRESSMAN; MAXIM, 2016).

Desta forma, nesta seção serão apresentados os métodos, procedimentos, técnicas e ferramentas utilizados no desenvolvimento deste projeto, sendo primeiramente, apresentada a metodologia de desenvolvimento de *software* utilizada.

3.1. METODOLOGIAS ÁGEIS

Há na comunidade, um grande debate sobre o que é ser “ágil”. Ao contrário de outras culturas de desenvolvimento, agilidade não está relacionada à obediência de protocolos preestabelecidos de produção, mas a novos padrões de comportamento e atitude. Portanto uma equipe não pode se dizer “ágil” se não se comportar assim (PRIKLADNICKI, WILL; MILANI, 2014, p.5).

Segundo Prikladnicki, Willi e Milani (2014, p.5) cada Método Ágil define suas próprias práticas, mas todos, em um momento ou outro, compartilham dos valores e princípios postulados pelo Manifesto Ágil.

Ele declarava:

> Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através desse trabalho, passamos a valorizar:
> Indivíduos e interação mais que processos e ferramentas

> Software funcionando mais que documentação de contratos
> Colaboração com o cliente mais que negociação de contratos
> Responder a mudanças mais que seguir um plano
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens da esquerda (Prikladnicki; Willi; Milani, 2014, p.5).

A partir deste manifesto obteve-se os princípios a seguir, segundo Ribeiro e Ribeiro (2015, p.23):

- Satisfação do cliente por meio da entrega contínua e adiantada de software com valor agregado;
- Mudanças nos requisitos são bem-vindas, mesmo que tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
- Entregas frequentes de software funcionando, com preferência à menor escala de tempo;
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho;
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é por meio de conversas face a face;
- Software funcionando é a medida primária de progresso. Os processos ágeis promovem desenvolvimento sustentável, enquanto que os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
- A atenção contínua à excelência técnica e bom design aumenta a agilidade;
- Simplicidade, sendo a arte de maximizar a quantidade de trabalho não realizado essencial;
- As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis;
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Desta forma, devido às razões citadas anteriormente, optou-se por aplicar as metodologias ágeis apresentadas a seguir, adaptando-as conforme a necessidade deste projeto.

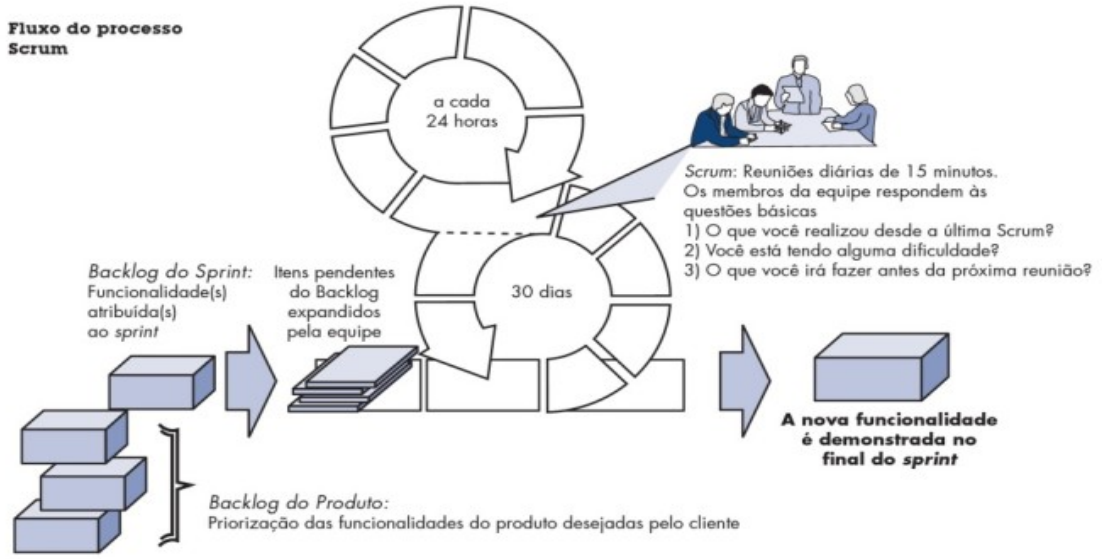
3.1.1. SCRUM

O *Scrum* é um *framework* ágil que auxilia no gerenciamento de projetos complexos e no desenvolvimento de produtos. É conhecido como um *framework* que prescreve um conjunto de práticas leves e objetivas, muito utilizadas na área de desenvolvimento de *software* (PRIKLADNICKI; WILLI MILANI, 2014, p.22).

Além disso, o *Scrum* tem como premissa a existência de um processo iterativo e incremental para o desenvolvimento, trazendo uma nova dimensão na capacidade de resposta e adaptabilidade de gestão dos processos (PRIKLADNICKI; WILLI; MILANI, 2014, p.22).

Ainda, segundo Pressman e Maxim (2016, p.78), os princípios utilizados no *Scrum* são coerentes com o Manifesto Ágil e orientam o desenvolvimento das atividades em um processo que incorpora as seguintes atividades metodológicas: requisitos, análise, projeto, evolução e entrega. O Fluxo de um processo *Scrum* é apresentado na FIGURA 8.

FIGURA 8 – FLUXO DO PROCESSO SCRUM



FONTE: PRESSMAN E MAXIM (2016).

Segundo Ribeiro e Ribeiro (2015), os recursos humanos envolvidos no projeto utilizando o framework *Scrum* possui três papéis: *Product Owner, Scrum Master, Dev. Team*.

O *Product Owner* (dono do produto) representa o cliente no time *Scrum*, ele é responsável por entender o que o cliente precisa e transportar este conhecimento para os desenvolvedores. Já o *Scrum Master* é responsável por garantir que o *Scrum* seja entendido e aplicado, ou seja, garantir que o time *Scrum* siga a teoria, as práticas e as regras do *Scrum*. Por fim, o *Dev. Team*, ou equipe de desenvolvimento, são profissionais encarregados de entregar uma versão usável que potencialmente é adicionada ao produto “Final" ao término de cada *sprint* (RIBEIRO; RIBEIRO, 2015).

Segundo Prikladnicki, Willi e Milani (2014, p.29) o *Backlog* da *Sprint* é o conjunto de itens selecionados para serem implementados durante a *Sprint* mais o plano para transformá-los em um Incremento.

Assim, ao final de cada Reunião de Planejamento de *Sprint*, um novo *Backlog* da *Sprint* é criado. Normalmente, o plano é composto por uma meta, os itens selecionados e as tarefas técnicas necessárias para transformar o item em um incremento do produto (PRIKLADNICKI; WILLI; MILANI, 2014, p.30).

O fluxo do processo *Scrum* (FIGURA 8) é apresentado da seguinte maneira: as *sprints* são compostas por uma reunião de *Planning Meeting*

(Planejamento da *sprint*), *Daily Scrum* (Reuniões Diárias), o trabalho de desenvolvimento, uma *Review Meeting* (Revisão da *sprint*) e *Retrospective Meeting* (Retrospectiva da *sprint*) (RIBEIRO; RIBEIRO, 2015, p.59).

Na *Planning Meeting* é definido o que será entregue e como a entrega será realizada. Esta reunião é realizada no início da *sprint* e todos os envolvidos no projeto participam. Já a *Daily Scrum* é uma reunião informal e rápida (aproximadamente 15 minutos), da qual participam apenas os integrantes do time *Scrum*. Nesta reunião, geralmente, cada integrante expõe o que fez desde a última reunião, o que irá fazer até a próximo e o que está dificultando o desenvolvimento do trabalho (RIBEIRO; RIBEIRO, 2015, p.59-p.61).

Ainda, segundo Ribeiro e Ribeiro (2015, p.61), a *Review Meeting*, ou, Reunião de Revisão, consiste em uma reunião, na qual o *time Scrum* apresenta ao cliente às tarefas concluídas na *Sprint* e o *feedback* é solicitado.

Por fim, a *Retrospective Meeting*, ou, em português, Reunião de Retrospectiva da *sprint*, é um recurso de melhoria contínua, uma ferramenta de comunicação e evolução do time. Desta forma, após a reunião de revisão, o time de desenvolvimento e o *Scrum Master* realizam uma retrospectiva para determinar o que eles fizeram bem que irão manter e o que foi ruim na *sprint*, e quais as recomendações de melhoria que irão seguir daqui para frente. Assim, um plano de ação é criado e esses itens são implementados ao longo da próxima *Sprint*, e revisado para a eficácia na retrospectiva da *sprint* seguinte (RIBEIRO; RIBEIRO, 2015, p.61-p.62).

No desenvolvimento deste trabalho, a metodologia *Scrum* foi utilizada de modo adaptado, uma vez que não existiu o papel do *Product Owner*, já que é um projeto de iniciativa dos próprios desenvolvedores.

Ademais, as atividades de todas as *sprints* da primeira parte do projeto foram definidas na primeira *sprint,* desta forma foram realizadas apenas reuniões de *Planning Meeting* adaptadas, nas quais se revia as *sprints*, realizava-se correções, discutia-se dificuldades enfrentadas e buscavam-se soluções para estas, dividia-se tarefas e discutia-se outras questões relevantes.

Na próxima seção será explicado o *Kanban*, o qual foi utilizado junto ao *Scrum* no desenvolvimento deste projeto.

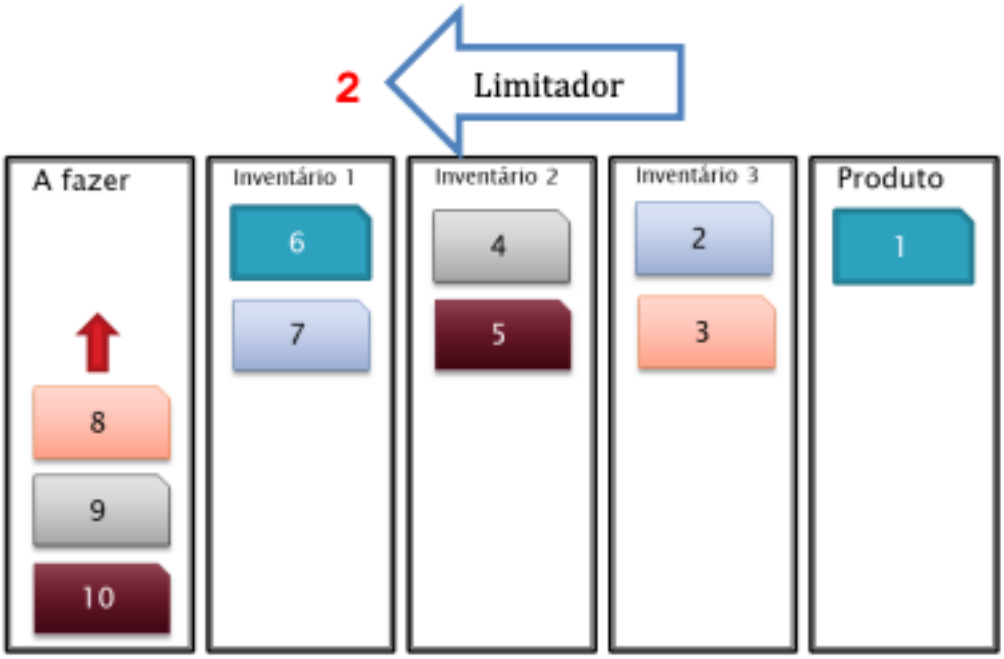
3.1.2. KANBAN

O *Kanban*, cujo significado é "cartão sinalizador", consiste em uma ferramenta visual que auxilia o acompanhamento do fluxo de trabalho e controle do WIP (Trabalho em Progresso, do inglês *Work in Progress*) (RIBEIRO; RIBEIRO, 2015, p.33).

No desenvolvimento de software, os cartões representam itens de trabalho, no entanto, a quantidade de WIPs é limitado por algum indicador de limite, normalmente associado a capacidade de atendimento da equipe. Desta forma, o *Kanban* permite a visualização em progresso o que permite aos desenvolvedores se auto organizarem, atribuindo suas próprias tarefas e movendo o trabalho de um *backlog* para a conclusão sem orientação de um gerente de projeto ou linha (ANDERSON, 2011).

Normalmente, os cartões são dispostos em um quadro. Segundo Ribeiro e Ribeiro (2015, p.34), o quadro *Kanban* permite visualizar o trabalho que está em andamento e limitar o WIP. Tradicionalmente, não sendo uma regra, o quadro *Kanban* é dividido em lista ou estados como *DO* (a fazer), *DOING* (fazendo) e *DONE* (feito), as tarefas que precisarão ser realizadas são listadas ("ou coladas") na parte *DO*, ao inicializá-las, as mesmas são movidas para o quadro *DOING* e quando são finalizadas vão para o status *DONE*. Todavia, os status podem ter outros rótulos, por exemplo, “Modelado”, “Em Desenvolvimento”, “Desenvolvido”, “Em Teste”, “Em Implantação”, “Pronto”, ou qualquer outro rótulo que faça sentido para o trabalho e para a equipe. Na FIGURA 9, é apresentado um quadro *Kanban* com um limitador igual a duas atividades por inventário.

FIGURA 9 – EXEMPLO QUADRO KANBAN COM LIMITADOR



FONTE: RIBEIRO E RIBEIRO (2019).

Para este trabalho, utilizou-se uma adaptação do *Kanban* para delegar tarefas das *sprints* definidas no *Scrum*. No entanto, o limitador de atividades em andamento não foi definido.

Na seção seguinte, apresenta-se os elementos de modelagem utilizados neste projeto.

3.2. MODELAGEM

A modelagem é uma parte central de todas as atividades que levam à implantação de um bom software. Construímos modelos para se comunicar a estrutura e o comportamento desejados do sistema. Construímos modelos para visualizar e controlar a arquitetura do sistema. Construímos modelos para compreender melhor o sistema (BROOCH; RUMBAUGH; JACOBSON, 2006, p.20).

Segundo Brooch, Rumbagh e Jacobson (2006, p.33) a UML (*Unified Modeling Language*) é uma linguagem-padrão para a elaboração de estrutura de projetos de software. Ela poderá ser empregada para a visualização, a

especificação, a construção e a documentação de artefatos que façam o uso de sistemas complexos de software.

Uma linguagem de modelagem é a linguagem cujo vocabulário e regras têm seu foco voltado para a modelagem, como a UML, é uma linguagem-padrão para a elaboração da estrutura de projetos de software (BROOCH; RUMBAUGH; JACOBSON, 2006, p.34).

O *Astah Community* é uma ferramenta gratuita voltada para a modelagem UML conhecida por sua praticidade e simplicidade em elaborar diagramas. Além deste, existem outras versões que disponibilizam outras funcionalidades que não a modelagem UML, no entanto, com licença comercial (BRONDANI et al, 2013).

Desta forma, utilizou-se a ferramenta *Astah Community* para desenvolver os diagramas de caso de usos, apresentado no APÊNDICE B e APÊNDICE C, o diagrama de classes, APÊNDICE E, e o diagrama de sequência, o qual pode ser visualizado no APÊNDICE F, deste trabalho.

Na próxima seção, é apresentada como a prototipação de telas foi realizada.

3.3. PROTOTIPAÇÃO

De acordo com Sommerville (2010, p.45), um protótipo é uma representação inicial do software, utilizado para demonstrar conceitos, ideias e opções de design. Pode ser desde um esboço em papel, uma simulação em vídeo, uma maquete ou um conjunto de telas vinculadas por *hyperlinks.*

O Figma é um software de prototipação, que possui recursos colaborativos em tempo real, desta forma, múltiplos usuários podem ter acesso ao mesmo projeto e contribuir com código e dar ou receber *feedbacks* simultaneamente (FIGMA, 2019).

Ainda, segundo Fidma (2019), no Figma, é possível trabalhar com ferramentas de criação como vetor e conexão à nuvem. Isso facilita o processo de desenvolvimento, uma vez que elimina a etapa de instalação.

Portanto, para o desenvolvimento deste projeto, utilizou-se essa ferramenta por apresentar uma interface amigável, e disponibilizar uma variedade grande de opções colaborativas e ferramentas de criação que se

preocupam com o design e a alta fidelidade de uma prototipação. A prototipação das telas é apresentada nas especificações de caso de usos, localizados no APÊNDICE D.

Na seção seguinte são apresentadas as tecnologias propostas para o desenvolvimento do aplicativo.

3.4. TECNOLOGIAS DE DESENVOLVIMENTO

Devido a praticidade e facilidade de acesso a qualquer momento, optou-se pelo desenvolvimento de um aplicativo de *smartphone*.

Segundo Statcounter (2018), citado por Casserly (2019), durante o período de janeiro de 2018 a janeiro de 2019, o *Android* representou 74,45% do mercado de smartphones mundial. Já no Brasil, a representação é ainda maior, cerca de 85,57% do setor (STATCOUNTER, 2019).

Assim, de modo a atingir o maior número de usuários brasileiros, definiu-se que a aplicação seria desenvolvida na tecnologia *Mobile* para o sistema operacional móvel *Android*. Na QUADRO 2, são apresentadas as tecnologias utilizadas para o desenvolvimento da aplicação *Mobile* definida.

QUADRO 2 – TECNOLOGIAS PROPOSTAS PARA O DESENVOLVIMENTO

Tecnologia	**Uso**
Linguagem Orientada a Objetos Java Netbeans 8.2 Hibernate 5.4 JPA 2.2 Glashfish 5.0 Postman7.30.1	Desenvolvimento *Webservice*
Linguagem de Programação *Javascript* *Android Studio* 3.6.1 React Native Cli 2.0.1 Node.js 12.16.1 NPM 6.13.4 Chocolatey 0.10.15 Phython 2.2.7	Desenvolvimento Aplicativo
Git 2.26.1.windows.1	Crontrole de versão
Projeto Padrão MVC	Desenvolvimento *Webservice*
Linguagem de Programação SQL MySQL 8.0.19	Desenvolvimento Banco de Dados

FONTE: Os Autores (2019).

3.4.1. Linguagem de Programação

Uma Linguagem de Programação é um conjunto de símbolos, como comandos, indicadores, caracteres, entre outros; e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador (BUFFONI, 2003).

No desenvolvimento deste trabalho, foram utilizadas as linguagens de programação *Javascript* e Java, sendo a primeira aplicada no desenvolvimento da aplicação *Android* (seção 3.5.3) e a última na implementação do *webservice* (seção 3.5.5). A seguir será realizada uma breve descrição sobre cada uma das linguagens supracitadas.

Segundo documentação da rede de desenvolvedores da Mozilla (MDN), o *Javascript* é uma linguagem de script orientada a objetos, multiplataforma, pequena e leve. Além disso, o principal diferencial da linguagem *Javascript* em relação a outras linguagens de programação, é a sua capacidade de rodar programas localmente, ou seja, do lado do cliente.

Já a linguagem de programação Java, segundo sua especificação, citada por Jandl Junior (2015, p.17), é uma linguagem de propósito geral, concorrente baseada em classes e orientada a objetos.

A plataforma de programação Java foi apresentada ao mercado em 1995 pela *Sun Microsystems*, como um ambiente completo de desenvolvimento e execução de programas que reúne um conjunto ímpar de facilidades: uma linguagem completamente orientada a objetos, robusta, muito portátil, que permite operação em rede, distribuição de aplicações e que incorpora diversas características voltadas à segurança (JANDL JUNIOR, 2015, p.17).

3.4.2. Sistema de controle de versão GIT

No desenvolvimento de qualquer projeto é essencial que exista uma organização, principalmente na área de tecnologia quando existe um código que múltiplos desenvolvedores trabalham ao mesmo tempo.

Uma das opções disponíveis no mercado é o *Git*, que de acordo com o site oficial é um software de linha de comando, que é instalado no sistema operacional e fica disponível para utilizar via terminal. Com o *Git* é possível criar

versões do sistema ou voltar para versões anterior de maneira simples, assim como analisar o histórico do que já foi feito.

No desenvolvimento da aplicação *UpMoney* foi utilizado o *Git Bash* para uso do Git – no site oficial essa opção é apresentada para uso no Windows - é uma aplicação Windows que emula um terminal Unix, oferecendo o mesmo conjunto de comandos, de forma a tornar a integração com o *Git* mais completa, dessa maneira é possível fazer uso de comandos como `cd` para mudar de pasta, `ls` para listar arquivos, entre outros. Os comandos do próprio *Git* – que podem ser encontrados na documentação oficial - também são disponibilizados, o que permite realizar um `git clone` para clonar um repositório remoto, `git add` para adicionar arquivos a serem comitados, `git commit` para comitar e `git push` para enviar essas alterações ao repositório.

3.4.3. Padrão de Projeto MVC e *Façade*

A arquitetura MVC (*Model, View, Controller*) é um paradigma para delimitar o código em segmentos funcionais, este padrão de projeto auxilia os desenvolvedores a construir aplicações separando seus principais componentes em camadas, seu principal propósito é fornecer um alto nível de reutilização dos modelos (FREEMAN, 2009, p. 420).

A *Model* (*Modelo*) contém a lógica e comunicação com os dados armazenados que serão visualizados na *View*. Estes dados podem ser armazenados em um banco de dados, arquivos XML (*Extensible Markup Language*) e afins que possam cumprir a funcionalidade. Além disso, apenas na *Model* ocorrem operações de alteração no banco de dados, como *create, read, update* e *delete* (CRUD) (F, 2013).

A *View (Visualização)* é a camada de apresentação, tende a ser controles usados para exibir e editar, sendo responsável por exibir todos os dados da aplicação para o usuário final, não importando a origem dos dados (FREEMAN, 2009, p. 420).

A *Controller (Controlador)* é responsável por administrar todo o fluxo da aplicação, é o que move a aplicação, a lógica trabalha com os dados de entrada da *View* e resolve qual operação utilizará da camada *Model,* logo a Controller faz

a mediação entre os estados mutáveis de cada camada (FREEMAN, 2009, p. 420).

O padrão *Façade* fornece uma interface unificada para um conjunto de interfaces em um subsistema, desta forma, *Façade* define uma interface de nível mais alto que simplifica o uso do subsistema (GAMMA et al, 2000, p.179).

Ainda, segundo Gamma et al (2000, p.179), estruturar um sistema em subsistemas ajuda a reduzir a complexidade, assim, objetiva-se minimizar a comunicação e as dependências entre subsistemas. Desta forma, é possível atingir esse objetivo ao utilizar *Façade*, pois esta fornece uma interface única e simplificada para os recursos e facilidades mais gerais de um subsistema (GAMMA et al, 2000, p.179).

Neste projeto, utilizou-se o padrão MVC, juntamente com Façade, no desenvolvimento da aplicação servidora. Desta forma, definiu-se como *View* a aplicação cliente, *Controller* a classes *Resources*, as quais recebem as requisições HTTP e *Model* as classes responsáveis pela comunicação com o SGDB. *Façade* foi utilizada para realizar a comunicação entre *Controller* e *Model*.

3.4.4. Aplicação *Mobile Android*

O *Android* é uma plataforma *open source*, ou seja, de código livre, para desenvolvimento de aplicações para *smartphone* e *tablets*. Esta plataforma é baseada no sistema operacional Linux e utiliza como linguagem de programação Java (OGLIO, 2013).

Desta forma, segundo Lecheta (2009), citado por Oglio (2013), o principal objetivo dessa plataforma é ser moderna e flexível, com o intuito de possibilitar o desenvolvimento de novos aplicativos de forma rápida e moderna, além de possibilitar a personalização das aplicações e componentes do seu sistema operacional.

Devido a flexibilidade que a plataforma *Android* proporciona, optou-se por utilizar o *framework React Native*, uma vez que este permite o desenvolvimento da aplicação na linguagem de programação *Javascript*.

Conforme escreveu Lopes (2016, p.1) “As plataformas nativamente oferecem a possibilidade de criar aplicativos. Usando o *Android* SDK (*Software*

Development kit) e a linguagem Java, podemos desenvolver para o sistema do Google".

Além disso, a maior parte das plataformas permite usar C++, que foi essencialmente pensado para desenvolvimento de jogos. Porém, as APIs (*Application Programming Interface*) e as bibliotecas sofrem recorrentes mudanças, desta forma, mesmo usando uma linguagem comum, é muito difícil escrever aplicações nativas multiplataforma (LOPES, 2016, p.1).

Ainda, segundo Silva (2018), um problema que a indústria enfrenta é contratar profissionais com conhecimento em ambas as plataformas, ou até mesmo, contratar um profissional para cada área de atuação, para criar aplicativos diferentes que, por consequência, exigem manutenção separada, podendo acarretar custos mais altos e a durações de projetos maiores, o que pode se tornar um problema de curto a longo prazo.

O objetivo do *React Native* é aumentar o aproveitamento do código entre as plataformas disponíveis hoje no mercado, ou seja, o mesmo código poderá ser parcialmente, ou até totalmente, reaproveitado em outras plataformas *Mobile*. Isso é possível, pois o *React Native* realiza a comunicação entre o *Javascript* e a linguagem Nativa (*Objective*-C/Java) em segundo plano, sendo responsável por abstrair e interpretar o código através da comunicação por meio de uma serialização, uma ponte assíncrona e em lote.

3.4.3.1 *React Native*

Antes de iniciar o desenvolvimento da aplicação final, realizou-se uma prova de conceito para testar as tecnologias e o escopo do projeto como um todo. Para essa amostra utilizou-se a ferramenta *Expo*, segundo o site oficial da ferramenta, Expo apps são aplicativos *React Native* que contém a Expo SDK. A SDK é uma biblioteca *Javascript* nativa que possui acesso a funcionalidades do sistema do dispositivo, como câmera, armazenamento local, entre outros hardwares. Essa ferramenta é uma alternativa que elimina a necessidade do uso de editores de texto como *XCode* ou *Android Studio*.

No entanto, para realizar o desenvolvimento do projeto final decidiu-se utilizar o *React Native Cli* devido à ausência de algumas funcionalidades no Expo Cli, portanto foi necessário instalar algumas dependências. De acordo com a

documentação oficial do *React Native*, recomenda-se que faça uso do *Chocolatey,* que no site da empresa descreve como sendo um gerenciador de pacotes, que busca reduzir a complexidade nas instalações no *Windows*. Utiliza-se esse gerenciador para executar as instalações das seguintes dependências:

- Node.js: De acordo com o site oficial, é um ambiente de execução *Javascript* assíncrono orientado a eventos, e indicado para servir como base de bibliotecas web ou framework;
- npm: *Node Package Manager*, ou em português Gerenciador de Pacotes do Node, tem como definição no seu site oficial como sendo um gerenciador de pacotes para o Node.js, que tem como intuito auxiliar desenvolvedores de *Javascript* a compartilhar de maneira facilitadas os módulos do código.
- *Python*: Na documentação oficial do *React Native* a razão pela qual é usado é porque o sistema de construção do *React Native* é 100% baseado nele. Na documentação é indicado também que se use a versão 2 desse software;
- JDK 8: *Java Development Kit*, no site da *Oracle* é disponibilizado o download e define-se como um conjunto de utilitários cuja a finalidade é a permissão para criação de programas para a plataforma Java.

Além dos softwares citados acima, para fazer uso do *React Native* fez-se necessário também a instalação do *Android Studio* – que de acordo com sua documentação oficial define-se como um ambiente de desenvolvimento integrado (IDE, na sigla em inglês), oficial para o desenvolvimento de apps para *Android* – nessa ferramenta fez-se uso dos emuladores disponibilizados no software.

Por fim, foi necessário realizar a instalação do *React Native Cli* que proporciona a opção de construir um projeto *React Native* por linhas de comandos, diferenciando-os de acordo com cada plataforma (iOS ou Android).

A seção seguinte expõe quais são as tecnologias utilizadas para criar o banco de dados do aplicativo *UpMoney*.

3.4.5. Banco de Dados

Segundo Date (2004, p.26), um banco de dados (BD) é um sistema de armazenamento de dados baseado em computador; isto é, um sistema cujo objetivo global é registrar e manter informação significativa para uma organização.

Portanto, um banco de dados é um depósito no qual vários dados podem formar coleções que se relacionam de forma a criar um sentido (informação). Informações podem ser utilizadas como fontes de pesquisa, sendo de suma importância para empresas na tomada de decisão.

Os dados geralmente são integrados e compartilhados. Entende-se integrados como sendo a unificação de diversos arquivos, que seriam distintos entre si, eliminando assim parcial ou totalmente qualquer redundância entre eles. Já por compartilhados entende-se que partes individuais de dados podem ser compartilhadas entre diversos usuários diferentes, significando que cada usuário pode ter acesso ao mesmo tempo à mesma parte do dado armazenado (DATE, 2004, p.26).

Desta forma, o banco de dados utilizado neste projeto foi obtido a partir da modelagem de dados.

Segundo Heuser (1998, p.5), um modelo de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados, em outras palavras, é a descrição formal da estrutura de um banco de dados.

De acordo com a intenção do modelador, um banco de dados pode ser modelado (descrito) há vários níveis de abstração (HEUSER, 1998, p.5). Neste trabalho, aplicou-se dois níveis de abstração de modelos de dados: modelo conceitual e modelo lógico.

Um modelo conceitual é uma descrição do banco de dados de forma independente de implementação em um SGBD (Sistema de Gerenciamento de Banco de Dados). O modelo conceitual registra quais dados podem aparecer no banco de dados, mas não registra como estes dados estão armazenados a nível de SGBD (HEUSER, 1998, p.5-p.6).

Já um modelo lógico, segundo Heuser (1998, p.6) é uma descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD. Assim, o modelo lógico é dependente do tipo particular de SGBD que está sendo usado.

Para obtenção dos modelos supracitados para este trabalho, utilizou-se o aplicativo *brModelo* versão 3.2 que é aplicativo gratuito e brasileiro destinado ao ensino da modelagem de bancos de dados relacional (BERNARDI, 2013).

Desta forma, com o auxílio do *brModelo*, utilizou-se as modelagens de banco de dados supracitadas e obteve-se o diagrama da modelagem conceitual, apresentado no APÊNDICE G e o diagrama lógico presente no APÊNDICE H.

No entanto, o banco de dados foi gerado automaticamente via programação em linguagem Java, utilizando a biblioteca *Hibernate* 5.4 (seção 3.4.5.1) com a especificação JPA 2.2.

Ao utilizar JPA, as classes Java que representam os dados a serem armazenados no SGBD são chamadas de Entidades. Estas são classes comuns que contêm apenas atributos e métodos *getters* (utilizados na recuperação de valor dos atributos) e *setters* (usados na alteração dos valores dos atributos).

O que torna a classe Java uma entidade é a presença de uma anotação do pacote javax.persistence, denominada *Entity*. Desta forma, o *Hibernate* associa a classe Java a uma tabela do banco de dados. Além disso, a própria implementação da especificação JPA se encarregará de criar uma tabela automaticamente no banco de dados relacional.

O Sistema de Gerenciamento de Dados utilizado neste projeto é o MySQL, o qual será descrito na próxima seção.

3.4.5.1. MySQL

O MySQL, segundo Milani (2006, p.22), é um servidor e gerenciador de banco de dados (SGBD) *open source* relacional, que utiliza a linguagem SQL (*Structured Query Language*) como base para manipular um banco de dados. Com ele é possível implementar as principais funcionalidades de um BD, conhecidas como, tabelas, *views*, funções, *stored procedures, triggers*, cursores entre outras para manutenção de um banco de dados robusto. Inicialmente foi projetado para trabalhar com aplicações de pequeno e médio porte, mas hoje atende também aplicações de grande porte.

Neste projeto, o MySQL foi utilizado o para armazenar, manipular e prover informações obtidas de um usuário por meio da aplicação, aproveitando

as funcionalidades existentes e implementando as melhores práticas com relação a manutenção dos dados.

Na seção seguinte será apresentado o *webservice* proposto para este projeto, o qual é responsável por realizar a integração entre o BD e a aplicação *Mobile*.

3.4.6. *Webservice*

Segundo Lecheta (2015, p.19), os *webservices* são usados como forma de integração e comunicação entre os sistemas, devido a diferença de infraestrutura, linguagem entre outras especificidades, a comunicação entre sistemas distintos pode se tornar algo complexo e custoso. Uma das grandes vantagens na construção de *webservices* é eles permitirem uma comunicação independente da linguagem de programação e padronizada, as comunicações são feitas em formato de chamadas, que podem enviar ou receber informações em diversos formatos, sendo que atualmente os mais populares são XML (*Extensible Markup Language*) e JSON (*JavaScript Object Notation*).

Atualmente, existem várias formas de criar *webservices,* porém as mais conhecidas e utilizadas no mercado são *webservices* em SOAP (*Simple Object Access Protocol*), ou REST (*Representational State Transfer*) (LECHETA, 2015, p.19).

Conforme descrito por SOAPUI ([2020]) servidores do tipo REST são mais flexíveis e mais fáceis de usar se comparados a servidores do tipo SOAP. Além disso, a curva de aprendizado de REST é menor, uma vez que faz uso de padrões fáceis como *swagger* e *penAPI Specification 3.0*. Ainda, REST permite o uso JSON, enquanto, SOAP só utiliza XML, o que torna o primeiro mais eficiente. REST, também, pode ser mais rápido pois não necessita processamentos extensivo (SOUPUI, [2020]).

Levando em consideração as informações apresentadas anteriormente, optou-se pelo desenvolvimento de um *webservice* do tipo REST com auxílio da especificação JAX-RS. O JAX-RS é a especificação de uma API Java que fornece recursos para obter e processar informações sobre o contexto de implementação do aplicativo e o contexto de requisições individuais (HADLEY; SANDOZ, 2008, p.25).

Existem diversas implementações JAX-RS para este trabalho foi utilizada *Jersey*. A estrutura *Jersey RESTful* é uma estrutura de código aberto, qualidade de produção, *framework* para desenvolver *RESTful Web Services* em Java que fornece suporte para APIs JAX-RS e serve como uma implementação de referência JAX-RS (JSR 311 e JSR 339) (ECLIPSE FOUNDATION, [2020?]). Ainda, segundo *Eclipse Foundation* ([2020?]), o *framework Jersey* fornece sua própria API que estende do kit de ferramentas JAX-RS com recursos e utilitários adicionais para simplificar ainda mais o serviço *RESTful* e o seu desenvolvimento.

A integração de dados entre *webservice* e banco de dados foi realizada por meio do *framework Hibernate* com a especificação JPA que serão apresentados a seguir.

3.4.6.1. *Hibernate e* JPA

O *Hibernate* é um *framework* para mapeamento objeto-relacional para linguagem Java. Sua finalidade é a persistência automática (e transparente) de objetos em uma aplicação Java para as tabelas em uma base de dados relacional, usando metadados que descrevem o mapeamento entre os objetos e a base de dados (BAUER e KING, 2005, p.25).

Ainda, segundo Bauer e King (2005, p.34), este framework utiliza um dialeto SQL próprio denominado HQL (*Hibernate Query Language*) que juntamente com a API Creteria produzem SQL (*Structured Query Language*) que independe do SGBD (*Data Base Management System)* utilizado, podendo ser configurado o dialeto no próprio Hibernate.

Esta é uma linguagem bastante parecida com SQL, no entanto, orientada a objetos permitindo, desta forma, o uso de herança, polimorfismo e encapsulamento.

Além disso, para realizar a conexão com o banco de dados utilizou-se a JPA (*Java Persistence API*). Esta consiste em uma especificação de interface de programação de aplicações Java a qual descreve o gerenciamento de dados relacionais, tanto em aplicações JAVA SE, como em aplicações Java EE. Além disso, esta especificação fornece um mapeamento objeto-relacional para

desenvolvedores Java, na questão de gerenciamento de dados relacionais em aplicativos Java (BAUER e KING, 2007, p.31).

Desta forma, o desenvolvimento foi realizado na linguagem de programação orientada a objetos Java por meio de um ambiente de desenvolvimento integrado, neste caso, *Netbeans 8.2*.

3.4.6.2. *Netbeans*

O *Netbeans* é um ambiente de desenvolvimento multiplataforma que auxilia na escrita, compilação, depuração e instalação de aplicações. Esta IDE (*Integrated Development Environment*) foi arquitetada em forma de uma estrutura reutilizável que visa simplificar o desenvolvimento e aumentar a produtividade, pois reúne em uma única aplicação todas estas funcionalidades (NETBEANS, [2020?]).

Para testar as funções desenvolvidas para *webservice* utilizou-se o *software Postman 7.30.1*. O *Postman* é uma plataforma de colaboração para desenvolvimento de APIs, a qual simplifica cada etapa de construção de uma API e otimiza a colaboração de forma a criar APIs melhores mais rapidamente (POSTMAN, 2020).

A conexão com o servidor foi realizada a partir de um *Dynamic Domain Name System* (DDNS).

3.4.6.3. *Dynamic Domain Name System*

Para melhor entender o conceito do DDNS, é preciso conhecer o que é um DNS comum, ou seja, um *Domain Name System*. A essência do DNS é a criação de um esquema hierárquico de atribuição de nomes, baseado no domínio, sua funcionalidade consiste em mapear endereços de IP (*Internet Protocol*), e seus respectivos domínios. Desta forma, os servidores DNS vinculam um rótulo acessível ao endereço real do IP daquela página (TANENBAUM e WETHERALL, 2011, p.384).

No entanto, esse serviço se torna ineficiente em situações em que o endereço IP varia, ou seja, no uso da maioria dos usuários e serviços. Desta forma, o DDNS demonstra sua funcionalidade, pois, ele atualiza constantemente

o histórico que correlaciona um IP a um DNS, o que garante a conectividade em todos os momentos. Assim, o DDNS contorna o problema atribuindo o endereço mais recente ao ponto de acesso.

Para garantir a segurança das informações ao acessar o servidor foi utilizado o *Token JWT* (*Json Web Token)*, o qual será apresentado a seguir.

3.4.6.4. *Token JWT*

O JWT é um padrão aberto (RFC-7519) de mercado que define uma maneira compacta e independente para transmitir informações com segurança entre partes como um objeto JSON (JWT, [201-]). Ainda, segundo JWT ([201-]), os JWTs podem ser assinados digitalmente por um algoritmo HMAC, ou um par de chaves pública/privada usando RSA ou ECDSA.

A estrutura de um *Token* JWT é formada por três partes, segundo JWT ([201-]):

- *Header*: trata-se de um objeto JSON codificado em *Base64Url* e o algoritmo de assinatura usado, como HMAC, SHA256 ou RSA.
- *Payload*: é um objeto JSON codificado em *Base64Url* que contém as *Claims* (informações) da entidade tratada, normalmente o usuário autenticado e dados adicionais;
- *Signature*: é a concatenação dos *hashes* gerados a partir das seções supracitadas usando *base64UrlEncode*, com uma chave secreta ou um certificado RSA, desta forma, garante-se a integridade do *Token*.

A próxima seção mostra como as metodologias ágeis foram empregadas na primeira etapa deste trabalho, na qual realizou-se a modelagem do aplicativo e, também, implementou-se uma prova de conceito com intuito de conhecer as tecnologias propostas para o desenvolvimento do aplicativo *UpMoney*.

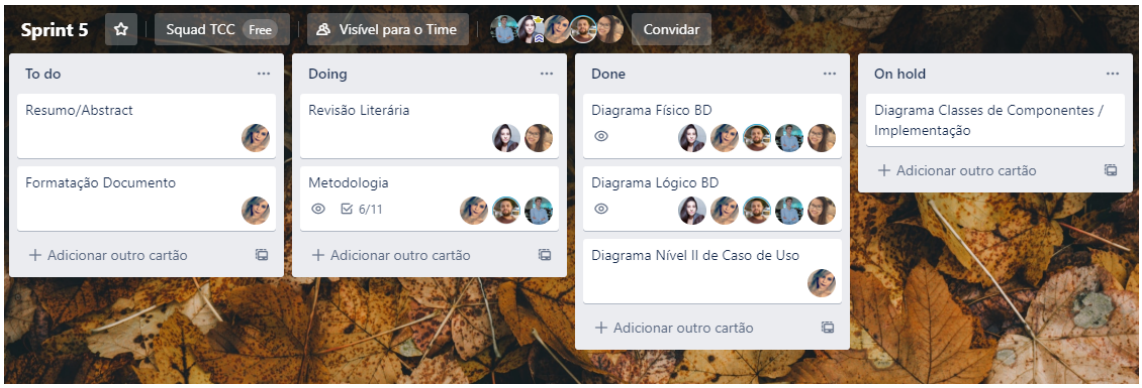
3.5. DESENVOLVIMENTO DO PROJETO

Ao aplicar a metodologia ágil *Scrum*, a primeira etapa deste trabalho foi dividida em sete *sprints*, sendo uma de uma semana e as demais com duração de duas semanas. Já na segunda etapa, utilizou-se 10 *sprints* de quinze dias.

Deste modo, obteve-se o controle das atividades realizadas em cada *sprint*, utilizou-se o quadro *Kanban.* Desta forma, este foi criado com a ferramenta Trello.

O Trello é uma ferramenta de colaboração que organiza seus projetos em quadros. Ele informa o que está sendo trabalhado, quem está trabalhando em quê, e onde algo está em um processo (TRELLO, 2015). Na Figura 10, é apresentado um quadro *Kanban* de uma *sprint* deste trabalho na ferramenta Trello.

FIGURA 10 – QUADRO KANBAN DA SPRINT 5



FONTE: Os Autores (2019).

Assim, utilizou-se o *Kanban* de modo a facilitar a implementação da metodologia *Scrum*, uma vez que a cada *planning meeting* realizada, as tarefas eram criadas no Trello, sendo estas atribuídas a um responsável (Seção 3.6) e um prazo definido para finalizá-las. O cronograma das *sprints* elaborado é apresentado no Diagrama de GANTT, situado no APÊNDICE I.

Na *planning meeting* seguinte, retomava-se os quadros no Trello para avaliar se as atividades foram executadas conforme planejado.

A seguir serão descritas as *sprints* realizadas na primeira fase deste projeto.

3.5.1. *Sprint* 1

O objetivo da primeira *sprint* do projeto foi definir os requisitos, apresentado no APÊNDICE A. Para isso, realizou-se um estudo de softwares semelhantes que estão inseridos no mercado atualmente, e comparou-se suas

principais funcionalidades, como é apresentado no QUADRO 1. Além disso, elaborou-se um diagrama inicial de caso de usos.

Em seguida, realizou-se uma reunião para discutir e definir os requisitos funcionais e não funcionais da aplicação. Por fim, foi desenvolvido o planejamento das *sprints* relativas à primeira parte do projeto.

3.5.2. *Sprint* 2

Na segunda *sprint*, corrigiu-se requisitos com base nos apontamentos da orientadora do projeto. Além disso, dedicou-se um tempo para realizar estudos sobre assuntos relativos ao mercado financeiro, os quais pudessem auxiliar na definição das funcionalidades do aplicativo, em especial, as funcionalidades relacionadas à meta.

Por fim, iniciou-se a elaboração do diagrama de classes, apresentado no APÊNDICE E, no entanto, optou-se por paralisá-lo devido a dúvidas, que posteriormente, seriam esclarecidas com a orientadora do projeto.

3.5.3. *Sprint* 3

Nesta *sprint*, os requisitos (APÊNDICE A) foram redefinidos, pois se identificou novas funcionalidades e retirou-se requisitos desnecessários. Além disso, replanejou-se as demais *sprints.*

O diagrama de casos de uso da aplicação, apresentado no APÊNDICE B, também, foi corrigido nesta etapa. E com base neste, iniciou-se a prototipação das telas do aplicativo e revisou-se e corrigiu-se o diagrama de classes. Na correção deste, instanciou-se objetos das classes criadas para identificar os relacionamentos entre elas. O diagrama de classes pode ser visualizado no APÊNDICE E.

Em paralelo, iniciou-se o documento do projeto, sendo desenvolvidos os seguintes tópicos: introdução, problema, justificativa, objetivo geral e objetivos específicos.

3.5.4. *Sprint* 4

A quarta *sprint* foi iniciada com a revisão e correção das atividades desenvolvidas na *sprint* anterior. Além disso, a prototipação de telas foi finalizada e a partir desta, as especificações de caso de usos foram elaboradas, estas podem ser visualizadas no APÊNDICE D.

A partir das especificações de caso de usos e dos diagramas de classe (APÊNDICE E), iniciou-se o desenvolvimento dos diagramas de sequência apresentado no APÊNDICE F.

Em paralelo, o padrão de projeto foi definido e um estudo da ferramenta *Power BI* foi realizado a fim de testar a viabilidade de seu uso neste projeto para a elaboração de gráficos. Após este estudo, o uso da ferramenta foi descartado.

3.5.5. *Sprint* 5

Nesta etapa do processo, finalizou-se os diagramas de sequência, APÊNDICE F, e então, desenvolveu-se o diagrama de caso de usos nível II (APÊNDICE C), o qual mostra o direcionamento das páginas por meio de *includes* e *extends*.

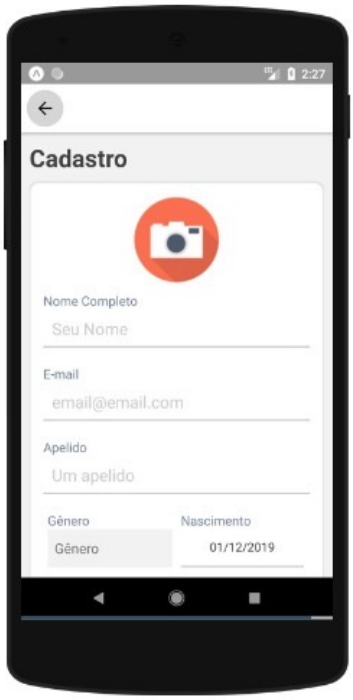
Em paralelo às atividades supracitadas, acrescentou-se ao documento do projeto alguns tópicos do capítulo de Materiais e Métodos, além de revisar e acrescentar informações no capítulo de Fundamentação Teórica.

3.5.6. *Sprint* 6

Na sexta *sprint*, iniciou-se o desenvolvimento da prova de conceito (POC), ou seja, o desenvolvimento de uma funcionalidade do aplicativo proposto. A funcionalidade selecionada foi cadastro de usuário.

Desta forma, implementou-se a tela de cadastro de usuário, a qual é apresentada na FIGURA 11.

FIGURA 11 – PROVA DE CONCEITO: CADASTRO DE USUÁRIO



FONTE: Os Autores (2019).

Além da tela, desenvolveu-se o *webservice*, o qual é responsável por realizar a integração da aplicação *Android* com o banco de dados. Nesta etapa, optou-se por criar um banco apenas com as tabelas referentes aos objetos Usuário e Conta, devido à complexidade de desenvolvimento de funções que consomem tabelas com relacionamentos entre elas. Além disso, desenvolveu-se uma pequena aplicação *Android* com framework *React-Native* para empregar testes de conexão com o banco de dados. Desta forma, realizou-se diversos testes para conexão entre banco de dados, *webservice* e a aplicação *Android*, obtendo-se sucesso.

3.5.7. *Sprint* 7

Na última *sprint* da primeira parte deste projeto, realizou-se a integração entre a tela e o *webservice* desenvolvidos na etapa anterior.

Além disso, realizou-se o planejamento das *sprints* da segunda parte do projeto, na qual o foco foi o desenvolvimento do aplicativo.

Após obter a função de cadastro de usuário, desenvolveu-se a função Fazer *Login*, no entanto, essa funcionalidade foi desenvolvida de forma

simplificada, ou seja, apenas com validações de existência de usuário no banco de dados e se a senha confere com a senha cadastrada.

Nesta etapa foram desenvolvidas as funcionalidades para realizar as requisições do tipo GET e POST ao *webservice*. No entanto, encontrou-se diversas dificuldades com a infraestrutura proposta, uma delas relacionada a utilização do *Expo Snack* no ambiente *web*, identificou-se que a requisição do tipo POST retornava um erro relacionado ao CORS (Compartilhamento de recursos com origens diferentes), onde há uma requisição *cross-origin* HTTP ao solicitar um recurso que tenha uma origem diferente (domínio, protocolo e porta) da sua própria origem. Esta situação inicialmente foi contornada utilizando o emulador disponibilizado pelo *Expo*, em um dispositivo *Android* conectado à mesma rede e realizando as requisições via IP Local, desta forma foi possível enviar e obter respostas do *webservice*.

A próxima seção apresenta o desenvolvimento da segunda etapa deste projeto, ou seja, o desenvolvimento do aplicativo proposto.

3.6. DESENVOLVIMENTO DO APLICATIVO

Nesta seção, são apresentadas as *sprints* realizadas na segunda etapa deste projeto, na qual foi desenvolvida a aplicação. O desenvolvimento foi baseado no diagrama de caso de usos situado no APÊNDICE C. Desta forma, o desenvolvimento da aplicação foi dividido em 10 *sprints* de quinze dias cada, o cronograma destas pode ser visualizado no diagrama de GANTT apresentado no APÊNDICE I.

É importante ressaltar que para cada funcionalidade implementada nas *sprints* seguintes, foram realizados testes para avaliar seu funcionamento. Além disso, ao iniciar uma nova *sprint*, dedicou-se tempo para possíveis correções da *sprint* anterior.

3.6.1. *Sprint* 8

Por se tratar de uma tecnologia nova para os desenvolvedores deste trabalho, reservou-se esta *sprint* para estudar a linguagem de *Javascript* aplicada ao *framework React Native.* Além disso, preparou-se o ambiente para

o desenvolvimento, ou seja, instalou-se os programas necessários com suas devidas configurações, descritos na seção 3.5.3.1 *React Native*. Como a parte do front-end de *Login* e Cadastro foram criados na prova de conceito, apesar de terem sido feitas na ferramenta *Expo,* o código de estilização pôde ser reutilizado e adaptado para uso no projeto final nessa *sprint*.

Optou-se por construir o servidor da aplicação em um computador exclusivo. Para isto, realizou-se a liberação do acesso externo ao modem, desta forma, configurou-se a porta externa 8082 para 8080, na qual encontra-se o servidor na rede interna e, também, a porta 8083 para a 3306 onde está o MySQL.

No entanto, encontrou-se dificuldade em utilizar um protocolo de internet público, pois este muda conforme a conexão com o provedor é realizada. Desta forma, encontrou-se como solução o uso do DDNS (*Dynamic Domain Name System*), o qual converte domínio para IP (*Internet Protocol*). Assim, utilizou-se o No-IP para contornar este problema, pois este fornece um domínio e uma ferramenta para ser instalada no servidor, a qual monitora e envia o endereço do IP público para o servidor DNS do No-IP, o que garante o acesso ao *webservice* desenvolvido pelo domínio.

3.6.2. *Sprint* 9

No planejamento das *sprints* definiu-se que primeiramente iria ser criado o *front-end* das telas, em paralelo com o desenvolvimento dos métodos necessários no servidor, para que a integração entre front e servidor pudesse ser executada em seguida.

3.6.2.1. Desenvolvimento Servidor

Na nona *sprint* foram implementadas as funções para fazer *login* e cadastro de novo usuário no servidor. Primeiramente, implementou-se a classe *model* que consiste em um POJO com as anotações usadas pelo framework *Hibernate* que permitem relacionar os objetos as tabelas do banco de dados. Desta forma, foi possível implementar, também, a classe DAO que consiste em métodos que utilizam a API JPA para realizar consultas, inserções, atualizações e exclusões no banco de dados. Então, de modo a organizar o código, e para

seguir as orientações do modelo MVC, desenvolveu-se uma classe intermediária que apenas acessa os métodos DAO por meio de um objeto, essas classes foram denominadas *Facade*. Por fim, implementou-se as classes *Resource* as quais utilizam JAX-RS e realizam a conexão com a aplicação *Android*.

Além de implementar as funções relacionadas ao usuário, estudou-se como implementar um *Token* JWT para acesso de funcionalidades que necessitam de controle de acesso.

3.6.2.2. Desenvolvimento *Front-end*

Na parte de *front-end,* após os estudos e a criação das telas de Login e Cadastro na 8ª *Sprint* definiu-se que seria utilizado o *Formik (2020)*, uma biblioteca que facilita o manuseio de formulários em *React Native*, juntamente com o NPMJS (2020), um pacote que facilita a validação nos formulários através de *schemas*, ele é basicamente cria um objeto formatado que se parece com o esquema pretendido para um objeto e em seguida, usa-se as funções do utilitário Yup para verificar se os objetos de dados e o esquema são correspondentes, consequentemente validando-os.

Tendo o *front-end* e a parte do servidor finalizados para *Login* e Cadastro, iniciou-se o desenvolvimento da integração dessas telas. No *Login* utilizou-se o *AsyncStorage,* uma API nativa do *React Native*, utilizada para armazenar dados persistentes no dispositivo. É um sistema de armazenamento de valor-chave não criptografado, assíncrono, persistente e global para o aplicativo de acordo com o site oficial do *React Native*. Dessa forma, o usuário ao logar no aplicativo na primeira vez, teria seus dados retidos dispensando a requisição destes novamente ao retornar ao app.

Para a parte de comunicação com o servidor, optou-se por utilizar o pacote *Axios (2020)* do React Native, é um cliente HTTP, que funciona tanto no browser quanto em node.js. A biblioteca é basicamente uma API que sabe interagir tanto com XMLHttpRequest quanto com a interface http do node.

3.6.3. *Sprint* 10

Nessa etapa necessitou-se realizar adaptações nas integrações realizadas na *sprint* anterior, além dos novos desenvolvimentos.

3.6.3.1. Desenvolvimento Servidor

Após realizar os estudos necessários, deu-se início a implementação das classes necessárias para a utilização de um *Token* JWT. O *Token* JWT consiste em código em Base64 que é armazenado no cabeçalho da requisição HTTP (BOSCARINO, 2019). Desta forma, ao executar a funcionalidade de *login,* o servidor passou a gerar o *Token* e enviá-lo no cabeçalho da requisição para o aplicativo. Assim, todas as funcionalidades com exceção do *login* e cadastro de novo usuário passaram a realizar a autenticação do usuário via *Token*.

Além disso, optou-se por implementar o banco de dados gradativamente conforme fossem implementadas as classes modelos no servidor. Assim, com base na modelagem do diagrama de classes (APÊNDICE E) e com o apoio da modelagem do banco de dados (APÊNDICE G e APÊNDICE I), logo que uma classe *model* é implementada, sua respectiva tabela é criada no banco de dados por meio da API JPA.

Ainda, no servidor, implementou-se uma classe para tratamento de exceções para que fossem encaminhadas mensagens de erro, assim como o código de requisição HTTP.

3.6.3.2. Desenvolvimento *Front-end*

Essa etapa iniciou-se com os ajustes na tela de Cadastro e *Login* para a integração do *Token,* ao se cadastrar o servidor cria um código de autenticação para o usuário, assim toda vez que o usuário se loga no aplicativo, o servidor envia esse *Token* pelo *header* da requisição e ao ser recepcionado no *client-side* é armazenado no *header* da *api* criada pelo axios, dessa forma, a cada requisição que o usuário realizar, o servidor irá verificar se o *Token* continua ativo, permitindo a interação com os dados ou não.

Além dessa alteração de autenticação, nessa etapa também foram desenvolvidos a estilização das telas de Visualizar Contas e Inserir Conta.

3.6.4. *Sprint* 11

A seguir serão descritas as atividades desenvolvidas no desenvolvimento do servidor e, posteriormente, no aplicativo.

3.6.4.1. Desenvolvimento Servidor

Na 11ª *sprint*, desenvolveu-se, no *webservice*, as funcionalidades de manter perfil de usuário, ou seja, a função de alterar usuário, obter o usuário pelo e-mail cadastrado, lembrando que o e-mail deve ser único. Além disso, implementou-se validações para garantir a integridade dos dados. Optou-se por implementar a função de exclusão de usuário após o desenvolvimento da aplicação completa, pois, ao excluir um perfil de usuário, deve-se excluir todos os dados a ele relacionado.

Além disso, foram implementadas as funcionalidades para inserir, alterar, excluir e consultar uma conta, seguindo a mesma sequência utilizada na *sprint* 9 (seção 3.6.2).

3.6.4.2. Desenvolvimento *Front-end*

Para o *front-end* continuou-se a integração da tela de Cadastro à medida que novos métodos eram finalizados pela parte do servidor. Nesse momento, integrou-se a opção de alteração dos dados do perfil de usuário.

Em paralelo iniciou-se também a integração da tela de Conta e Visualizar Contas, na tela de Visualização de Contas foi necessário realizar certas adaptações para que a tela comportasse da maneira correta. Um exemplo foi a necessidade de adicionar um botão de atualização para que os dados da conta selecionada fossem alterados, conforme FIGURA 12. Essa adaptação se fez necessária pois ao utilizar o componente “carrossel” para dar a usabilidade de *slides* de *cards* de contas ao usuário, não se conseguia captar a identificação da conta apenas por esse movimento, ou seja, a cada vez que se deslizava para a esquerda ou direita o id da conta permanecia o mesmo.

Esse problema resolveu-se, portanto, com a inclusão desse botão de atualizar em cada cartão, ao clicar o objeto conta era recuperado e permite a interação com os dados vinculados a ela.

FIGURA 12 – ADAPTAÇÃO DE FUNCIONALIDADE (TELA DE CONTAS)

FONTE: Os Autores (2020).

Além das telas relacionadas às Contas, iniciou-se também o desenvolvimento das telas de Manter Despesa, Manter Receita, Manter Subcategoria Despesa e Manter Categorias Receita.

3.6.5. *Sprint* 12

A seguir serão descritas as atividades desenvolvidas desta *sprint*.

3.6.5.1. Desenvolvimento Servidor

Nesta *sprint*, desenvolveu-se os *models* para subcategoria, categoria despesa e categoria receita. Por se tratar do mesmo tema, optou-se por criar uma única classe DAO, *Facade* e *Resource* para estas. Desta forma, focou-se

em implementar todos os métodos relacionados a estas funcionalidades durante esse período.

Além disso, iniciou-se o desenvolvimento das classes POJO de receita e de despesa.

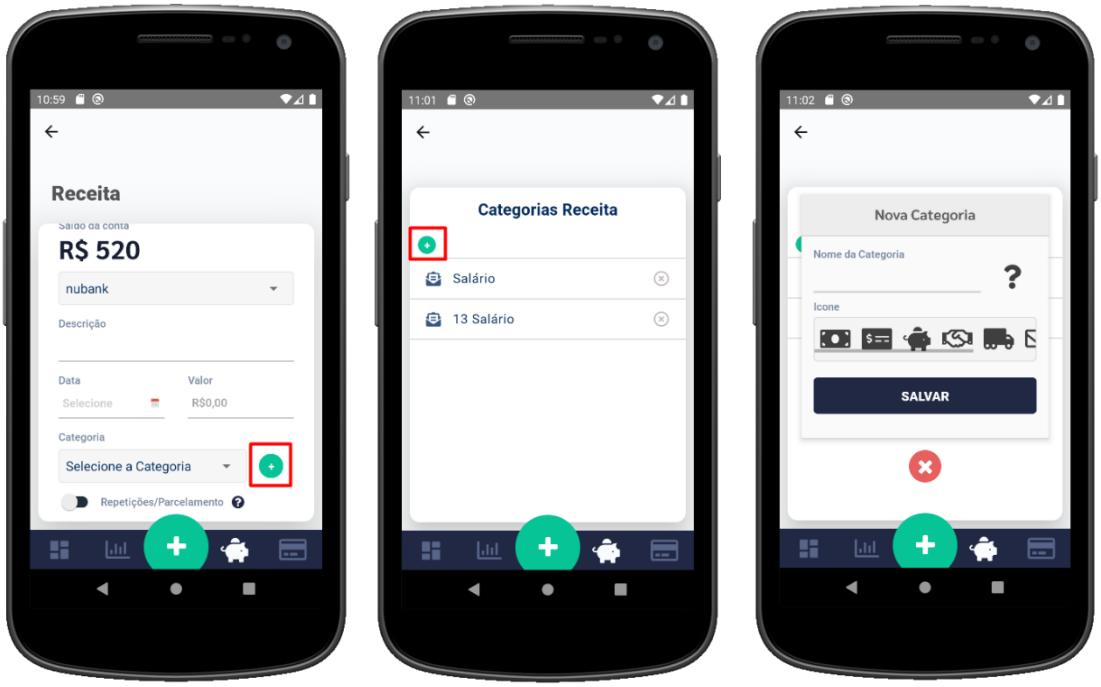
3.6.5.2. Desenvolvimento *Front-end*

Na parte de *front-end*, assim que as telas de Subcategorias de Despesas e Categorias de Receitas foram finalizadas, juntamente com os métodos respectivos do servidor, iniciou-se a integração das funcionalidades relacionadas a essas telas (exclusão, inserção, exclusão e alteração).

As telas de subcategorias apresentam os dados das diferentes categorias que o aplicativo pode possuir para melhor organização das despesas e receitas do usuário, de acordo com o seu gosto (Exemplo: as mensalidades de um usuário com academia, podem todos ser vinculados à uma categoria “Academia”).

Apesar das telas e dados de receitas e despesas serem praticamente iguais, existem algumas diferenças que valem ser ressaltadas. O caso de Receitas é mais simples, para as receitas podem ser criadas várias categorias para agrupá-las e diferenciá-las entre si, conforme FIGURA 13.

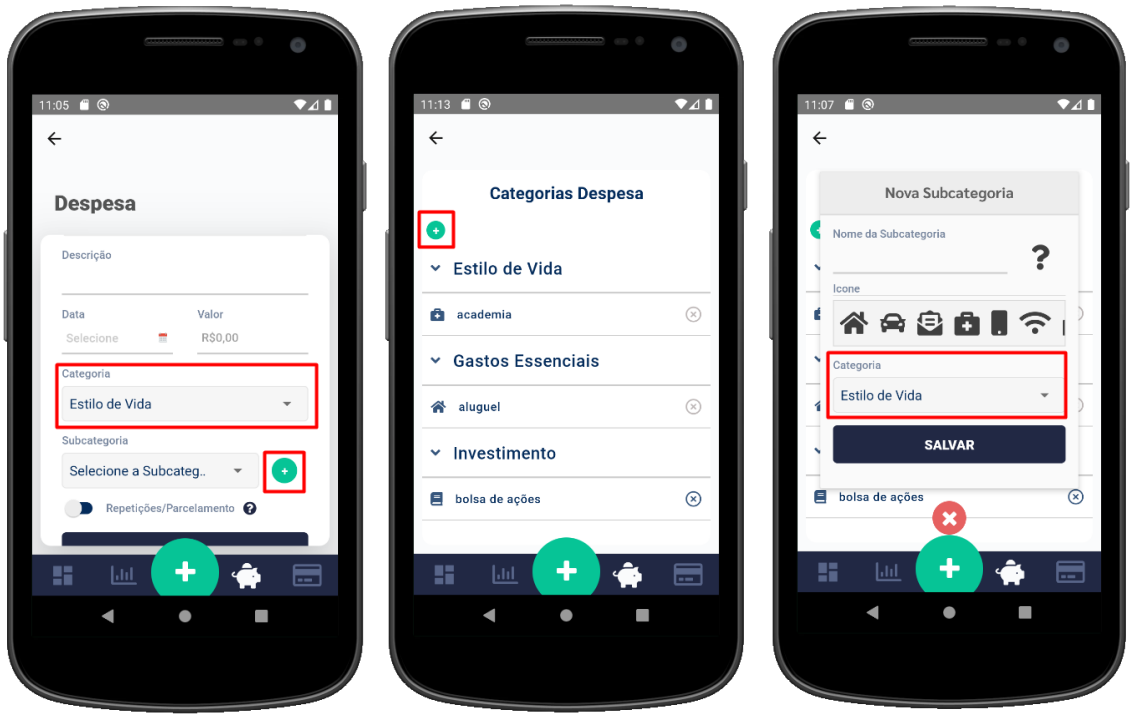
FIGURA 13 – TELAS DE ADICIONAR NOVAS CATEGORIAS PARA RECEITAS



FONTE: Os Autores (2020).

No entanto, em Despesas, existem 3 categorias bases, que são inalteráveis e trazidas por *default* pela aplicação, sendo elas: Gastos Essenciais, Estilo de Vida e Investimento. Essas 3 categorias bases são essenciais para o controle das despesas do usuário, pois é por meio delas que conseguimos aplicar a regra dos 50/35/15, no qual o escopo do nosso projeto é construído. Portanto, nesse caso, o usuário pode criar novas subcategorias para organizar seus gastos, escolhendo abaixo de qual categoria base irá ficar, conforme FIGURA 14.

FIGURA 14 – TELAS DE ADICIONAR NOVAS SUBCATEGORIAS PARA DESPESAS



FONTE: Os Autores (2020).

3.6.6. *Sprint* 13

A seguir serão listadas as atividades desenvolvidas neste período.

3.6.6.1. Desenvolvimento Servidor

Na décima terceira *sprint*, prosseguiu-se com o desenvolvimento das funcionalidades referentes a Despesa e Receita no servidor, assim, como em categoria, agrupou-se as funcionalidades nas classes DAO e *Facade* denominando-as como *LancamentoDAO* e *LancamentoFacade*. No entanto, no desenvolvimento do *Resource*, manteve-se a divisão, criando assim uma classe *Resource* para despesa e uma para receita.

O desenvolvimento para as funções de cadastro e alteração de despesa e receita levou mais tempo que as demais funcionalidades, uma vez que havia a opção de inserções com repetição.

3.6.6.2. Desenvolvimento *Front-end*

Nessa *sprint* iniciou-se o desenvolvimento das telas de Remanejar Gastos e Análise de Gastos, apenas a estilização. Na parte de integração finalizou-se as telas de Despesas e Receitas, integrando as funcionalidades de inserção, alteração e exclusão das mesmas.

A funcionalidade de repetição ou parcelamento presente nas duas telas foi um desafio para a parte de *front-end*, pois foi preciso pensar em uma maneira de separar a chamada de diferentes métodos em um único componente, portanto optou-se pelo uso do *switch*, tanto para conseguir diferenciar chamadas, quanto para mostrar diferentes campos caso acionados.

3.6.7. *Sprint* 14

Nos itens seguintes serão apresentadas as atividades desenvolvidas nesta etapa.

3.6.7.1. Desenvolvimento Servidor

Nesta *sprint,* no *webservice*, trabalhou-se nas funções para a parte meta de despesa, essas classes foram denominadas de Limite, sendo criadas classes POJO para limite de categoria despesa e limite de subcategoria. Assim como em categoria, agrupou-se as classes DAO, *Facade* e *Resource*.

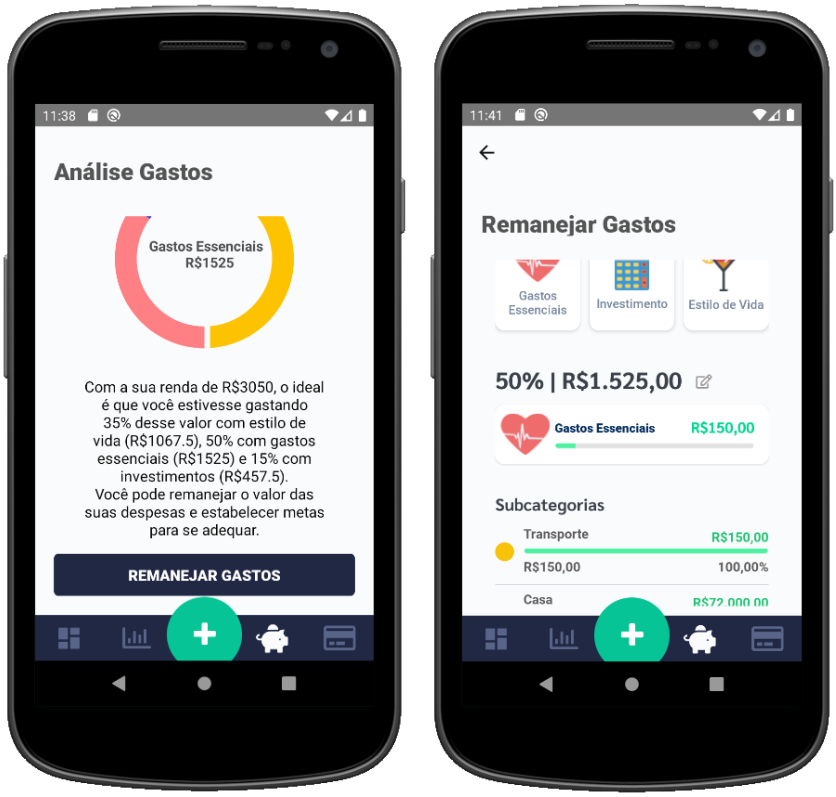
3.6.7.2. Desenvolvimento *Front-end*

Na parte de *front-end,* iniciou-se a estilização das telas de Relatórios em paralelo com a integração de análise e remanejamento de gastos.

Como a ideia principal do projeto se baseia na vida financeira saudável, tendo como base a regra 50/35/15, era muito importante que houvesse um retorno de como anda a distribuição dos gastos do usuário. Portanto, antes de entrar na tela de remanejar gastos, onde existe a possibilidade de impor e controlar as metas para cada subcategoria de despesa, é apresentado uma tela de análise (FIGURA 15). Nesse momento, o aplicativo mostra quanto o usuário

está gastando atualmente em cada categoria base e quanto deveria estar gastando.

FIGURA 15 – TELAS DE ANÁLISE E REMANEJAR GASTOS



FONTE: Os Autores (2020).

3.6.8. *Sprint 15*

Nesta seção serão descritas as atividades desenvolvidas na sprint 15.

3.6.8.1. Desenvolvimento Servidor

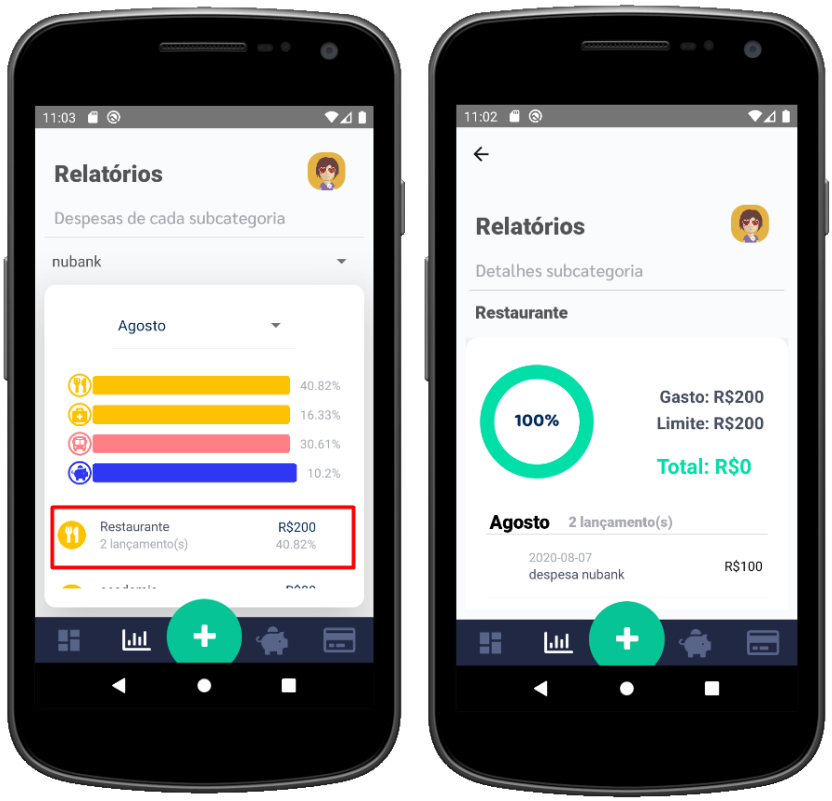
Na *sprint* 15, com relação ao servidor, focou-se em desenvolver as funcionalidades que atendessem os relatórios apresentados no aplicativo. Nesta etapa houve a necessidade de alterar a versão inicialmente usada do *Hibernate* (versão 4.3) para a versão 5.4, uma vez que aquela já não atendia algumas consultas necessárias, com isso, também foi necessário atualizar o *Netbeans* 8.0.1 para a versão 8.2 e o *glashfish* 4.0 para o 5.0. Desta forma, houve a necessidade de estender o desenvolvimento destas funcionalidades para a *sprint* 16.

3.6.8.2. Desenvolvimento *Front*-end

Nessa sprint, para a parte de *front-end*, iniciou-se o visual da tela de Recuperar Senha e a parte de integração dos Relatórios, a dificuldade nessa tela foi conseguir apresentar o máximo de dados relevantes, de forma concisa e de fácil compreensão. Como o foco do aplicativo é em cima das despesas, os relatórios foram gerados apenas em cima delas para que o usuário conseguisse visualizar e controlar em um só lugar todas os seus gastos.

Utilizou-se o recurso de cores, para separar as subcategorias nas 3 categorias bases principais, detalhando também a quantidade de lançamentos realizados dentro de cada subcategoria, conforme FIGURA 16.

FIGURA 16 – TELAS DE RELATÓRIOS



FONTE: Os Autores (2020).

3.6.9. *Sprint* 16

A seguir serão apresentadas as atividades realizadas no desenvolvimento do servidor da aplicação neste período. E, logo em seguida, as atividades envolvidas na programação do aplicativo *Android*.

3.6.9.1. Desenvolvimento Servidor

Nesta *sprint*, finalizou-se os métodos referentes a funcionalidades referentes ao relatório no servidor. Além disso, desenvolveu-se a função para recuperar senha e, também, corrigiu-se alguns erros encontrados no desenvolvimento do aplicativo.

Além disso, inicializou-se a correção e escrita do documento do projeto.

3.6.9.2. Desenvolvimento *Front*-end

Ao chegar nas sprints finais do desenvolvimento o a última tela de integração foi o Recuperar Senha, a parte do *front-end* foi simples, pois a complexidade ficou na parte do servidor. Portanto nesse momento iniciou-se o levantamento e a organização de todos os bugs que foram encontrados no aplicativo, assim como aprimoramentos na usabilidade, visual e fluxo entre telas.

3.6.10. *Sprint* 17

Por fim, na última *sprint*, no *webservice*, implementou-se a função de exclusão do usuário, assim como, desenvolveu-se funcionalidades que atendessem a escolha de um avatar pelo usuário. Além disso, focou-se em testar as funcionalidades implementadas e realizar correções quando necessárias. Além disso, finalizou-se documento do projeto.

Na próxima seção é apresenta a infraestrutura de desenvolvimento deste projeto.

3.7. INFRAESTRUTURA DE DESENVOLVIMENTO

Para o desenvolvimento desse projeto foram usados seis notebooks que possuem as seguintes configurações:

Notebook *1:*

- Nome do Computador: AmandaZ
- Proprietário (a): Amanda dos Santos Zanon
- Fabricante: Asus
- Sistema Operacional: *Windows* 10 home
- Memória RAM: 8 GB
- Processador: Intel(R) *Core* i7-7500U 2.90 GHz
- Espaço de armazenamento: 1TB

Notebook 2:

- Nome do Computador: *Macbook Air*
- Proprietário (a): Bruna Fonseca
- Fabricante: *Apple*
- Sistema Operacional: *masOS Catalina*
- Memória RAM: 8 BG
- Processador: 1,8 GHz Intel Core i5 *Dual-Core*
- Espaço de armazenamento: 128 GB

Notebook 3:

- Nome do Computador: Lenovo Yoga 520
- Proprietário (a): Cleverson
- Fabricante: Lenovo
- Sistema Operacional: Linux Deepin 15.11
- Memória RAM: 8 GB
- Processador: *Intel Core* i5 7200u
- Espaço de armazenamento: 1TB

Notebook 4:

- Nome do Computador: Acer E5-571G-760Q

- Proprietário (a): Fernando Perin
- Fabricante: Acer
- Sistema Operacional: *Windows* 8.1
- Memória RAM: 8 BG
- Processador: Intel i7-5500U
- Espaço de armazenamento: 1TB

Notebook 5:

- Nome do Computador: Lenovo Ideapad S145 Intel Core i7
- Proprietário (a): Lígia Nakazato
- Fabricante: Lenovo
- Sistema Operacional: *Windows* 10 home
- Memória RAM: expansível - Até 20GB (4GB soldado + 16GB *slot* DDR4 2400MHz)
- Processador: *Intel Core* i7-8565U
- Espaço de armazenamento: 1TB

Notebook 6:

- Nome do Computador: Lenovo Ideapad u410
- Proprietário (a): Servidor
- Fabricante: Lenovo
- Sistema Operacional: *Windows* 10
- Memória RAM: 8 GB
- Processador: *Intel Core* i7-3537U 3.1 GHz
- Espaço de armazenamento: 1TB

A seção seguinte apresenta a distribuição de tarefas para os responsáveis por este projeto.

3.8. ATRIBUIÇÃO DAS RESPONSABILIDADES

As atividades para a implementação deste projeto foram estruturadas de acordo com entregáveis que foram subdivididos em tarefas.

Após a estruturação das tarefas, os prazos de execução e seus responsáveis foram atribuídos, o cronograma de desenvolvimento do projeto, assim como, os responsáveis por cada atividade podem ser visualizados no gráfico de GANTT (APÊNDICE I).

4. APRESENTAÇÃO DO SOFTWARE

Nas próximas seções será detalhada a arquitetura do projeto, assim como as funcionalidades de cada tela.

4.1. ARQUITETURA

Nesta seção serão apresentadas as arquiteturas definidas para o servidor e para o aplicativo *Android*.

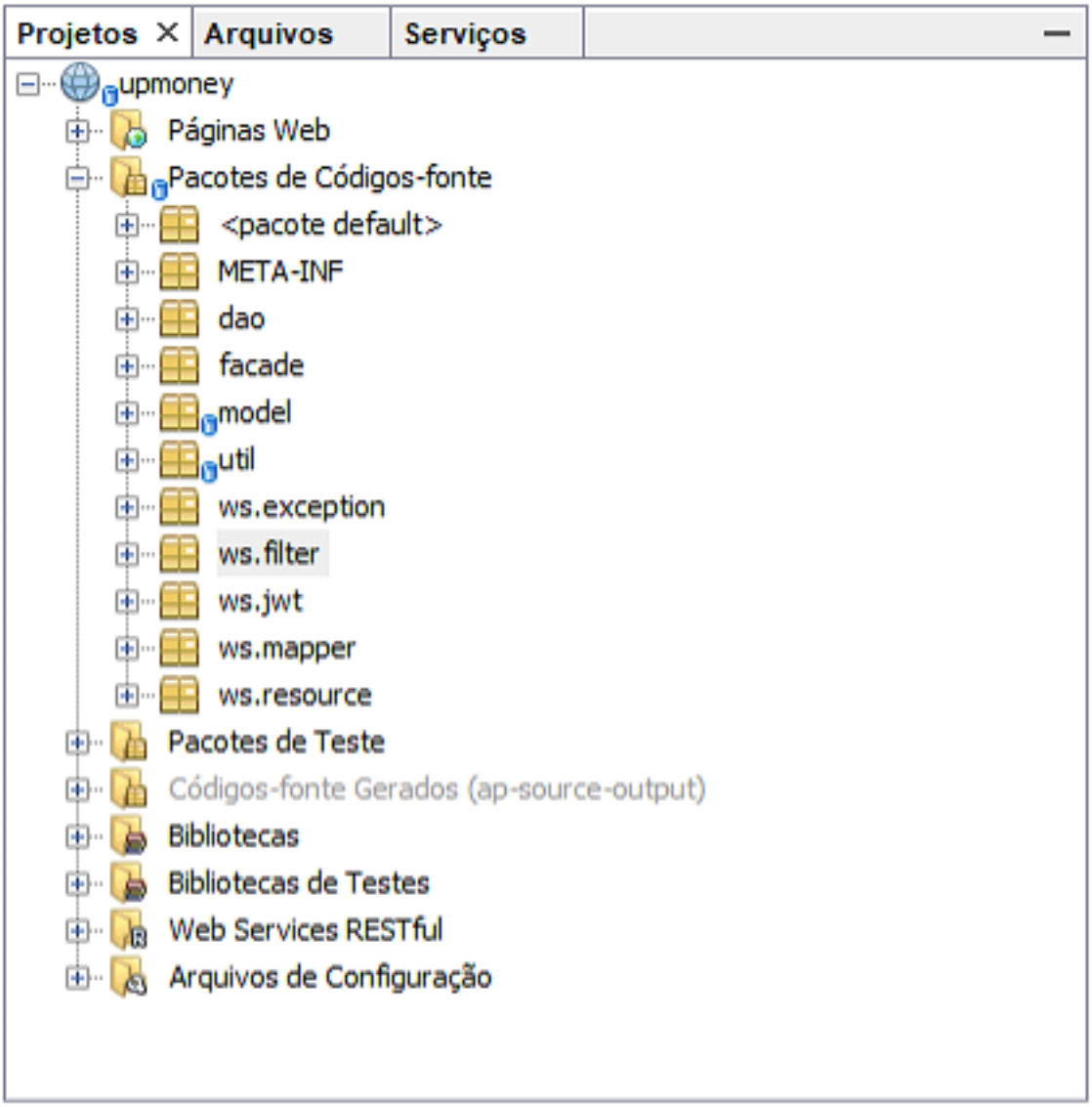
4.1.1. *Webservice*

Como apresentado na seção 3.5.5, o *webservice* realiza a integração entre a aplicação e o banco de dados. Desta forma, definiu-se o desenvolvimento de um *webservice* REST, utilizando a especificação JAX-RS para realizar a conexão da aplicação servidora com a aplicação cliente, neste caso, o aplicativo *Android*.

Já para realizar a conexão do servidor em questão com o SGDB, responsável pelo armazenamento das informações enviadas pela aplicação cliente, optou-se pelo uso do *framework Hibernate* juntamente com a API JPA de persistência de dados. Desta forma, foi possível gerar o banco de dados de forma automática a partir das classes *entities* utilizando JPA.

O projeto foi organizado no padrão MVC (seção 3.4.2) e Façade, deste modo, entende-se como *View* a aplicação *Android*, *Controller* as classes *Resources*, que são as classes que recebem as requisições da aplicação cliente e *Model* que são as classes *DAO* responsáveis por realizar a integração com SGDB, sendo possível consultas, inserções, alterações e exclusões nos dados armazenados no banco de dados. Em conjunto, com a utilização do padrão MVC, optou-se por aplicar o padrão *Façade*, assim, criou-se classes que intermediam o relacionamento entre *Controller* e *Model* formadas por métodos que chamavam os métodos das classes *DAO*. Na FIGURA 17, é apresentada a estrutura de pacotes da aplicação servidora.

FIGURA 17 – ARQUITETURA WEBSERVICE



FONTE: Os Autores (2020).

Como é possível visualizar na FIGURA 17, além das estruturas MVC e *Façade* supracitadas, criaram-se mais cinco pacotes auxiliares para garantir o funcionamento esperado pelo *webservice*.

No pacote *util*, foram criadas as classes *AES*, *JPAUtil*, *JavaMail* e *PassawordGenerator*. Na primeira, é realizada a criptografia da senha do usuário recebida da aplicação cliente. Já a *JPAUtil* é responsável por gerar o *EntityManager*, responsável este pelo gerenciamento das *entities*, ou seja, por meio dele é possível controlar o ciclo de vida das *entities*, operações de sincronização com a base de dados, consultar entidades e outras. A classe *JavaMail* contém configurações para o envio de um e-mail na funcionalidade de

recuperação de senha. Por fim, a classe *PasswordGeneration* é responsável por gerar uma senha provisória, a qual será enviada para *e-mail* do usuário caso este selecione a opção recuperar senha.

Além disso, implementou-se os pacotes *exception* e *mapper* com classes que realizam o tratamento de exceções geradas no *webservice*, de modo a enviar para a aplicação cliente o número do código HTTP gerado na resposta a requisição, assim como uma mensagem com a exceção gerada

O pacote *jwt* contém classes responsáveis pela criação, manutenção e utilização do *Token JWT* para garantir a segurança na troca de informações entre aplicação cliente e aplicação servidora.

Por fim, criou-se um pacote para armazenar as classes de filtro do *webservice*, ou seja, aquelas que serão executadas assim que a requisição chegar ao servidor. Foram criadas duas classes com filtros, *CorsFilter* e *JwtAuthenticationFilter*. A primeira foi implementada para permitir o acesso ao servidor de domínios externos, uma vez que se optou pela implantação da aplicação servidora em um computador exclusivo para ela. Já na última, verifica-se a existência do *Token* no *header* da requisição.

Desta forma, ao realizar a autenticação do usuário, gera-se uma *string* na base64UrlEncode e esta é enviada por meio do *header* da requisição HTTP para a aplicação cliente.

A aplicação cliente ao realizar uma nova requisição HTTP ao servidor, deve enviar no cabeçalho da mesma o *token* recebido. Deste modo, com exceção do *login* e do cadastro de um novo usuário, é realizada a validação do *token* JWT e apenas, com este validado permite-se o acesso as demais funcionalidades.

Na próxima seção será apresentada a arquitetura da aplicação *Android*, ou seja, a aplicação cliente.

4.1.2. Aplicação *Android*

O início do desenvolvimento na parte de *front-end* foi o período de testes, no qual estudou-se, criou-se uma prova de conceito simples para chegar à tomada de decisão do caminho que iria se seguir no desenvolvimento da aplicação.

O uso do *Formik e Yup* como explicado no tópico 3.6.2.2, foi uma das decisões tomadas logo no início, para garantir um padrão na integração em todas as telas. O *React Native* permite o uso de bibliotecas nativas e *open source* que facilitam a construção e desenvolvimento, no entanto isso traz a dificuldade de conseguir cruzar as versões de diferentes bibliotecas da maneira correta.

4.1.3. Estruturação e planejamento do projeto

Uma vez que as versões dos diferentes pacotes, eram uma peça fundamental para se manter de forma coesa e organizada, criou-se um repositório no Git com um projeto *React Native* vazio, os arquivos que sofriam alterações por parte de toda a equipe que estava desenvolvendo o *front-end* da aplicação, são basicamente a pasta *src*, que contém os dados mutáveis do projeto e o arquivo *package.json* (FIGURA 18) que contém o nome de todas as bibliotecas utilizada no projeto, com sua determinada versão. Com uso desse controle de versionamento, foi possível desenvolver de forma simultânea em um mesmo projeto, no qual todos da equipe possuíam dados atualizados.

FIGURA 18 – ARQUIVO PACKAGE.JSON

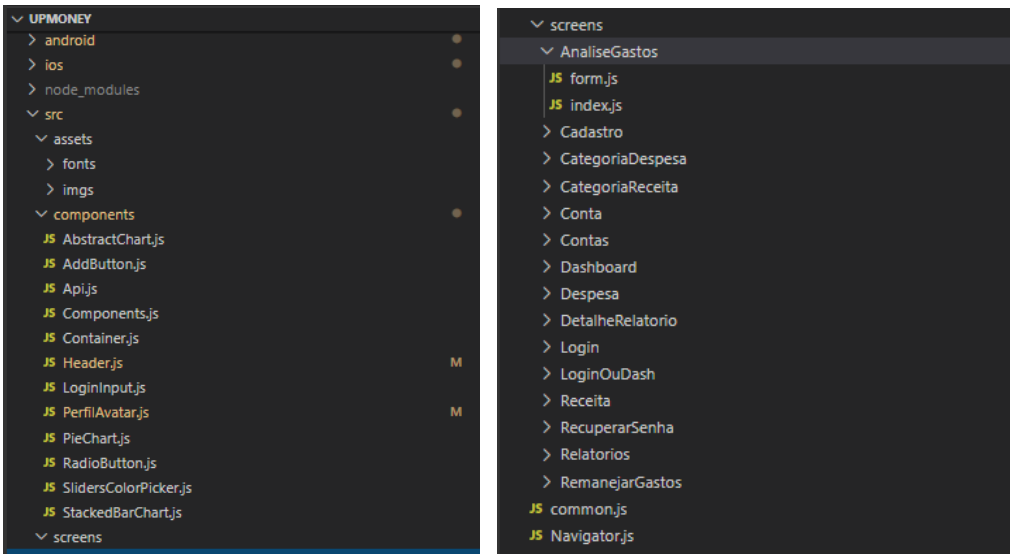
FONTE: Os Autores (2020).

Com o repositório criado, iniciou-se a criação da estrutura de pastas, conforme FIGURA 19, para organizar o desenvolvimento da aplicação. Optou-se por criar uma pasta principal nomeada como *src.* Abaixo dela encontra-se a pasta *assets* (encontram-se imagens e fontes utilizadas no projeto), a pasta

components - utilizada tanto para conjuntos isolados de lógica, visualização e possível estilização – e por fim a pasta *screens* que é composta por subpastas que representam cada tela do aplicativo, essas subpastas contém dois arquivos, um *form.js* e um *index.js.*

O *index.js* seria a base de toda a estrutura da tela, no qual são aplicadas as estilizações padrões do software construído. Dentro do *index.js*, é realizado a chamada do *form.js,* onde está contido todo o conteúdo do formulário em si, desde a estilização específica para aquele formulário quando as requisições ao servidor.

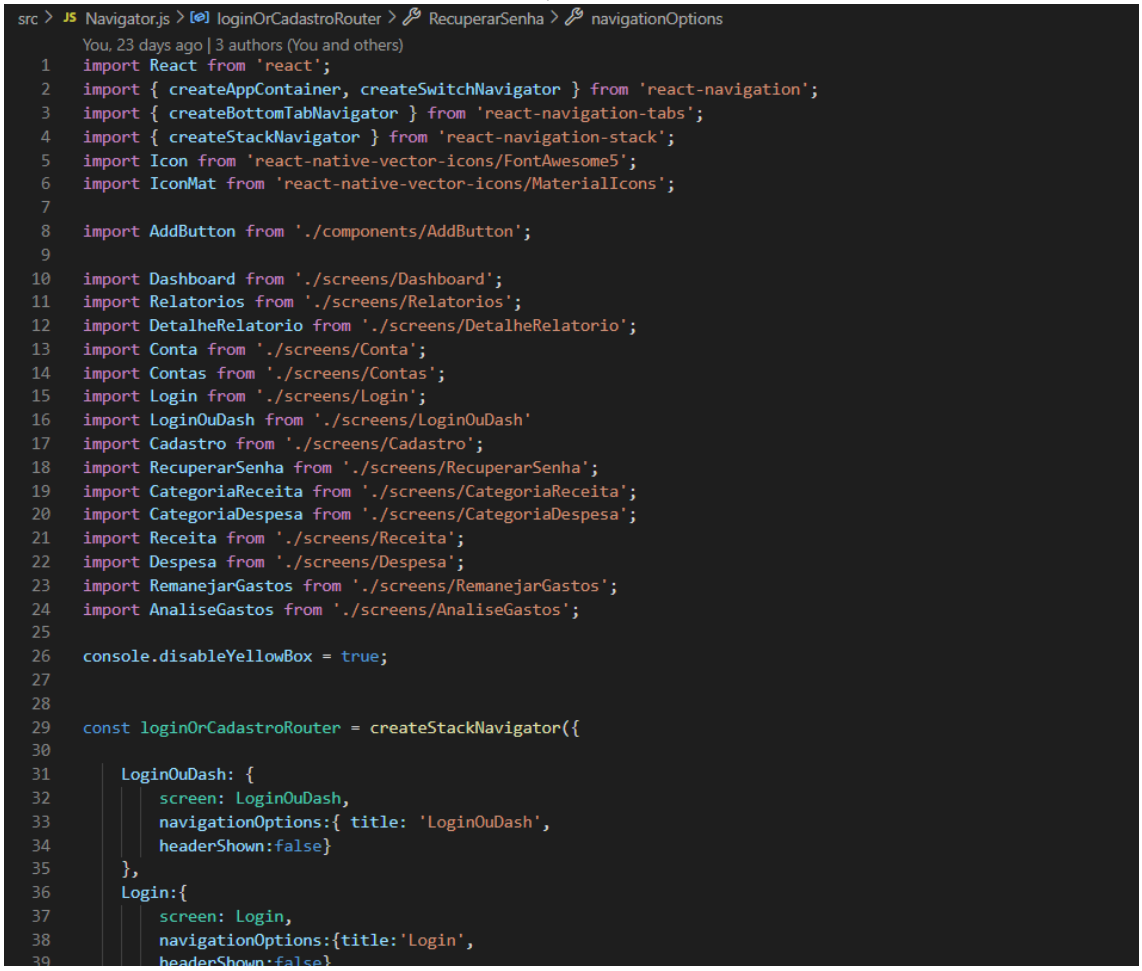
FIGURA 19 – ESTRUTURA DE PASTAS APP UPMONEY



FONTE: Os Autores (2020).

Um arquivo muito importante do projeto também, é o *Navigator.js* (FIGURA 20), que contém todas as rotas de navegação do aplicativo. Ao criar um arquivo que agrupe essas informações, existe a melhoria do esforço de manutenção e organiza a estrutura do projeto como um todo.

FIGURA 20 – ARQUIVO NAVIGATOR.JS

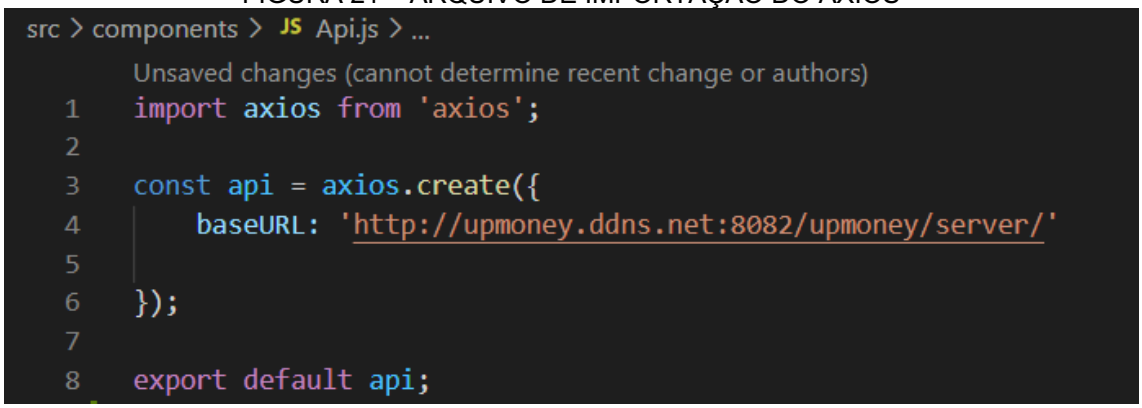


```
src > JS Navigator.js > loginOrCadastroRouter > RecuperarSenha > navigationOptions
You, 23 days ago | 3 authors (You and others)
import React from 'react';
import { createAppContainer, createSwitchNavigator } from 'react-navigation';
import { createBottomTabNavigator } from 'react-navigation-tabs';
import { createStackNavigator } from 'react-navigation-stack';
import Icon from 'react-native-vector-icons/FontAwesome5';
import IconMat from 'react-native-vector-icons/MaterialIcons';

import AddButton from './components/AddButton';

import Dashboard from './screens/Dashboard';
import Relatorios from './screens/Relatorios';
import DetalheRelatorio from './screens/DetalheRelatorio';
import Conta from './screens/Conta';
import Contas from './screens/Contas';
import Login from './screens/Login';
import LoginOuDash from './screens/LoginOuDash'
import Cadastro from './screens/Cadastro';
import RecuperarSenha from './screens/RecuperarSenha';
import CategoriaReceita from './screens/CategoriaReceita';
import CategoriaDespesa from './screens/CategoriaDespesa';
import Receita from './screens/Receita';
import Despesa from './screens/Despesa';
import RemanejarGastos from './screens/RemanejarGastos';
import AnaliseGastos from './screens/AnaliseGastos';

console.disableYellowBox = true;


const loginOrCadastroRouter = createStackNavigator({

    LoginOuDash: {
        screen: LoginOuDash,
        navigationOptions:{ title: 'LoginOuDash',
        headerShown:false}
    },
    Login:{
        screen: Login,
        navigationOptions:{title:'Login',
        headerShown:false}
```

FONTE: Os Autores (2020).

Ainda pensando em reutilização de código, criou-se o arquivo *api.js.* Para fazer a comunicação entre a parte de cliente e servidor, utilizou-se o pacote *axios*. Para utilizá-lo é preciso importá-lo em cada tela, mas para simplificar essas chamadas, criou-se esse arquivo único na pasta de *components,* conforme FIGURA 21 que faz a importação da biblioteca e traz a *url* base do servidor, para evitar repetições desnecessárias na hora das requisições. Dessa forma, ao invés do *axios*, esse arquivo é importado em todas as telas, simplificando as chamadas de *urls.*

FIGURA 21 – ARQUIVO DE IMPORTAÇÃO DO AXIOS



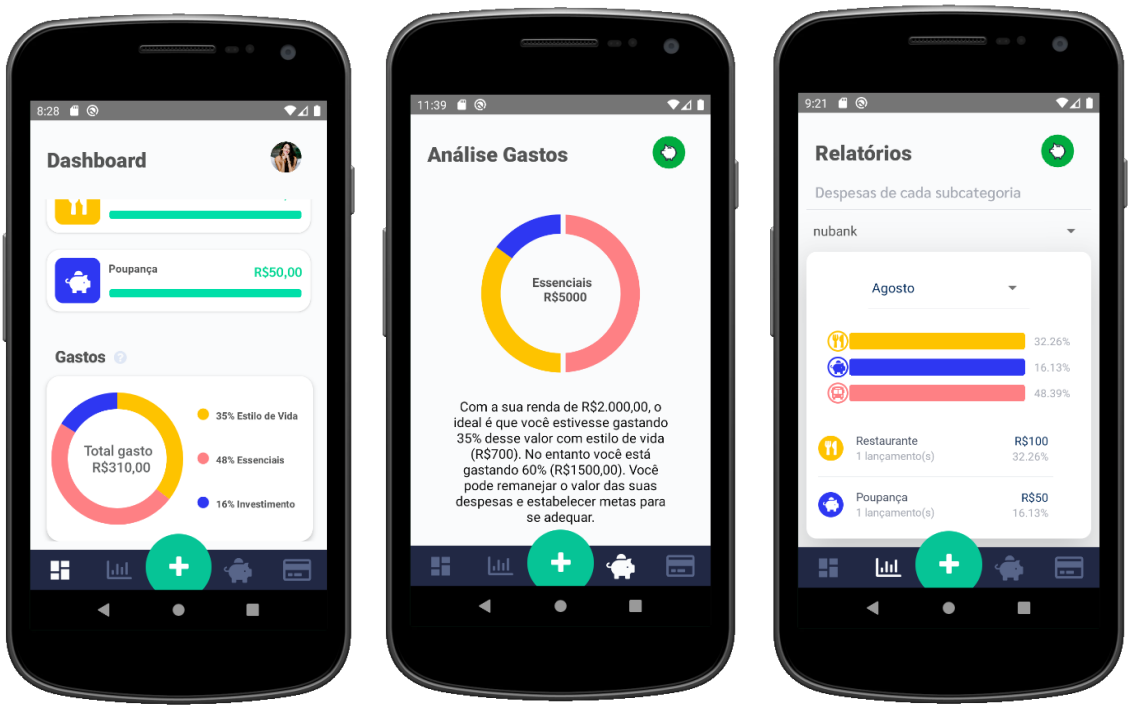
FONTE: Os Autores (2020).

4.1.4. Visualização de dados

Como foi explicado no tópico 1.1 desse projeto, na contextualização do problema, o aplicativo é focado em planejamento financeiro utilizando a regra 50/35/15 como base. Portanto, as informações de gastos do usuário foram pensadas e estruturadas para trazer os dados de despesas e receitas, agrupadas e organizadas dentro dessa regra, de maneira que fosse possível a fácil compreensão do usuário.

Por esse motivo, criar um layout de padronização pras telas foi essencial, assim como a padronização de cores entre as 3 categorias bases. Definiu-se que Gastos Essenciais receberia a cor rosa claro para sua identificação, Investimento a cor azul e por fim Estilo de vida a cor amarela. Esse padrão se mantém ao longo de todas as telas, conforme exemplo apresentado na FIGURA 22.

FIGURA 22 – EXEMPLO DE PADRONIZAÇÃO DE ESTILOS E CORES



FONTE: Os Autores (2020).

4.2. FUNCIONALIDADES

Nesta seção serão apresentadas as funcionalidades do aplicativo *Android* desenvolvido.

4.2.1. *Login*

Para efetivar o *login*, o usuário deve informar o e-mail e a senha. Além disso, a tela apresenta um *link* que direciona o usuário para a tela de recuperar senha, outro para o cadastro de um novo usuário e por fim o botão que realiza a entrada no usuário no aplicativo, realizando a validação dos dados informados e retornando um *token* de segurança para mantê-lo conectado caso minimize ou feche o aplicativo. A tela de *login* pode ser visualizada na FIGURA 23.

FIGURA 23 – TELA DE LOGIN

FONTE: Os Autores (2020).

4.2.2. Recuperar senha

Para realizar a recuperação de senha, o usuário deve informar o e-mail cadastrado e clicar no botão de enviar. Em poucos minutos o usuário recebe um e-mail com uma nova senha gerada automaticamente pelo sistema para que possa realizar o login, feito isso é possível alterá-la caso desejar (FIGURA 24).

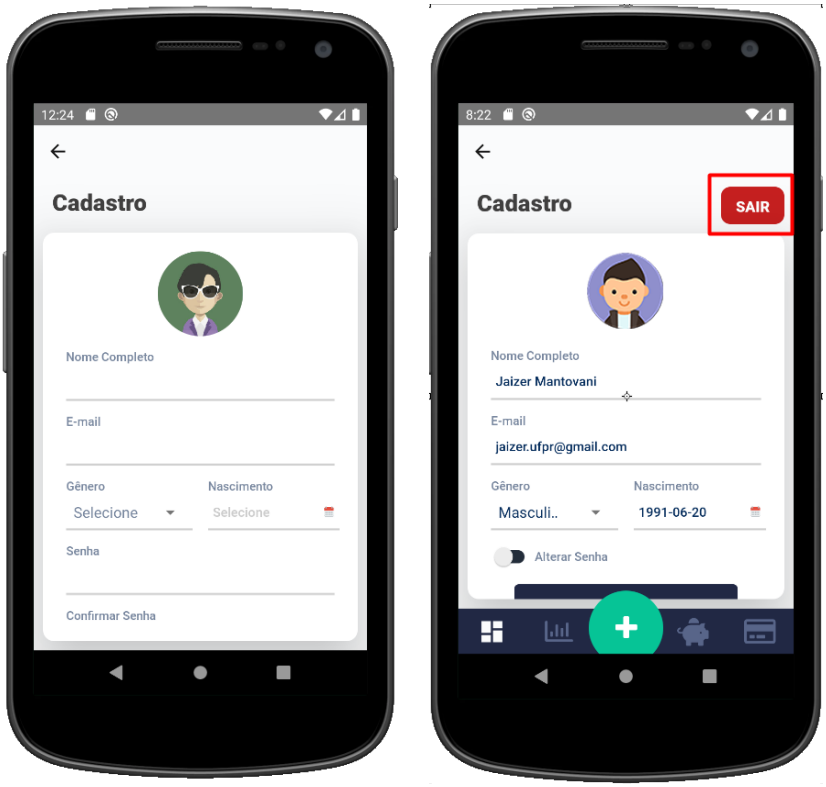
FIGURA 24 – TELA DE RECUPERAR SENHA

FONTE: Os Autores (2020).

4.2.3. Cadastro e Perfil

Para a realizar o cadastro de um novo usuário, este deverá escolher um avatar, informar nome completo, e-mail, gênero, data de nascimento, senha e confirmar a senha. Ao clicar no botão Enviar, o aplicativo fará todas as validações e enviará os dados ao servidor. Logo em seguida uma mensagem de sucesso será exibida e haverá um redirecionamento para o *login*. Por fim, a mesma tela poderá ser acessada após a autenticação no aplicativo para a edição dos dados de perfil do usuário, além disto, este poderá realizar o *logout* da aplicação através do botão localizado no *header*, o que invalida o *token* gerado na autenticação e redireciona o usuário para a tela de *login*. A FIGURA 25 apresenta as telas com as funcionalidades descritas anteriormente.

FIGURA 25 – TELA DE MATER O USUÁRIO



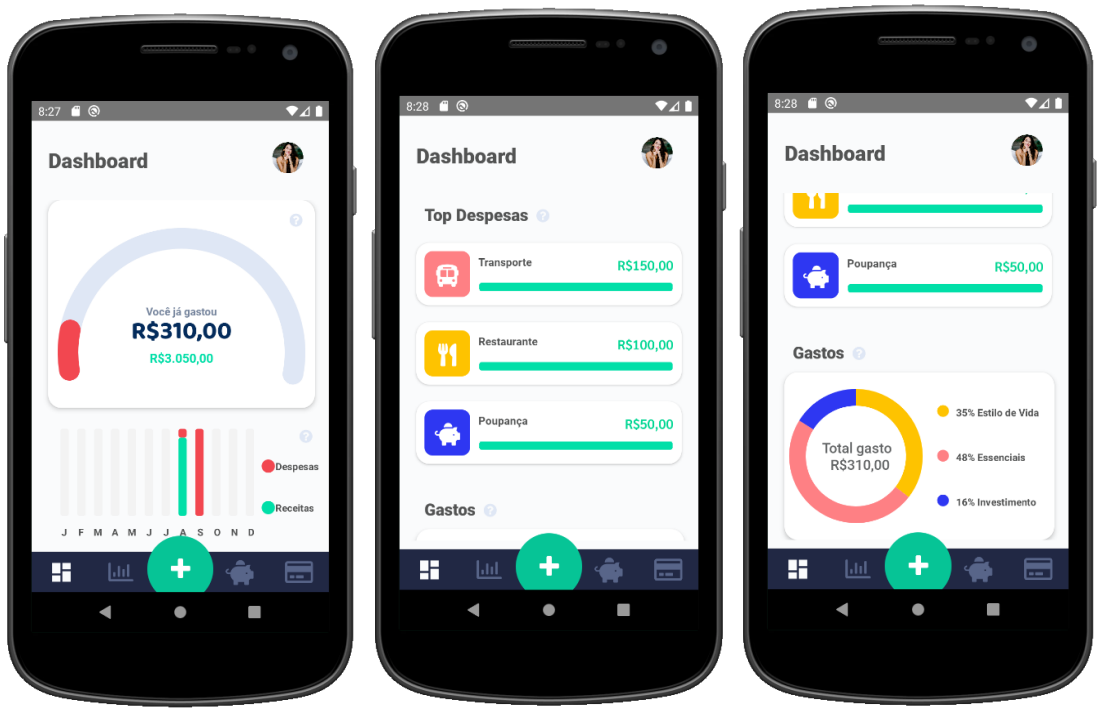
FONTE: Os Autores (2020).

4.2.4. *Dashboard*

Após a autenticação do *login*, existe o redirecionamento para a tela de *Dashboard* (FIGURA 26), sendo está composta por quatro relatórios:

- Total de gastos realizados sob as receitas cadastradas no mês vigente;
- Porcentagem entre receitas e despesas ao longo dos meses do ano;
- As três subcategorias com mais despesas do usuário cadastradas no mês vigente;
- A porcentagem de gastos em cada categoria de despesa para o mês em questão.

FIGURA 26 – TELAS DASHBOARD



FONTE: Os Autores (2020).

Cada gráfico possui no seu título um botão “?” que ao clicar irá fornecer informações dos dados que estão sendo apresentados, para uma melhor compreensão e análise.

4.2.5. *Header* e *Toolbar*

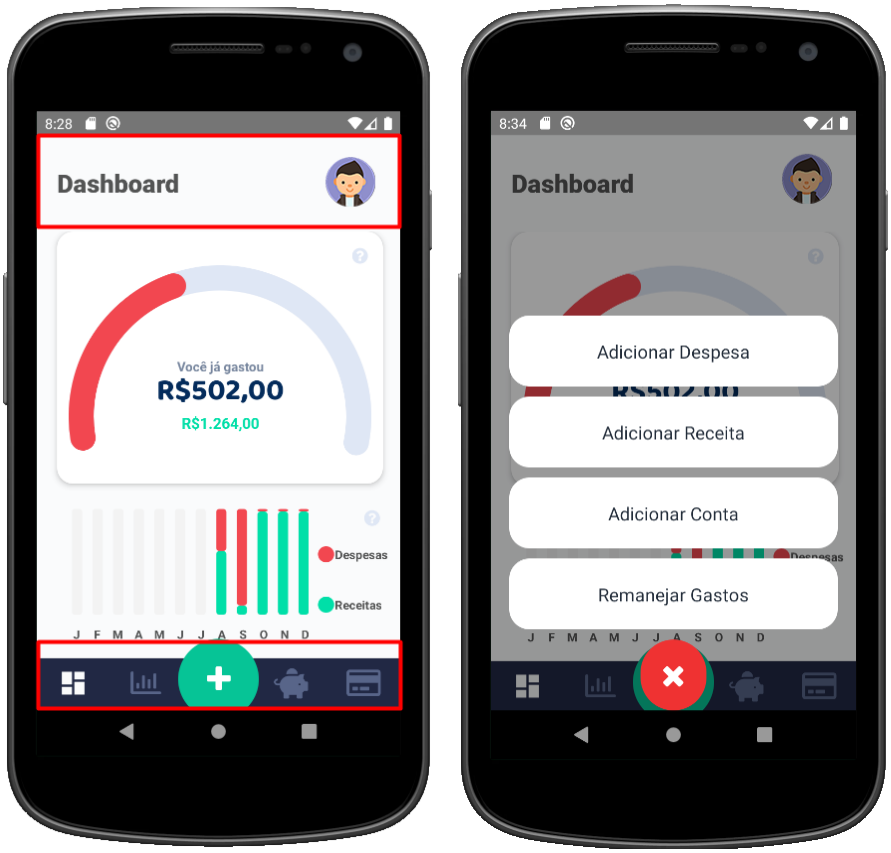
Após realizar o login no aplicativo, todas as interfaces seguintes irão possuir um *header* no superior da tela e um *toolbar* localizado no inferior desta. O *header* é composto pelo título da tela, o ícone do avatar do usuário, pelo qual é possível acessar a tela de edição de perfil e uma seta “voltar” pela qual o usuário é direcionado a tela acessada anteriormente. Na FIGURA 27 são apresentados o *header* e *toolbar*.

O *toolbar* será constituído por cinco diferentes ícones que correspondem a diferentes navegações do aplicativo, sendo elas:

- *Dashboard*;
- Relatórios;
- Análise de Gastos;
- Contas;

- Submenu que possui acesso as funcionalidades de Adicionar Despesa, Adicionar Receita, Adicionar Conta e Remanejar Gastos.

FIGURA 27 – *HEADER* E *TOOLBAR*



FONTE: Os Autores (2020).

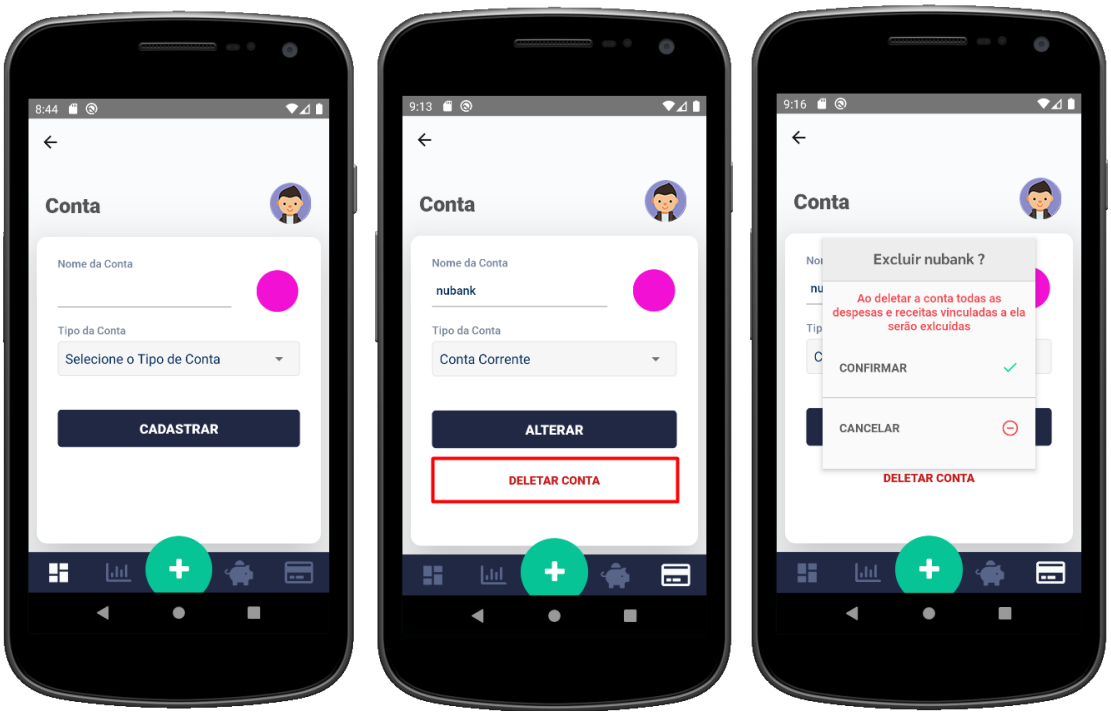
4.2.6. Conta

Para o cadastro de uma nova conta, é necessário informar o nome da conta, cor, tipo da conta e qual o valor inicial da mesma, sendo, posteriormente, editar todas as informações exceto o valor inicial. Após clicar no botão de envio de dados será mostrada uma mensagem de sucesso para o usuário redirecionando-o para a tela de contas.

Além disso, no modo de edição será exibido um botão com a opção de deletar a conta. Ao pressionar esta opção, um modal irá aparecer para confirmação de conta. Caso haja a confirmação de exclusão, todas as receitas

e despesas vinculadas também serão. A FIGURA 28 apresenta as telas com as funcionalidades descritas anteriormente.

FIGURA 28 – TELAS DE MANTER CONTA

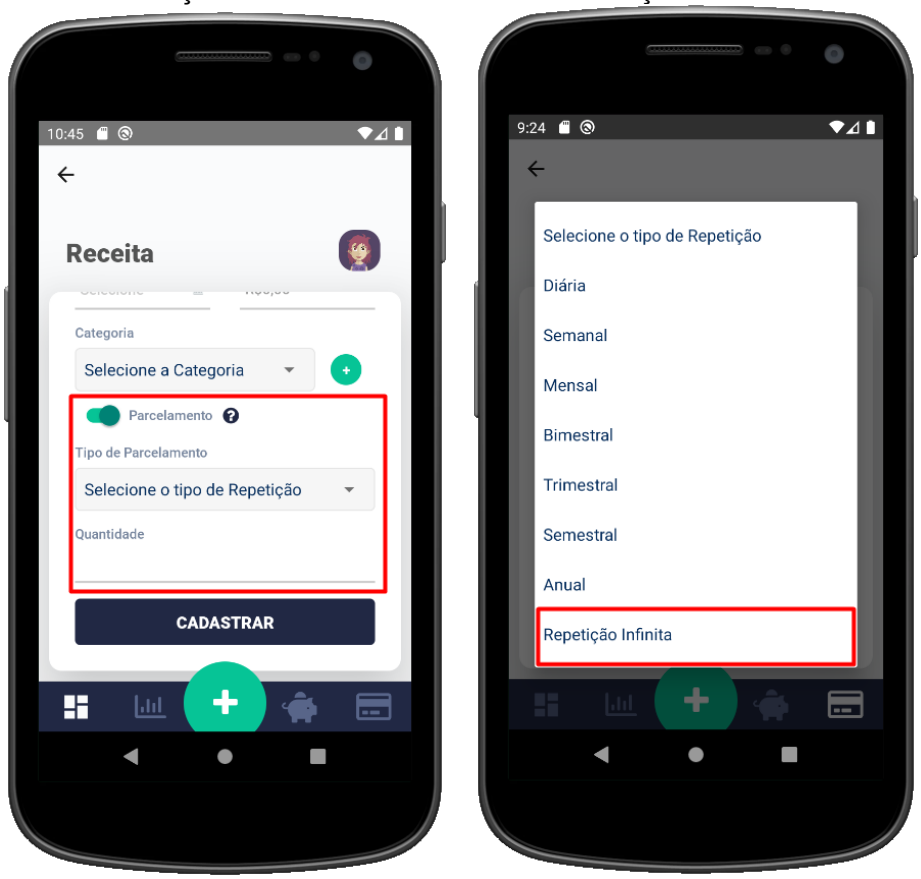


FONTE: Os Autores (2020).

4.2.7. Receita

Para o cadastro de uma nova receita é necessário informar obrigatoriamente a conta, descrição, data, valor total e a categoria. Caso selecione a opção de parcelamento, a tela abrirá dois novos campos para serem preenchidos (FIGURA 29), sendo eles tipo de parcelamento e a quantidade. Ao selecionar essa opção, o campo “Valor total” será parcelado na quantidade informada e cada parcela será debitada de acordo com o período selecionado no *combobox* “Tipo de parcelamento”. Ainda sobre o tipo de parcelamento, dentro do *combobox* existe uma opção “repetição infinita”, o usuário ao selecionar essa opção ao invés de ter o campo “Valor total” parcelado, terá esse valor debitado mensalmente durante 5 anos.

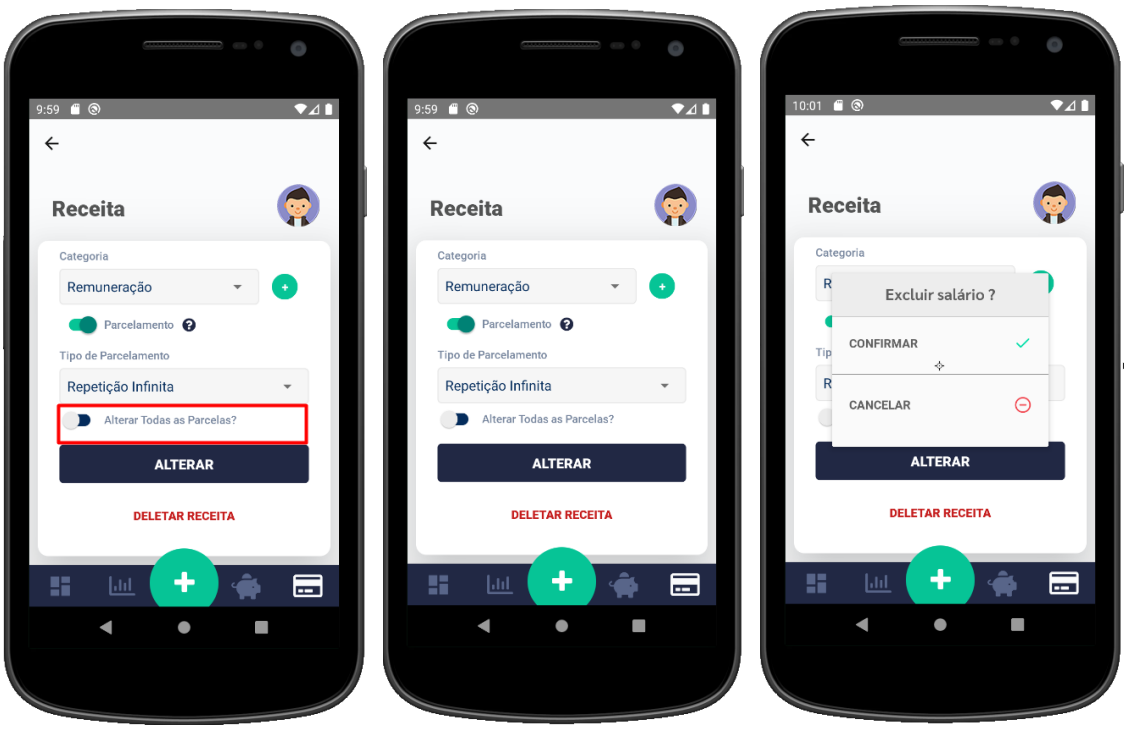
FIGURA 29 – OPÇÃO DE PARCELAMENTO E REPETIÇÃO INFINITA RECEITA



FONTE: Os Autores (2020).

No modo de edição são exibidas algumas opções a mais, uma delas é um *switch* de alterar todas as parcelas, caso a receita a ser editada seja uma com valor parcelado ou repetição infinita essa opção é apresentada. Se caso selecionado, as informações serão alteradas e recalculadas a partir dessa receita para todas ainda restantes. Além disso é exibido um botão com a opção deletar a receita. Ao pressionar esta opção, um modal de confirmação de exclusão aparece (FIGURA 30). Assim que confirmar a exclusão, uma mensagem do sistema é apresentada e em seguida é feito o redirecionamento para a tela de “Visualizar Contas”.

FIGURA 30 – TELAS DE MANTER RECEITA

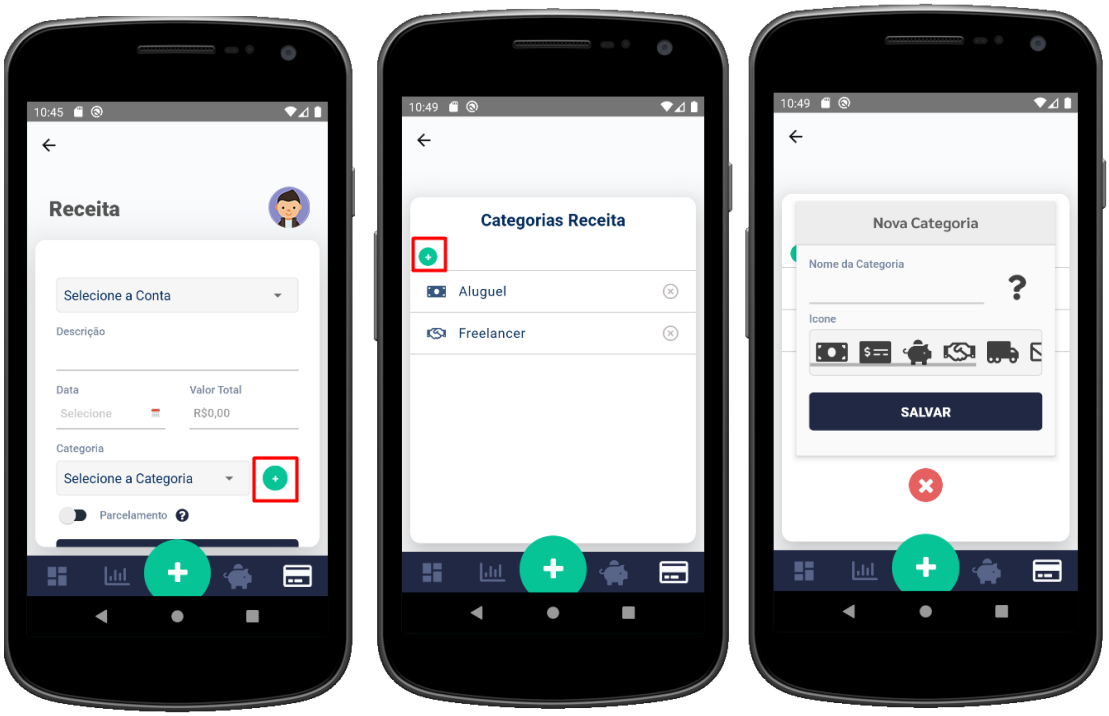


FONTE: Os Autores (2020).

4.2.8. Categoria Receita

Ao clicar no ícone de “+” na tela de receita, a tela de categorias receita é apresentada, como mostra a FIGURA 31. Esta é composta por um botão de “+” para adicionar uma nova categoria, uma lista com todas as categorias do usuário sendo que cada uma delas possuirá as opções de editar e deletar. Ao pressionar o botão de adicionar uma categoria, um modal será apresentado (FIGURA 29) para que o usuário preencha as seguintes informações: nome da categoria e ícone. Assim que pressionar o botão de salvar, uma mensagem de sucesso é apresentada e a lista de categorias é recarregada.

FIGURA 31 – TELA DE CATEGORIA RECEITA E MODAL NOVA CATEGORIA RECEITA

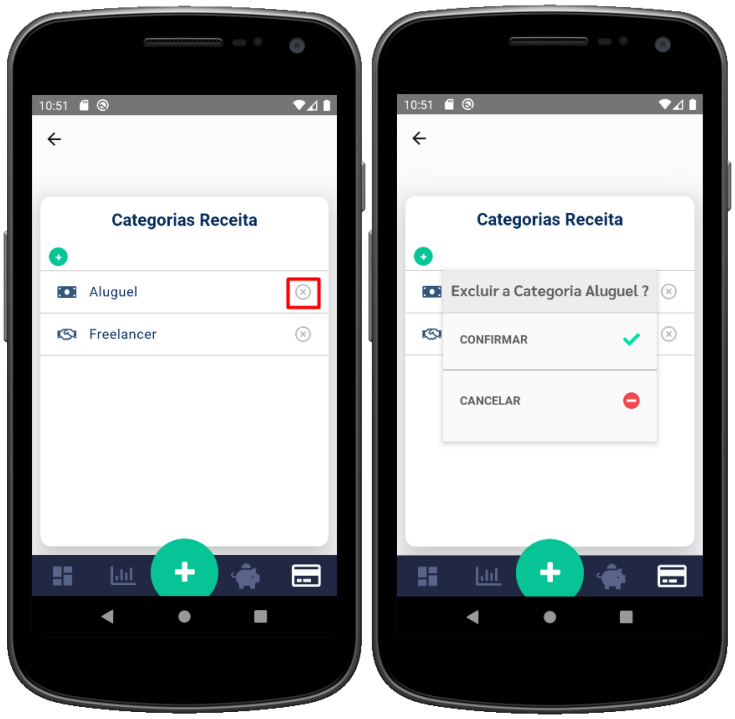


FONTE: Os Autores (2020).

Caso o usuário queira editar uma categoria, basta clicar na linha da categoria desejada na lista e o mesmo modal de inserção é apresentado, entretanto com as informações já preenchidas da categoria selecionada.

Por fim, ao clicar no ícone de exclusão um modal é exibido (FIGURA 32) para confirmar a exclusão da categoria. Assim que a exclusão for efetivada, o usuário recebe uma mensagem do sistema e em seguida a lista de categorias será recarregada.

FIGURA 32 – MODAL EXCLUIR CATEGORIA RECEITA

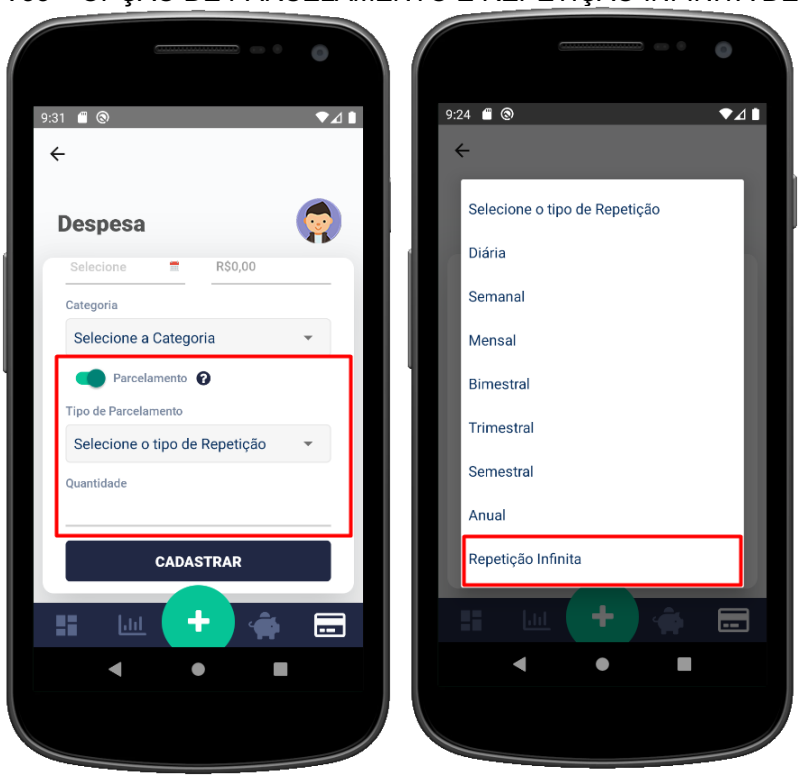


FONTE: Os Autores (2020).

4.2.9. Despesa

Ao cadastrar uma nova despesa o usuário deve informar obrigatoriamente a conta, descrição, data, valor total, categoria e subcategoria. Caso selecione a opção de parcelamento (FIGURA 33), a tela abrirá dois novos campos para serem preenchidos, sendo eles tipo de parcelamento e a quantidade. Ao selecionar essa opção, o campo “Valor total” será parcelado na quantidade informada e cada parcela será descontada de acordo com o período selecionado no *combobox* “Tipo de parcelamento”. Ainda sobre o tipo de parcelamento, dentro do *combobox* existe uma opção “repetição infinita”, o usuário ao selecionar essa opção ao invés de ter o campo “Valor total” parcelado, terá esse valor descontado mensalmente durante 5 anos.

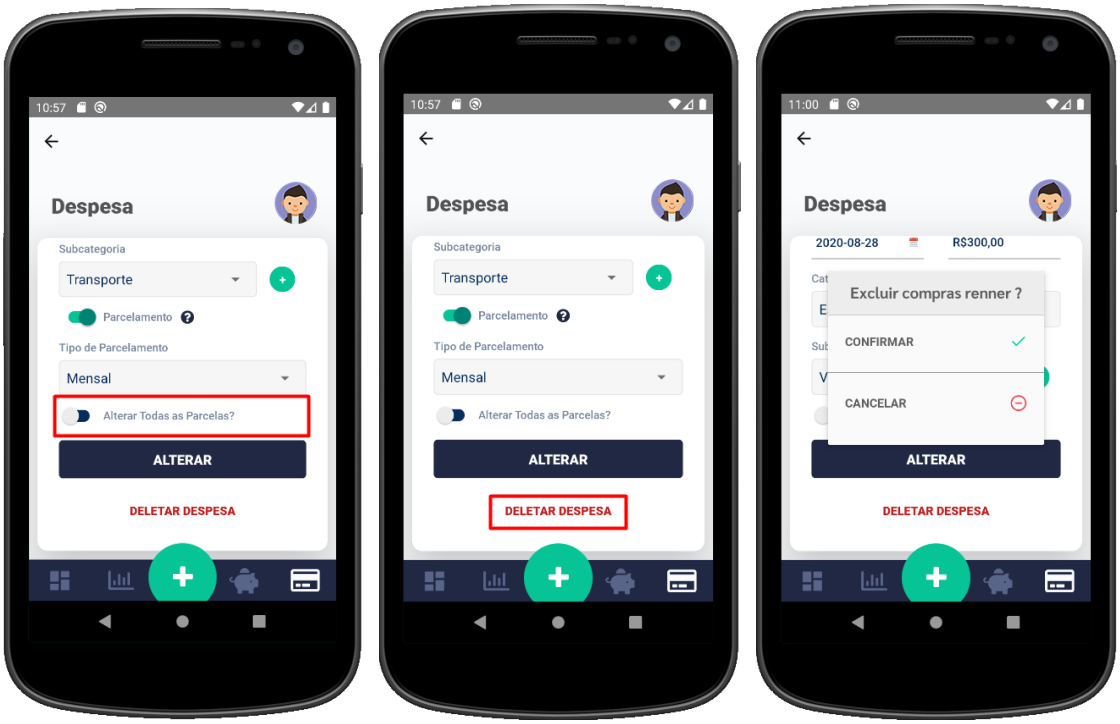
FIGURA 33 – OPÇÃO DE PARCELAMENTO E REPETIÇÃO INFINITA DESPESA



FONTE: Os Autores (2020).

No modo de edição são exibidas algumas opções a mais, uma delas é um *switch* de alterar todas as parcelas, caso a despesa a ser editada seja uma com valor parcelado ou repetição infinita essa opção é apresentada. Se caso selecionado, as informações serão alteradas e recalculadas a partir dessa despesa para todas ainda restantes. Além disso é exibido um botão com a opção deletar a receita. Ao pressionar esta opção, um modal de confirmação de exclusão aparece (FIGURA 34). Assim que confirmar a exclusão, uma mensagem do sistema é apresentada e em seguida é feito o redirecionamento para a tela de “Visualizar Contas”.

FIGURA 34 – TELAS DE MANTER DESPESA



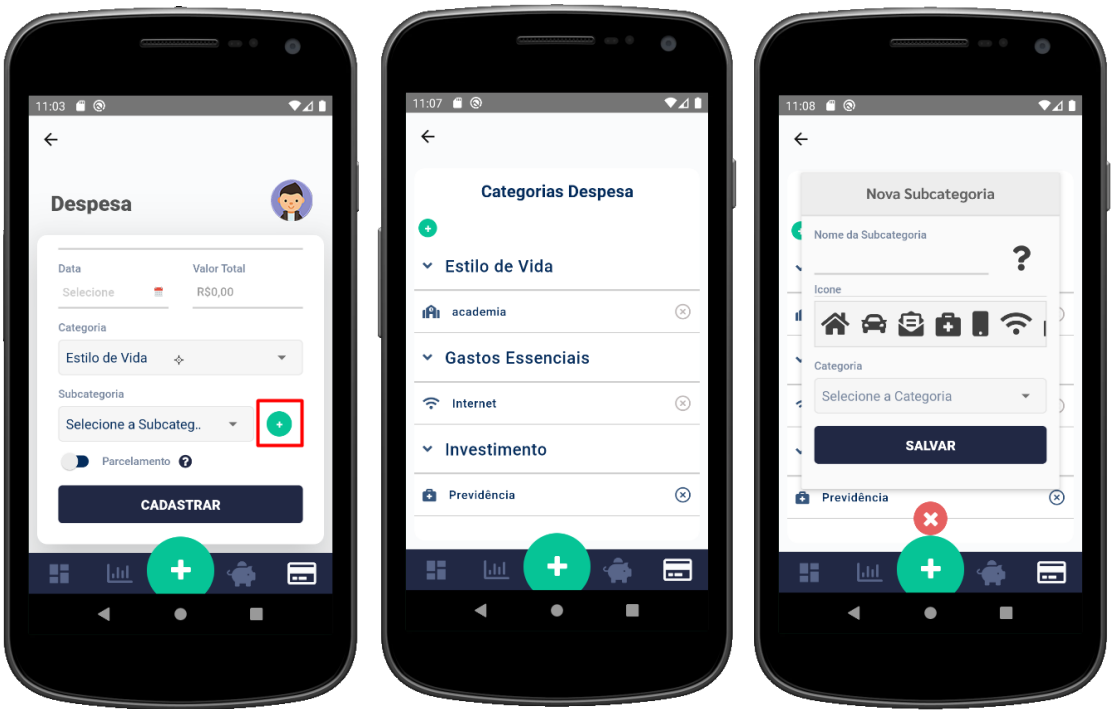
FONTE: Os Autores (2020).

4.2.10. Categoria Despesa

Ao clicar no ícone de “+” na tela de Manter Despesa, a tela de subcategorias despesa onde é possível adicionar novas subcategorias e visualizar as já cadastradas. As subcategorias são divididas entre as categorias bases de acordo com as escolhas feitas pelo usuário, a partir da seleção dos *dropdowns* é possível visualizar as listas com as opções de editar e excluir (FIGURA 35).

Ao escolher a opção de adicionar uma subcategoria, um modal será apresentado para que o usuário possa preencher as seguintes informações nome da categoria, ícone e categoria base. Assim que pressionar o botão de salvar uma mensagem de sucesso será apresentada e a lista de subcategorias será recarregada.

FIGURA 35 – TELA DE CATEGORIA DESPESA E MODAL NOVA CATEGORIA DESPESA

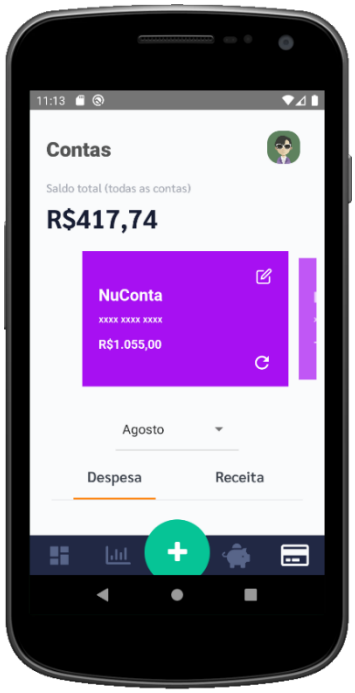


FONTE: Os Autores (2020).

4.2.11. Visualizar Contas

No *toolbar* existe o redirecionamento para a tela de “Visualizar Contas”, nela são apresentados uma lista de cartões que representam as contas cadastradas, as informações apresentadas são a soma do saldo de todas as contas, o saldo individual de cada um, um combo selecionável dos meses e duas listas referentes às despesas e receitas da conta e do mês selecionados, conforme ilustrado na FIGURA 36.

FIGURA 36 – TELA DE VISUALIZAR CONTAS

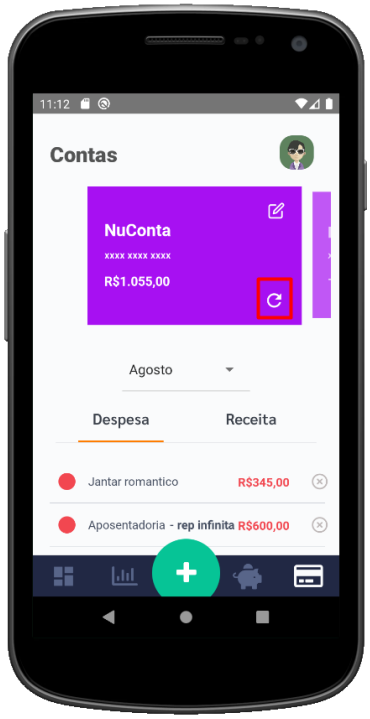


FONTE: Os Autores (2020).

O ícone de recarregar localizado em cada cartão de conta é responsável pela atualização dos dados nessa tela, portanto deve ser acionado sempre que o usuário necessitar que os dados sejam carregados novamente (FIGURA 37).

Essa tela irá buscar todas as listas de receita e despesa do mês selecionado. Além disso o usuário poderá analisar suas receitas e despesas de outros meses alterando o combo abaixo dos cartões. Ao clicar no ícone de edição localizado na parte superior do cartão, o usuário será redirecionado para a tela de conta em modo de edição. Por fim caso o usuário queira editar qualquer informação de uma receita ou despesa, basta clicar na linha desejada na lista de receita ou despesa que em seguida será redirecionado para a tela correspondente em modo de edição. A opção de exclusão de despesas e receitas também é disponibilizada nessa tela.

FIGURA 37 – ATUALIZAR DADOS

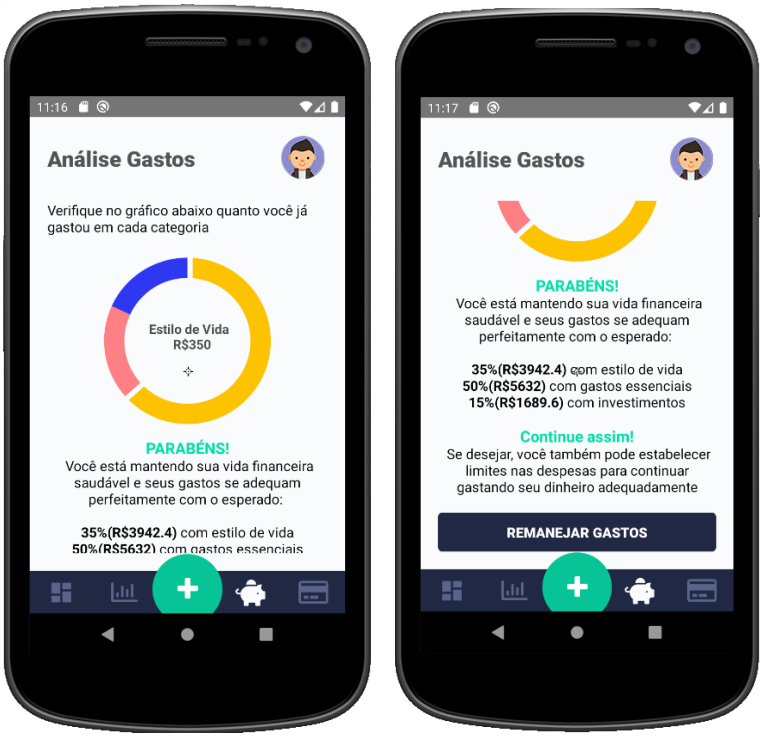


FONTE: Os Autores (2020).

4.2.12. Análise de Gastos

No *toolbar* existe o redirecionamento para a tela de “Análise de Gastos”, nesse momento é apresentado ao usuário os seus gastos separados pelas categorias bases em um gráfico pizza interativo, ou seja, é possível selecionar cada fatia que representa cada categoria base e saber qual o gasto atual em cada uma delas. Nesse momento é informado se as despesas estão adequadas à regra 50/35/15 (FIGURA 38) ou não.

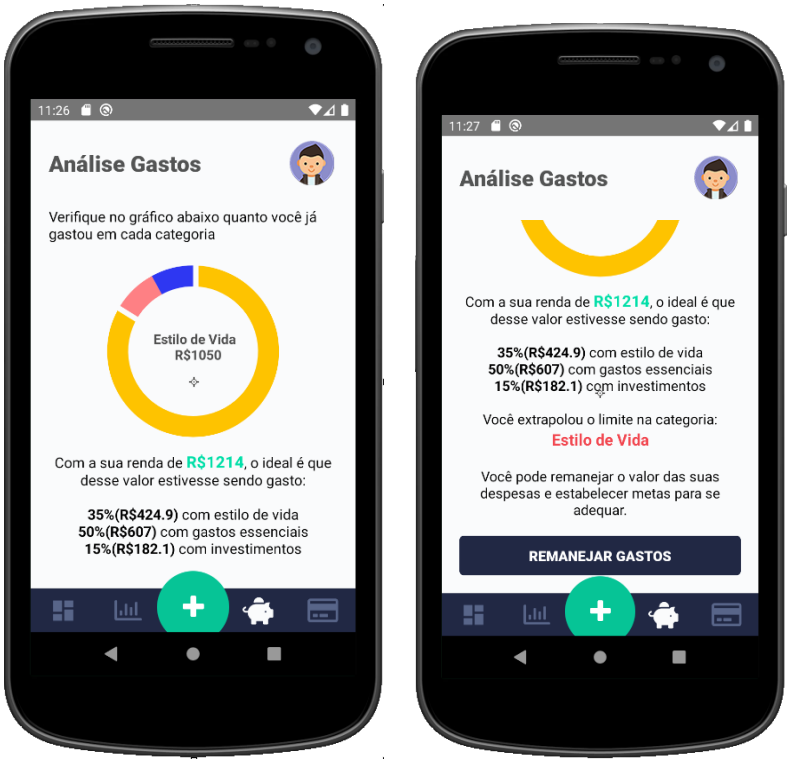
FIGURA 38 – ANÁLISE POSITIVA



FONTE: Os Autores (2020).

Além do gráfico é apresentado um texto explicativo dinâmico que mostra os valores, informações de categoria e adequação financeira. O texto sofre alterações caso os gastos estejam adequados ou extrapolando os limites, caso em alguma categoria o valor esteja ultrapassando a regra, será informado na tela qual é essa categoria (FIGURA 39). Por fim, é disponibilizado um botão para acessar a tela de remanejar os gastos, possibilitando a correção ou manutenção alertada na tela de análise de gastos.

FIGURA 39 – ANÁLISE NEGATIVA



FONTE: Os Autores (2020).

4.2.13. Remanejar Despesas

Após ser redirecionado da tela de análise de gastos para a tela de remanejar despesa é possível interagir com os limites das categorias e subcategorias. Na parte superior da tela, o usuário poderá escolher quais informações ele deseja observar selecionando a categoria base (FIGURA 40).

FIGURA 40 – TELAS DE REMANEJAR GASTOS

FONTE: Os Autores (2020).

Logo abaixo serão apresentadas as informações da categoria, como a porcentagem, o valor recomendado e o valor gasto. As porcentagens padrões que o aplicativo cadastra para novos usuários é sempre 50, 35 e 15. Caso o usuário queira adaptar estes valores, basta clicar na opção de edição ao lado do valor recomendado e um modal com três campos será apresentado (FIGURA 41). Com isso o usuário deverá informar o valor das três categorias mães, vale ressaltar que a somatória dos três valores deve resultar em 100%, caso contrário será apresentado uma mensagem dizendo que o valor gasto extrapolou (FIGURA 41).

FIGURA 41 – MODAL DE ALTERAÇÃO PORCENTAGEM CATEGORIA BASE

FONTE: Os Autores (2020).

Caso o valor gasto supere o valor recomendado da categoria, a cor do box em questão será alterada (FIGURA 42). Logo abaixo destas informações, irá possuir uma lista com todas as subcategorias referentes a categoria mãe com as seguintes informações descrição, valor limite, valor gasto e a porcentagem de quanto o usuário já gastou referente ao seu limite. Para a alteração do limite de uma subcategoria, o usuário deve clicar acima da linha desejada e um modal será apresentado com um campo a ser preenchido com o novo valor (FIGURA 39). Assim que usuário enviar os dados, a tela irá recarregar novamente a lista de subcategorias, atualizando seus dados. Caso a somatória dos valores limites sejam superiores ao valor recomendado da categoria, uma mensagem será exibida ao usuário alertando-o.

FIGURA 42 – VALIDAÇÕES DE GASTOS E LIMITES

FONTE: Os Autores (2020).

4.2.14. Relatórios

Para exibir os dados do relatório de despesas de subcategorias, o usuário deverá selecionar uma das contas disponíveis no combo, além do mês que deseja analisar. Logo em seguida a tela apresentará as porcentagens referentes aos gastos de cada subcategoria em um gráfico de barras, acompanhado de uma lista com todos os lançamentos separados por subcategoria (FIGURA 39). As cores das barras são definidas de acordo com a sua categoria base. Caso o usuário queira mais detalhes dos lançamentos, basta clicar acima da linha desejada e será redirecionado para o relatório de detalhes de subcategoria (FIGURA 43), no qual será apresentado o valor gasto, o valor limite que é definido em “Remanejar Gastos”, porcentagem, e a quantidade restante para atingir o limite estabelecido. Caso o limite seja extrapolado, a barra de progresso circular e o valor total mudarão para a cor vermelha. Logo abaixo, é apresentado uma lista com todas as despesas realizadas para a subcategoria em questão.

FIGURA 43 – TELA DE RELATÓRIOS DE SUBCATEGORIAS

FONTE: Os Autores (2020).

FIGURA 44 – TELA DE DETALHE DE GASTOS DE UMA SUBCATEGORIA

FONTE: Os Autores (2020).

Na seção seguinte serão descritas as considerações finais com relação ao desenvolvimento deste projeto.

5. CONSIDERAÇÕES FINAIS

O projeto *UpMoney* teve como foco, para a primeira fase, a delimitação de escopo, bem como a documentação e a modelagem do aplicativo. Para isso, foram realizados diversos *brainstormings* para definir os requisitos do aplicativo (APÊNDICE A). Uma vez definido o escopo do projeto, para representação dos requisitos levantados foi utilizada a linguagem unificada UML para a construção dos Diagramas de Caso de Uso, apresentados nos APÊNDICES B e C, de Classes (APÊNDICE E) e de Sequência (APÊNDICE F).

Para facilitar a etapa de desenvolvimento do aplicativo, foram elaboradas, também, as especificações de caso de usos (APÊNDICE D) e realizada uma etapa de fundamentação teórica do projeto, bem como a definição dos materiais e métodos a serem utilizados. Além disso, desenvolveu-se uma prova de conceito para ambientação com as tecnologias propostas.

O projeto como um todo foi desafiador na parte técnica, muitas decisões foram tomadas baseadas no conhecimento e no tempo disponível. Ao tentar trazer funcionalidade às telas elaboradas na prototipação, notou-se que muitos elementos desta tinham um desenvolvimento complexo, portanto foi necessário realizar concessões e adaptações para simplificar sem perder o conceito principal do projeto.

Com relação a configuração do ambiente de desenvolvimento do *webservice* deste projeto, obteve-se dificuldade em sincronizar as versões das bibliotecas utilizadas, no entanto, após diversos testes obteve-se a combinação ideal para o funcionamento do mesmo.

Apesar de encontrar diversas dificuldades ao longo do desenvolvimento da aplicação, ora uma tecnologia desconhecida a se aprender, ora um erro não identificado, atingiu-se o objetivo inicial proposto, ou seja, todas as funcionalidades requeridas pela aplicação cliente estão cumprindo seu papel.

Assim obteve-se na segunda fase, uma aplicação *Android* voltada para auxiliar o controle financeiro através de funcionalidades que permitem criar registros e consolidar receitas e despesas do usuário, fornecendo uma análise da sua vida financeira baseada na metodologia 50/35/15, além de possibilitar cadastro de meta para limitar seus gastos. A aplicação foi desenvolvida em

Javascript, que opera em conjunto com uma aplicação servidora, desenvolvida na linguagem orientada a objetos *Java*.

Entretanto algumas funcionalidades planejadas na primeira fase como conciliação bancária, com o intuito de facilitar o registro de receitas e despesas do usuário, e a funcionalidade de criação de metas referente a investimento, tendo como objetivo principal sugerir investimentos adequados a renda do usuário, não puderam ser desenvolvidas devido a limitações de tempo e conhecimento da tecnologia. Dessa forma, estas funcionalidades seriam opções de melhoria para uma segunda versão do projeto.

REFERÊNCIAS

ANDERSON, D. J. **Kanban:** Mudança Evolucionária de Sucesso para Seu Negócio de Tecnologia. 1. ed. Sequim: Blue Hole Press, 2011.

ANDROID STUDIO. **Conheça o Android Studio**. Disponível em: <https://developer.android.com/studio/intro?hl=pt-br>. Acesso em: 1 maio 2020.

ARAUJO, F. C.; CALIFE, F. E. **A história não contada da Educação Financeira no Brasil**. Boa Vista: SCPC, 2014. 1-11p.

ASSOCIAÇÃO BRASILEIRA DAS ENTIDADES DOS MERCADOS FINANCEIRO E DE CAPITAIS (ANBIMA). **O raio X do investidor brasileiro**. 2018. Disponível em: <https://cointimes.com.br/wp-content/uploads/2018/08/Relatorio-Raio-X-Investidor-PT.pdf>. Acesso em: 20 set. 2019.

AXIOS. 2020. Disponível em: <https://www.npmjs.com/package/axios/>. Acesso em: 1 maio 2020.

BAUER, C.; KING, G. **HIbernate in action.** Greenwitch: Manning Publications, 2005.

BAUER, C.; KING, G. **Java Persistence with HIbernate.** ed. Revisada. Greenwitch: Manning Publications, 2007.

BASSOTTO, L. **Educação financeira:** problema para a maioria dos brasileiros**.** 2018. Disponível em: <https://cointimes.com.br/educacao-financeira-para-brasileiros/>. Acesso em: 20 set. 2019.

BERNARDI, R. **brModelo Portable 2.0**. 2013. Disponível em: <http://dev.rbtech.info/download-brmodelo-portable/>. Acesso em 02 de dezembro de 2019.

BLANCO, S. **Planejamento Financeiro**. Disponível em:< https://docplayer.com.br/18749642-Planejamento-financeiro.html>. Acesso em: Acesso em 20 setembro 2019.

BODIE, Z.; MERTON, R. C. **Finanças**. Porto Alegre: Bookman,1999.

BONO, E. **Novas Estratégias de Pensamento**. [S. l.]: Livrarias Nobel SA, 2000.

BOOCH G.; RUMBAUGH J.; JACOBSON I. **UML Guia do Usuário**. [S.l.]: Elsevier Editora Ltda, 2006.

BRONDANI, C. H.; AREND; C. F.; SOUZA; D. A. Z. de et al. **Tutorial Guia Prático de utilização da ferramenta Astah Community 6.1**. 2013. Disponível

em: <https://pt.scribd.com/doc/139768773/Astah-Comnunity>. Acesso em: 29 de out. 2019.

BUFFONI, S. S. O. **Apostila de Algoritmo Estruturado**. 2013. 14 p. Material de Aula Faculdades Integradas Anglo-Americano. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~pbueno/Arquivos/Algoritmos.pdf>. Acesso em: 01 dez. 2019.

BUSETTI, L. **Gerenciamento Financeiro Pessoal: Modelo de planejamento e controle para construção patrimonial.** 2012. 168 p. Trabalho de conclusão de curso (Bacharelado Administração)–Universidade Federal do Rio Grande do Sul - UFRS, Porto Alegre, 2012.

CASSERLY, M. **IOS vs Android: quem domina o mercado de smartphones?**. 2019. Disponível em: <https://pcworld.com.br/ios-vs-android-quem-domina-o-mercado-de-smartphones/>. Acesso em: 29 de out. 2019.

CERBASI, G. P. **A complexa educação financeira**. Disponível em: <http://www.maisdinheiro.com.br/artigos/4/91/a-complexa-educacao-financeira>. Acesso em: 29 out. 2019.

CHOCOLATEY: **The Package Manager for Windows**. Version: 0.10.15. [S.l]: Chocolatey, 2020. Disponível em: <https://chocolatey.org> Acesso em: 1 maio 2020.

CONFEDERAÇÃO NACIONAL DE DIRIGENTES LOJISTAS (CNDL), **45% dos brasileiros não controlam as próprias finanças, mostra pesquisa sobre educação financeira do SPC Brasil e CNDL**. 2018. Disponível em <https://site.cndl.org.br/45-dos-brasileiros-nao-controlam-as-proprias-financas-mostra-pesquisa-sobre-educacao-financeira-do-spc-brasil-e-cndl/>. Acesso em: 17 out. 2020.

_____; SPC. **Inadimplência de Pessoas Físicas**. Disponível em: <www.spcbrasil.org.br › uploads › 2019/02 › Análise-1>. Acesso em: 17 out. 2019.

COHN, M. **Desenvolvimento de software com Scrum:** aplicando métodos ágeis com sucesso. Porto Alegre: Bookman, 2011. 496 p.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. Rio de Janeiro: Campus, 2004.

ECLIPSE FOUNDATION. **Eclipse Jersey**. [2020?]. Disponível em: <https://eclipse-ee4j.github.io/jersey/>. Acesso: 18 de set. 2020.

ESTRATÉGIA NACIONAL DE EDUCAÇÃO FINANCEIRA (ENEF). Conceito de Educação Financeira no Brasil. 2017. Disponível em: <https://www.vidaedinheiro.gov.br/educacao-financeira-no-brasil/>. Acesso em: 20 out. 2019.

EWALD, L. S. **Sobrou dinheiro!: lições de economia domestica**. 2. ed. Rio de Janeiro: Bertrand Brasil, 2003.

EXPO. **Introduction to Expo**. [2019?]. Disponível em:<https://docs.expo.io>. Acesso em: 30 nov. 2019.

FIGMA. Prototyping Features. 2020 Disponível em: <https://www.figma.com/prototyping/>. Acesso em: 22 nov. 2019.

FORMIK docs. **Overview**. 2020. Disponível em: <https://formik.org/docs/overview>. Acesso em: 1 maio 2020.

FORTUNO. **Fortuno**. 2019. Disponível em: <https://fortuno.app/>. Acesso em: 30 nov. 2019.

FREEMAN, E. **Use a cabeça: padrões e projetos**. 2. ed. rev. Rio de Janeiro: Alta Books, 2009

GAMMA, E. et al. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos.** Porto Alegre: Bockman, 2000.

GIT. **Documentation**. 2020 Disponível em: <https://git-scm.com/docs/git/>. Acesso em: 22 nov. 2019.

GUIABOLSO. **Escolhas inteligentes pro seu dinheiro**. 2019. Disponível em: < https://www.guiabolso.com.br>. Acesso em: 30 nov. 2019.

HADLEY, M.; SANDOZ, P. (eds.). **JAX-RS:** JAVA™ API for RESTful Web Services. [S.l.]: Sun Microsystems, 2008. v.1. 25p.

HEUSER, C. A. **Projeto de Banco de Dados**. 4. ed. [S. l.]: Bookman, 1998.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). **Desemprego**. Disponível em <https://www.ibge.gov.br/explica/desemprego.php>. Acesso em: 18 nov. 2019.

JWT. **Introduction to Json Web Tokens**. [201-]. Disponível em: <https://jwt.io/introduction/>. Acesso em: 18 set. 2020.

JANDL JUNIOR, P. **Java:** Guia do Programador Atualizado Java 8. [S. L.]: Novatec, 2015.

LARMAN, C. **Utilizando a UML e Padrões:** Uma Introdução à Análise e ao Projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed., [S. L.]: Bookman, 2007.

LECHETA, R. **Web Services RESTful**. São Paulo: Editora Novatec, 2015.

LOPES, S. **Aplicações mobile híbridas com Cordova e PhoneGap**. São Paulo: Casa do Código, 2016.

MACEDO JÚNIOR, J. S. **A árvore do dinheiro:** guia para cultivar a sua independência financeira. Rio de Janeiro: Elsevier Editora, 2010.

MALMANN, F. **Finanças Pessoais: Quanto, onde e como investir**. 2008. 57 f. Trabalho de Conclusão de Especialização (Especialização em Finanças Empresariais)–Universidade Federal do Rio Grande do Sul - UFRS, Porto Alegre, 2008.

MDN web docs. **JavaScript**. 2019. Disponível em <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Introduction>. Acesso em: 14 set. 2020.

MILANI, A. **MySQL Guia do Programador.** São Paulo: Novatec, 2006.

MINHAS ECONOMIAS**. Gerenciador financeiro online e gratuito**. 2019. Disponível em: <http://minhaseconomias.com.br/>. Acesso em: 30 nov. 2019.

MOBILLS. **Gerenciador Financeiro**. 2019. Disponível em: <https://www.mobills.com.br/>. Acesso em: 30 nov. 2019.

MONEFY. **Personal finance application that makes money management easy**. 2019. Disponível em: <http://www.monefy.me/>. Acesso em: 30 nov. 2019.

NETBEANS. **Netbeans IDE – The Smarter and Faster Way to Code**. [2020?]. Disponível em <https://netbeans.org/features/index.html>. Acesso em: 18 set. 2020.

NODEJS. **Sobre Node.js®**. Disponível em: <https://nodejs.org/pt-br/about/>. Acesso em: 01 maio 2020.

NPMJS. **NPM**. Disponível em: <https://www.npmjs.com/> Acesso em: 1 maio 2020.

_____. **YUP**. 2020. Disponível em: <https://www.npmjs.com/package/yup/>. Acesso em: 22 nov. 2019.

NUBANK. **Regra 50-15-35: ela pode ajudar a juntar dinheiro?**. 2019. Disponível em <https://blog.nubank.com.br/regra-50-15-35-financas/>. Acesso em 15 de agosto de 2020.

ORACLE. **Java SE Development Kit 8 Downloads**. Disponível em: <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html> Acesso em: 01 maio 2020.

OGLIO, M. D. **Aplicativo Android para o Ambiente Univates Virtual**. 2013. 30p. Monografia (Bacharelado Engenharia da Computação)–Centro Universitário Univates, Lajeado, 2013.

ORGANIZZE. **Controle pessoal online, fácil de usar e seguro**. 2019. Disponível em: <https://www.organizze.com.br/>. Acesso em: 30 nov. 2019.

POSTMAN. **The Postman API Platform**. 2020. Disponível em: <https://www.postman.com/api-platform/>. Acesso em: 18 set. 2020.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: uma abordagem profissional.** 8. ed. Porto Alegre: AMGH, 2016.

PRIKLADNICKI, R. **Métodos ágeis para desenvolvimento de software**. Porto Alegre: Bookman, 2014. 289 p.

REACT NATIVE. **AsyncStorage**. Disponível em: <https://reactnative.dev/docs/asyncstorage>. Acesso em 14 set. 2020.

_____. **Setting up the development environment**. Disponível em: <https://reactnative.dev/docs/environment-setup#docsNav>. Acesso em 01 maio 2020.

RIBEIRO, A. P. **Quando se trata de educação financeira, Brasil fica mal na foto.** 2016. Disponível em: <:https://oglobo.globo.com/economia/negocios/quando-se-trata-de-educacao-financeira-brasil-fica-mal-na-foto-20385966>. Acesso em: 20 set. 2019.

RIBEIRO, R. D.; RIBEIRO, H. C. S. **Gerenciamento de Projetos com Métodos Ágeis.** 1. ed. Rio de Janeiro: [S.n.], 2015.

SILVA, W. C. S. **Aplicações Móveis Nativas com React Native e Firebase:** Um Estudo de Caso. 2018. 73p. Monografia (Bacharelado Ciência da Computação)–Universidade Federal do Maranhão – UFMA, São Luís, 2018.

SOUPUI. **SOAP vs REST 101:** Understand The Differences. [2020]. Disponível em: <https://www.soapui.org/learn/api/soap-vs-rest-api/>. Acesso em: 18 set. 2020.

SOMMERVILLE, I. **Software Engineering.** 9. ed. [S. l.]: Pearson Education, 2011.

STATCOUNTER, G. **Mobile Operating System Market Share in Brazil - September 2019.** 2019. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide/2019>. Acesso em 29 out. 2019.

TANENBAUM, A. S., WETHERALL J.: **Redes de Computadores**, Editora Campus, 5a Edição: Pearson Education, 2011.

TRELLO. **O que é Trello?.** Disponível em: <https://trello.com/b/gkjYO7qu/conheça-o-trello-seu-novo-jeito-de-monitorar-times-tarefas-e-projetos>. Acesso em 01nov. 2019.

WARREN, B. **Os ensaios de Warren Buffett: lições para investidores e administradores**. Rio de Janeiro: R.T.S. Rebouças, 2005.

APÊNDICE A – REQUISITOS DO SISTEMA

1.1 REQUISITOS FUNCIONAIS

Login

IDENTIFICADOR: RF0001	**Requisito: Realizar *login***
O usuário deve entrar no aplicativo a partir do *e-mail* e senha cadastrados.	

IDENTIFICADOR: RF0002	**Requisito: Recuperar senha**
O usuário pode solicitar a modificação de senha clicando em um botão “Recuperar Senha”. Ao selecionar essa opção será enviado uma mensagem ao e-mail cadastrado com uma nova senha, que o usuário poderá alterar ao logar novamente no aplicativo.	

IDENTIFICADOR: RF0003	**Requisito: Realizar *logout***
Enquanto conectado no aplicativo o usuário pode se desconectar clicando na opção “Sair” e retornar para a página de *login*.	

IDENTIFICADOR: RF0004	**Requisito: Cadastrar novo perfil**
O usuário cadastra-se no aplicativo registrando: nome completo, e-mail, foto, gênero, data de nascimento, senha e confirmar senha.	

IDENTIFICADOR: RF0005	**Requisito: Visualizar informações principais**
Após realizar l*ogin,* o usuário consegue visualizar saldo total (soma das receitas - despesa), a comparação entre receita e despesa ao longo do ano (comparativo visual mês a mês), top 3 despesas (as 3 subcategorias de despesas nas quais o usuário possui maiores gastos no mês) e por último um gráfico que mostra o valor total que o usuário já gastou, desse total é apresentado as porcentagens de cada categoria que compõe esse valor.	

Contas

IDENTIFICADOR: RF0006	Requisito: Cadastrar conta
O usuário pode cadastrar uma nova conta informando descrição, tipo de conta (dinheiro, conta corrente, conta poupança, cartão de crédito, outro e investimento) e cor. Sub requisitos: RF008, RF0009, RF0010	

IDENTIFICADOR: RF0007	Requisito: Excluir conta
O usuário pode excluir uma conta cadastrada. Ao excluir uma conta, todas as informações vinculadas à essa conta serão excluídas (receitas e despesas). Deve ser apresentada uma confirmação de exclusão.	

IDENTIFICADOR: RF0008	Requisito: Editar conta
O usuário pode editar as seguintes informações de uma conta: descrição, tipo de conta, valor e cor.	

Despesas

IDENTIFICADOR: RF0009	Requisito: Cadastrar nova despesa
O usuário pode cadastrar uma nova despesa informando sua descrição, o valor, data do lançamento da despesa, categoria (gastos essenciais ou estilo de vida), subcategoria (alimentação, educação, transporte) e deve vincular ela a conta à qual pertence. Assim como optar por parcelamento ou repetição periódica.	

IDENTIFICADOR: RF0010	Requisito: Editar despesa
Ao selecionar uma despesa o usuário pode editar as seguintes informações: o valor, descrição, subcategoria, conta pertencente, data do lançamento da despesa e opção de lançamento periódico. O usuário pode alterar a categoria principal (Gastos essenciais, estilo de vida e investimento), na qual a subcategoria dessa despesas está cadastrada para que os gastos totais de cada categoria estejam encaixados na regra 50/35/15.	

IDENTIFICADOR: RF0011	Requisito: Excluir despesa
O usuário pode excluir uma despesa ao selecionar essa opção. Deve ser apresentada uma mensagem de confirmação de exclusão.	

Receitas

IDENTIFICADOR: RF0012	Requisito: Cadastrar nova receita
O usuário pode cadastrar uma nova receita informando sua descrição, o valor, categoria, data lançamento da receita, e deve vincular a ela, a conta à qual pertence. Além da possibilidade de configurar o lançamento da receita de forma periódica.	

IDENTIFICADOR: RF0013	Requisito: Editar receita
Ao selecionar uma receita o usuário pode editar as seguintes informações: o valor, descrição, categoria, conta pertencente, data do lançamento da receita e opção de lançamento periódico.	

IDENTIFICADOR: RF0014	Requisito: Excluir receita
O usuário pode excluir uma receita ao selecionar essa opção. Deve ser apresentada uma mensagem de confirmação de exclusão.	

Metas

IDENTIFICADOR: RF0015	Requisito: Criar meta para despesa
O usuário pode cadastrar um valor para ser gasto nessa subcategoria de despesa, estabelecendo esse valor como meta de limite de gasto.	

IDENTIFICADOR: RF0016	Requisito: Editar meta de despesa
O usuário pode editar a informação de valor de limite de gastos de uma subcategoria.	

IDENTIFICADOR: RF0016	Requisito: Desativar meta de despesa
O usuário pode desativar o uso de determinada subcategoria de despesa na opção de remanejar despesas.	

IDENTIFICADOR: RF0018	Analisar gastos
O sistema deve realizar uma análise da saúde financeira do usuário utilizando a regra 50/35/15 - 50% para gastos essenciais, 35% para estilo de vida e 15% para investimentos - antes de mostrar a opção de remanejar despesas (impor limites). Dessa maneira ele obterá um retorno de como anda a situação financeira dele. Para chegar na análise de gastos é preciso que existam despesas cadastradas e divididas em cada categoria, dessa forma o sistema conseguirá calcular que categoria está extrapolando ou não a sua porcentagem default (regra 50/35/15).	

Relatórios

IDENTIFICADOR: RF0020	Requisito: Visualizar relatório despesas por subcategoria
O usuário pode visualizar o relatório de despesas por categoria em um gráfico. Por default o gráfico seleciona o mês vigente da visualização, porém o aplicativo deve possibilitar a visualização de outros meses em que haja despesas registradas.	

IDENTIFICADOR: RF0021	Requisito: Visualizar relatório despesas por conta
O usuário pode visualizar o relatório despesas por conta em um gráfico. Por default o gráfico seleciona o mês vigente da visualização, porém o aplicativo deve possibilitar a visualização de outros meses em que haja despesas registradas.	

1.2 REQUISITOS NÃO FUNCIONAIS

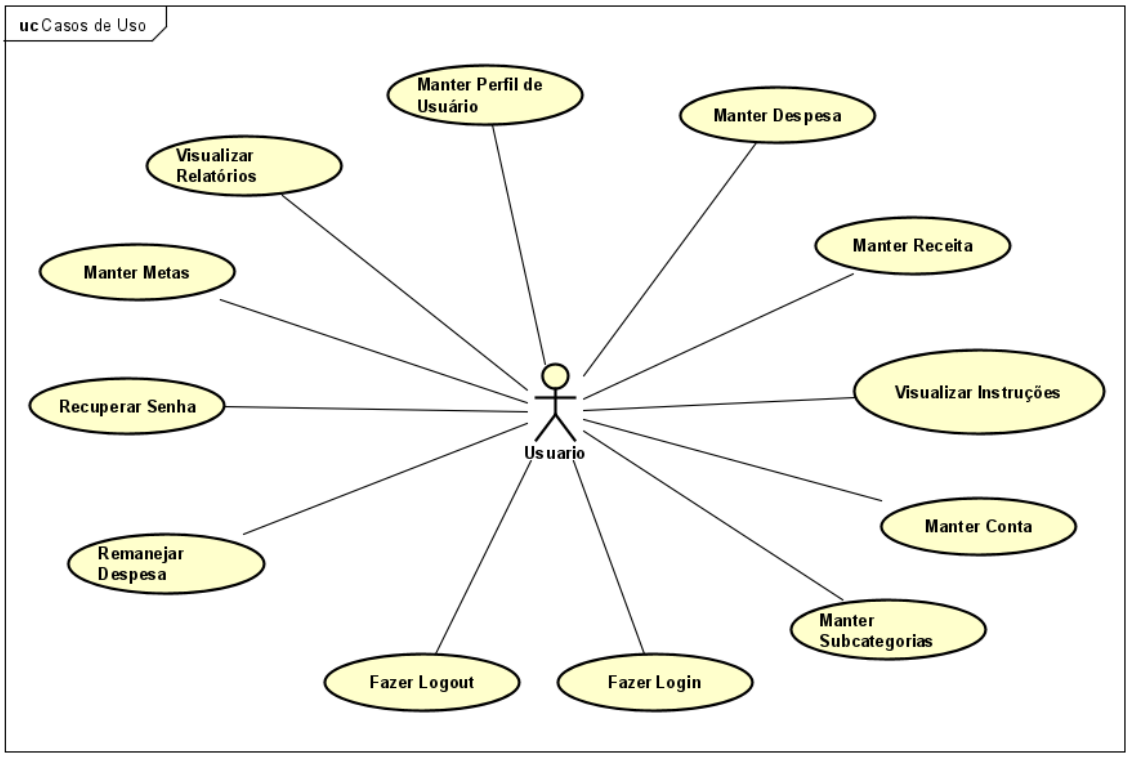
IDENTIFICADOR: RNF0001	**Requisito: Manter dados de login ativos**
Uma vez que o usuário se logou, os dados de login e senha devem ser salvos de maneira que se ele clicar para abrir o app em um determinado intervalo de tempo, não necessite passar suas informações de login.	
Notas adicionais: O usuário permanecerá logado até que : • app seja fechado • usuário se deslogue manualmente • o aplicativo permaneça em estado de hibernação por 5 min, sem nenhuma interação com o usuário.	

1.3 REGRAS DE NEGÓCIO

IDENTIFICADOR: RN0001	**Requisito: Restringir cadastro**
O usuário não poderá cadastrar uma receita ou despesa sem vincular uma conta.	

APÊNDICE B – DIAGRAMA DE CASO DE USOS NÍVEL I

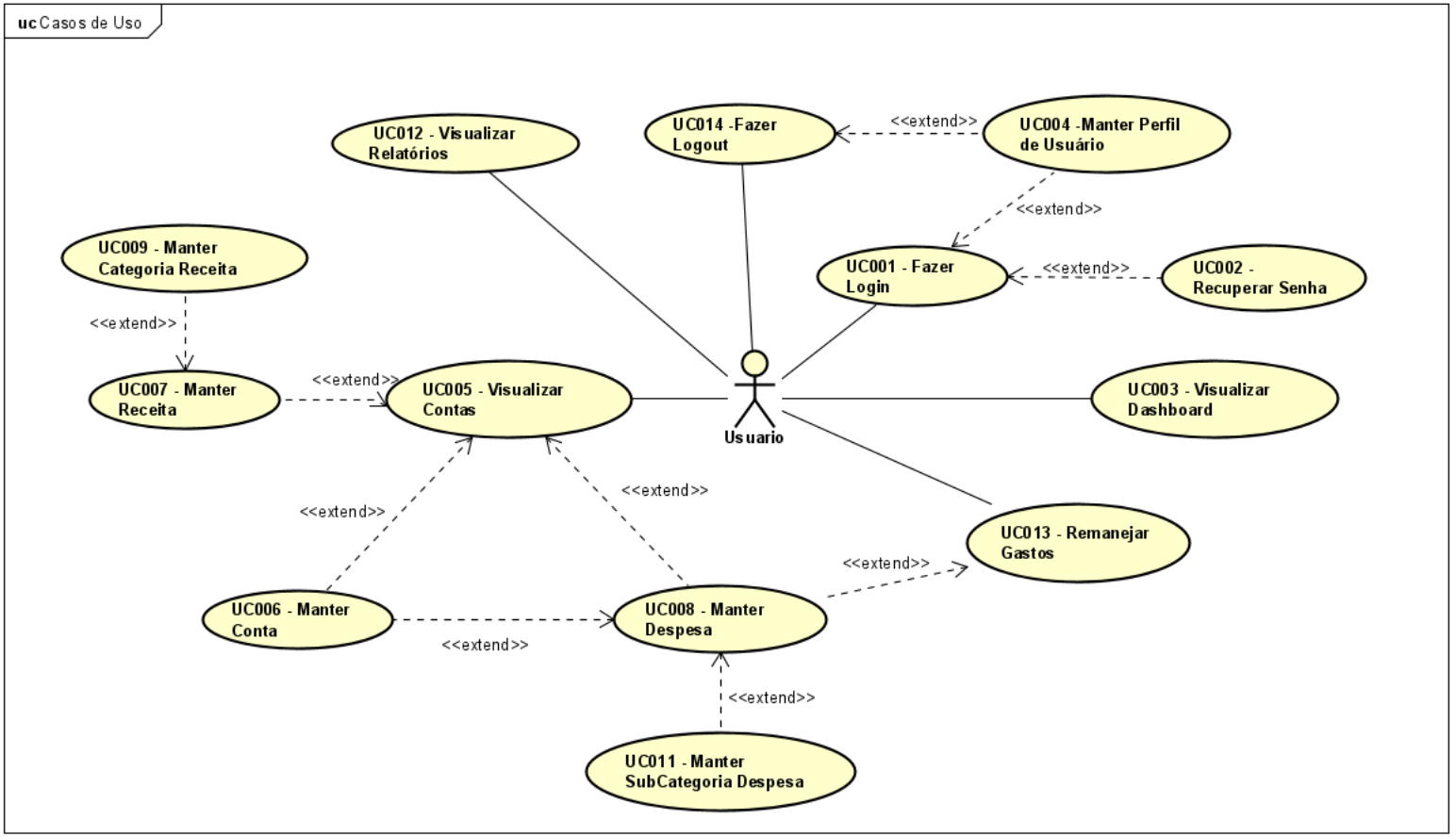
FIGURA 45 – DIAGRAMA DE CASO DE USOS NÍVEL I



FONTE: Os Autores (2019).

APÊNDICE C – DIAGRAMA DE CASO DE USOS NÍVEL II

FIGURA 46 – DIAGRAMA DE CASO DE USOS NÍVEL II



FONTE: Os Autores (2020).

APÊNDICE D – ESPECIFICAÇÕES DE CASO DE USOS

UC001 – Fazer *Login*

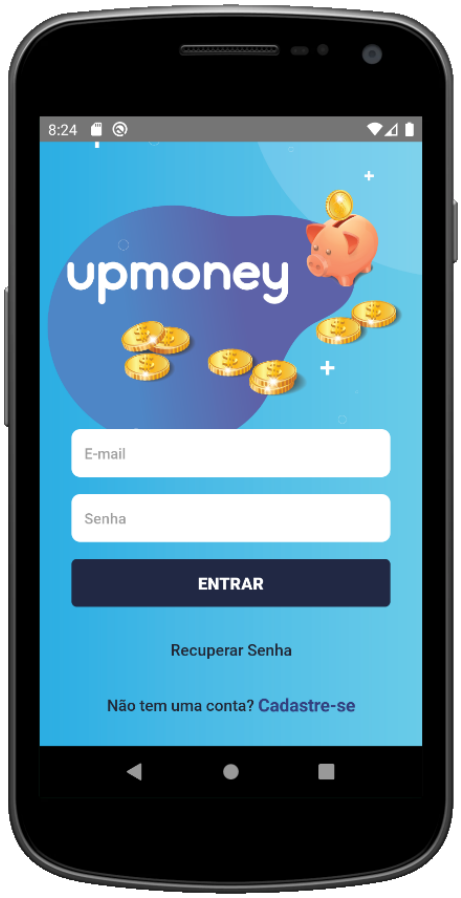
Descrição

Este caso de uso permite o acesso do usuário ao sistema.

Data View

DV1 - Tela Fazer Login.

FIGURA 47 – APP UPMONEY: FAZER LOGIN



FONTE: Os Autores (2020).

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema apresenta a tela (DV1).
2. O usuário insere o usuário e senha (A1) (A2).
3. O usuário pressiona botão Entrar.
4. O sistema verifica os dados enviados (R1) (E1) (E2).
5. O sistema autentica o usuário.
6. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. *Link* Recuperar Senha é pressionado:

1. O sistema chama o Caso de Uso UC002 - Recuperar Senha.
2. O Caso de Uso é Encerrado.

A2. Link Cadastrar-se é pressionado:

1. O sistema chama o Caso de Uso UC004 - Manter Perfil de Usuário com a função = Novo.
2. O Caso de Uso é Encerrado.

Fluxos de Exceção

E1. *Login* inválido:

1. O sistema retorna a mensagem “Usuário ou senha Inválido”.
2. O Caso de Uso é reiniciado.

E2. Campos vazios:

1. O sistema retorna a mensagem “Preencha todos os campos”.
2. O Caso de Uso é reiniciado.

Regras de Negócio

R1. O sistema verifica se usuário e senha do usuário constam no banco de dados e se são compatíveis.

UC002 – Recuperar Senha

Descrição

Este caso de uso permite a recuperação da senha de acesso do usuário.

Data View

DV1 - Tela de Recuperar Senha.

FIGURA 48 – APP UPMONEY: RECUPERAR SENHA

FONTE: Os Autores (2020).

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema apresenta a tela (DV1).
2. O usuário insere o e-mail (A1).
3. O usuário pressiona botão Enviar.
4. O sistema verifica os dados do usuário (E1) (E2) (R1).
5. O sistema envia um e-mail para o usuário (R2).
6. O sistema apresenta a mensagem “E-mail enviado”.
7. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. Botão Voltar é pressionado.

1. O sistema chama o Caso de Uso UC001 - Fazer Login.
2. O Caso de Uso é encerrado.

Fluxos de Exceção

E1. E-mail incorreto ou inexistente:

1. O sistema retorna a mensagem “E-mail incorreto ou inexistente”.
2. O Caso de Uso é reiniciado.

E2. Campos vazios:

1. O sistema retorna a mensagem “Preencha todos os campos”.
2. O Caso de Uso é reiniciado.

Regras de Negócio

R1. O sistema verifica se o e-mail informado é igual ao e-mail cadastrado no banco de dados.

R2. O e-mail enviado contém um link para uma página, na qual o usuário cadastra uma nova senha.

UC003 – Visualizar Dashboard

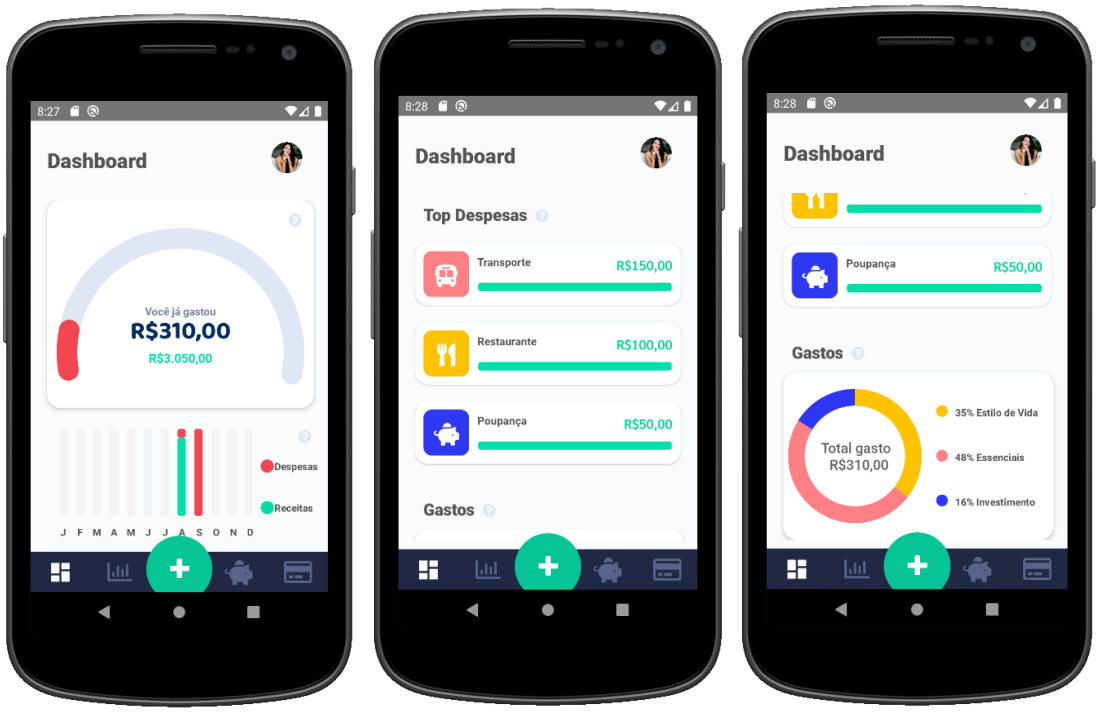
Descrição

Este caso de uso permite a visualização do Dashboard.

Data View

DV1 - Tela de Dashboard.

FIGURA 49 – APP UPMONEY DASHBOARD



FONTE: Os Autores (2020).

Ator Primário

Usuário

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Fluxo de Eventos Principal

1. O sistema apresenta a tela (DV1).
2. O usuário visualiza o Dashboard (R1).
3. O Caso de Uso é encerrado.

Regras de Negócio

R1. O sistema mostra informações de saldo, top despesas com as três subcategorias com as maiores despesas mensais, comparação receita e despesa em um período anual (mostrado mês a mês), gráfico com os gastos conforme categoria.

UC004 – Manter Perfil de Usuário

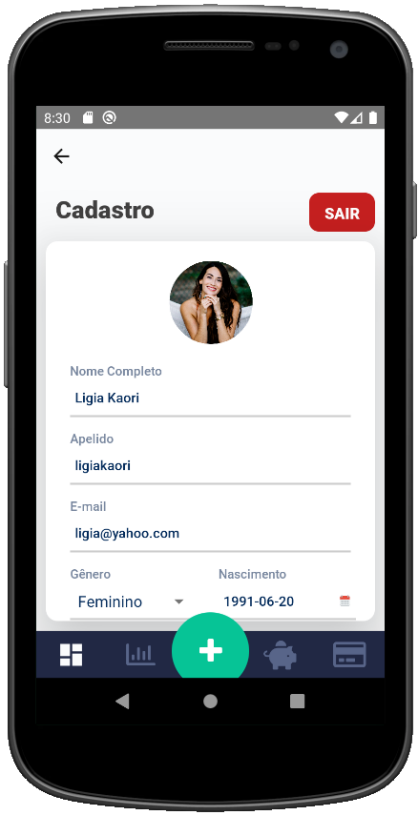
Descrição

Este caso de uso permite o cadastro de um novo usuário ou a alteração de dados por este.

Data View

DV1 - Tela Manter Perfil Usuário

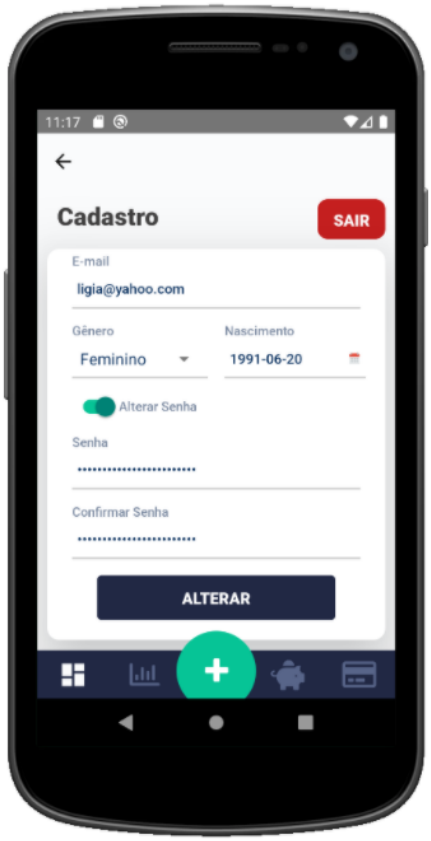
FIGURA 50 – APP UPMONEY: PERFIL USUÁRIO



FONTE: Os Autores (2020).

DV2 - Tela Manter Perfil com Alterar Senha

FIGURA 51 – APP UPMONEY: PERFIL USUÁRIO COM ALTERAR SENHA



FONTE: Os Autores (2020).

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema recebe a função = Novo (A1).
2. O sistema apresenta a tela (DV1).
3. O usuário preenche os dados pessoais.
4. O usuário pressiona botão Enviar.
5. O sistema verifica os dados do usuário (E1) (E2) (R1).
6. O sistema grava os dados do usuário.
7. O sistema apresenta a mensagem “Usuário cadastrado com sucesso”.
8. O sistema chama o Caso de Uso UC001 - Fazer Login.

9. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. Função = Editar

1. O sistema torna o campo e-mail não editável.
2. O sistema apresenta a tela (DV1).
3. O usuário altera os dados desejados.
4. O usuário pressiona o Botão Enviar.
5. O sistema verifica os dados do usuário (E1) (E2) (R1).
6. O sistema grava as alterações.
7. O sistema exibe a mensagem "Suas alterações foram realizadas com sucesso".
8. O sistema retorna à página anterior.
9. O Caso de Uso é encerrado.

A2. *Switch* “Alterar Senha” é selecionado:

1. O usuário altera campos de senha.
2. O usuário pressiona botão Alterar(R2).
3. O sistema verifica os dados enviados (E1).
4. O sistema grava os dados enviados.
5. O Caso de Uso é encerrado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem “Preencha os campos obrigatórios”.
2. O sistema retorna a tela (DV1).

E2. E-mail já cadastrado:

1. O sistema retorna a mensagem “O e-mail informado já está cadastrado”.

Regras de Negócio

R1. Não deve ser permitido em um novo cadastro informar um e-mail já cadastrado.

UC005 – Visualizar Contas

Descrição

Este caso de uso permite visualizar detalhes de contas.

Data View

DV1 - Tela Visualizar Conta

FIGURA 52 – APP UMONEY: VISUALIZAR CONTA

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema busca informações de contas, receitas e despesas.
2. O sistema preenche o combo de meses (R1).
3. O sistema calcula o saldo de todas as contas (R2).
4. O sistema apresenta a tela (DV1).
5. O usuário desliza o carrossel de cartões de contas para esquerda (A1) (A2) (A3).
6. O usuário escolhe um mês no combo.
7. O usuário clica no ícone de atualizar.
8. O sistema preenche as listas de despesas e receitas da conta selecionada.
9. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. Ícone de editar é pressionado.

1. O sistema chama o caso de uso UC006 - Manter Conta com a função = Alterar e os dados da conta.
2. O Caso de Uso é encerrado.

A2. O link Despesa é pressionado

1. O sistema busca informações de despesas (R2).
2. O sistema carrega lista de despesas (A4) (A6).
3. O caso de uso é encerrado.

A3. O link Receita é pressionado

1. O sistema busca informações de receitas (R2).
2. O sistema carrega lista de receitas (A5) (A6).
3. O caso de uso é encerrado.

A4. Um item da lista despesa é pressionado.

1. O sistema chama o caso de uso UC008 - Manter Despesa com a função = Alterar e os dados da despesa.
2. O caso de uso é encerrado.

A5. Um item da receita é pressionado.

1. O sistema chama o caso de uso UC007 - Manter Receita com a função = Alterar e os dados da receita.
2. O caso de uso é encerrado.

A6. Ícone “x” na lista de despesas é pressionado.

1. O sistema mostra um pop-up de confirmação com a mensagem "Tem certeza que deseja deletar essa informação?".
2. O usuário pressiona "OK".
3. O sistema deleta a informação.
4. O sistema grava os dados alterados.
5. O caso de uso é reiniciado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem “Preencha todos os campos”.
2. O Caso de Uso é reiniciado.

Regras de Negócio

R1. Por default o combo de meses deve trazer o mês atual selecionado.

R2. O saldo total é obtido a partir da soma de todas as receitas subtraído da soma de todas as despesas do mês vigente.

R3. As informações são buscadas conforme dados da conta passada.

UC006 – Manter Contas

Descrição

Este caso de uso permite adicionar ou editar uma conta.

Data View

DV1 - Tela Manter Conta

FIGURA 53 – APP: MANTER CONTA

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema recebe a função = Novo (A1).
2. O sistema busca informações de tipos de contas.
3. O sistema preenche o combo tipo de conta.
4. O sistema apresenta a tela (DV1).
5. O usuário preenche nome da conta.
6. O usuário seleciona cor da conta.
7. O usuário seleciona tipo de conta.
8. O sistema apresenta o campo “Valor”.
9. O usuário preenche o valor da conta.
10. O usuário pressiona botão Cadastrar (R1).
11. O sistema verifica os dados enviados. (E1)
12. O sistema grava os dados enviados.
13. O sistema apresenta a mensagem “Conta cadastrada com sucesso”.
14. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. O sistema recebe a função = Alterar

1. O sistema preenche a tela conforme dados recebidos do caso de uso UC005 - Visualizar Contas.
2. O sistema apresenta a tela (DV1).
3. O usuário altera os dados desejados.
4. O usuário pressiona o botão Alterar (R1).
5. O sistema verifica os dados enviados. (E1)
6. O sistema grava os dados enviados.
7. O sistema apresenta a mensagem “Conta alterada com sucesso”.
8. O Caso de Uso é encerrado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem “Preencha os campos obrigatórios”.
2. O sistema retorna a tela (DV1).

Regras de Negócio

R1. O botão “cadastrar” recebe o nome de cadastrar para a função = Novo e o nome de Alterar para a função = Alterar.

UC007 – Manter Receita

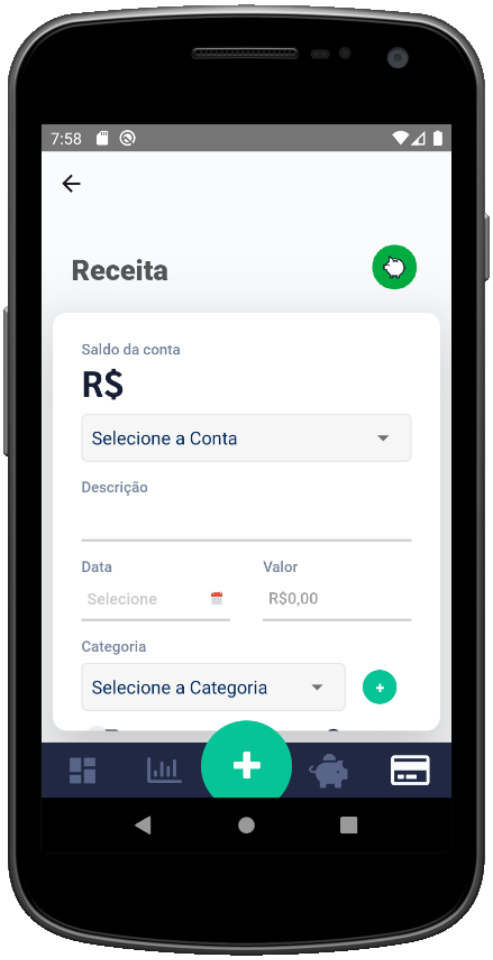
Descrição

Este caso de uso permite adicionar ou editar uma receita.

Data View

DV1 - Tela Manter Receita

FIGURA 54 – APP UPMONEY: MANTER RECEITA



FONTE: Os Autores (2020).

DV2 - Tela Manter Receita com Repetições

FIGURA 55 – APP UPMONEY: MANTER RECEITA COM REPETIÇÕES

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema recebe a função = Novo (A1).
2. O sistema verifica se o usuário possui pelo menos um tipo de conta cadastrado (A2) (R1).

3. O sistema preenche o combo de contas do usuário.
4. O sistema preenche o combo Categoria.
5. O sistema apresenta a tela (DV1).
6. O usuário seleciona a conta desejada.
7. O usuário preenche os dados de receita.
8. O usuário seleciona a categoria desejada (A3) (A4)(A5).
9. O usuário pressiona botão Cadastrar (R2).
10. O sistema verifica os dados enviados (E1).
11. O sistema grava os dados enviados.
12. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. O sistema recebe a função = Alterar

1. O sistema preenche a tela conforme dados recebidos do caso de uso UC005 - Visualizar Contas.
2. O sistema apresenta a tela (DV1).
3. O usuário altera os dados desejados (A3) (A4) (A6).
4. O usuário pressiona o botão Alterar (R1).
5. O sistema verifica os dados enviados (E1).
6. O sistema grava os dados enviados.
7. O sistema apresenta a mensagem “Receita alterada com sucesso”.
8. O Caso de Uso é encerrado.

A2. Nenhum tipo de conta foi cadastrado

1. O sistema apresenta a mensagem “Você deve possuir ao menos um tipo de conta cadastrado.”
2. O sistema chama o Caso de Uso UC006 - Manter Conta passando a função = Novo.
3. O caso de uso é encerrado.

A3. Botão adicionar categoria é pressionado:

1. O sistema chama o Caso de Uso UC009 - Pesquisar Categoria Receita.
2. O Caso de Uso é encerrado.

A4. *Switch "*Parcelamento" é selecionado:

1. O sistema preenche combo Tipo de Parcelamento.
2. O sistema mostra campos de parcelamento (DV2).
3. O usuário preenche campos de parcelamento.
4. O usuário pressiona botão Cadastrar (R2)(A6).
5. O sistema verifica os dados enviados (E1).
6. O sistema grava os dados enviados.
7. O Caso de Uso é encerrado.

A5. Botão "?" de informações é pressionado:

1. O sistema apresenta instruções de uso dos campos de repetições e parcelamentos.

A6. *Switch* "Alterar todas as parcelas" é pressionado:

1. O sistema irá recalcular todas as parcelas à frente da que o usuário está alterado.
2. O usuário pressiona o botão "Alterar".
3. O sistema grava os dados enviados.
4. O Caso de Uso é encerrado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem "Preencha os campos obrigatórios os campos".
2. O sistema retorna a tela (DV1).

Regras de Negócio

R1. O usuário deve possuir ao menos um tipo de conta já cadastrado.

R2. O botão Cadastrar recebe o nome de “Cadastrar” para a função = Novo e o nome de “Alterar” para a função = Alterar.

UC008 – Manter Despesa

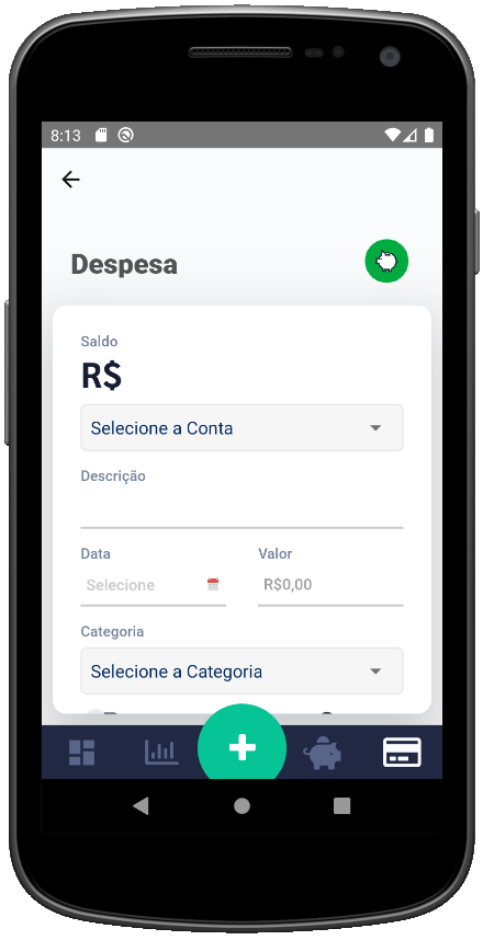
Descrição

Este caso de uso permite adicionar ou editar uma despesa.

Data View

DV1 - Tela Manter Despesa

FIGURA 56 – APP UPMONEY: MANTER DESPESA



FONTE: Os Autores (2020).

DV2 - Tela Manter Despesa com Repetições

FIGURA 57 – APP UPMONEY: MANTER DESPESAS COM REPETIÇÕES

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema recebe a função = Novo (A1).

2. O sistema verifica se o usuário possui pelo menos um tipo de conta cadastrado (A2) (R1).
3. O sistema preenche o combo de contas do usuário.
4. O sistema preenche o combo Categoria.
5. O sistema apresenta a tela (DV1).
6. O usuário seleciona a conta desejada.
7. O usuário preenche os dados de despesa.
8. O usuário seleciona a categoria desejada (A3) (A4)(A5)(A6).
9. O usuário pressiona botão Cadastrar (R2).
10. O sistema verifica os dados enviados (E1).
11. O sistema grava os dados enviados.
12. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. O sistema recebe a função = Alterar

1. O sistema preenche a tela conforme dados recebidos do caso de uso UC005 - Visualizar Contas.
2. O sistema apresenta a tela (DV1).
3. O usuário altera os dados desejados (A3) (A4)(A5).
4. O usuário pressiona o botão Alterar (R1).
5. O sistema verifica os dados enviados. (E1)
6. O sistema grava os dados enviados.
7. O sistema apresenta a mensagem “Despesa alterada com sucesso”.
8. O Caso de Uso é encerrado.

A2. Nenhum tipo de conta cadastrado

1. O sistema apresenta a mensagem “Você deve possuir ao menos um tipo de conta cadastro.”.
2. O sistema chama o Caso de Uso UC006 - Manter Conta.
3. O caso de uso é encerrado.

A3. Botão adicionar Subcategoria é pressionado:

1. O sistema chama o Caso de Uso UC011 - Pesquisar Subcategoria Despesa.
2. O Caso de Uso é encerrado.

A4. *Switch "*Parcelamento" é selecionado:

1. O sistema preenche combo Tipo de Parcelamento.
2. O sistema mostra campos de parcelamento (DV2).
3. O usuário preenche campos de parcelamento.
4. O usuário pressiona botão Cadastrar (R2)(A6).
5. O sistema verifica os dados enviados (E1).
6. O sistema grava os dados enviados.
7. O Caso de Uso é encerrado.

A5. Botão "?" de informações é pressionado:

1. O sistema apresenta instruções de uso dos campos de repetições e parcelamentos.

A6. *Switch* "Alterar todas as parcelas" é pressionado:

1. O sistema irá recalcular todas as parcelas à frente da que o usuário está alterado.
2. O usuário pressiona o botão "Alterar".
3. O sistema grava os dados enviados.
4. O Caso de Uso é encerrado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem "Preencha todos os campos".
2. O Caso de Uso é reiniciado.

Regras de Negócio

R1. O usuário deve possuir ao menos um tipo de conta já cadastrado.

R2. O botão Cadastrar recebe o nome de “Cadastrar” para a função = Novo e o nome de “Alterar” para a função = Alterar.

UC009 – Manter Categorias Receita

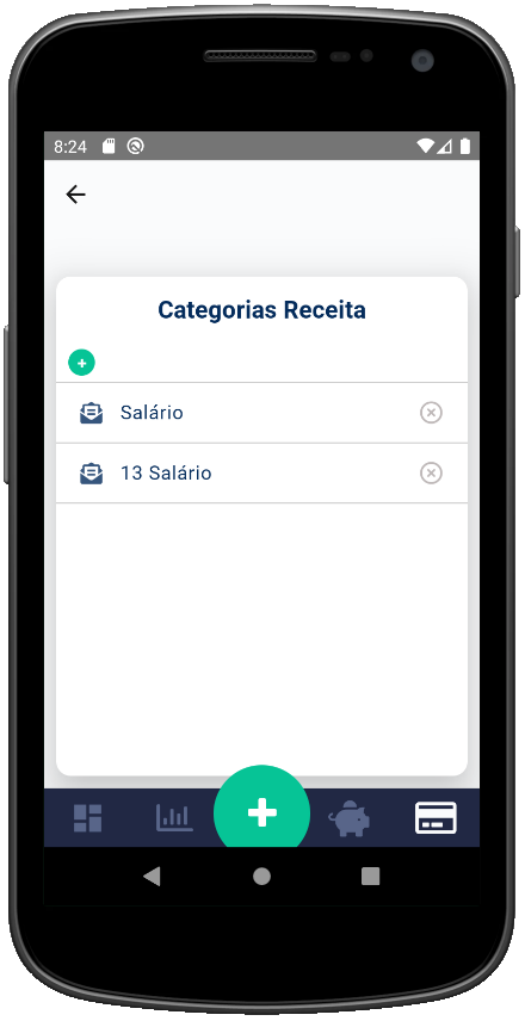
Descrição

Este caso de uso permite visualizar, adicionar ou editar as categorias de receita.

Data View

DV1 - Tela Visualizar Categoria Receita

FIGURA 58 – APP UPMONEY: VISUALIZAR CATEGORIA RECEITA



FONTE: Os Autores (2020).

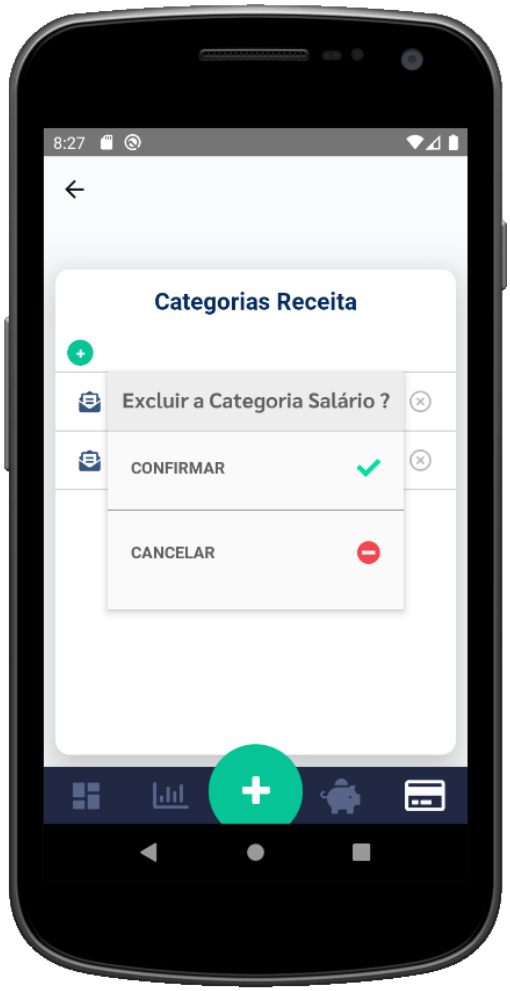
DV2 - Tela Editar Categoria Receita

FIGURA 59 – APP UPMONEY: EDITAR CATEGORIA RECEITA

FONTE: Os Autores (2020.

DV3 - Tela Excluir Categoria Receita

FIGURA 60 – APP UPMONEY: EXCLUIR CATEGORIA RECEITA



FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema busca a lista de categorias de receita do usuário.
2. O sistema preenche a lista de categorias.

3. O sistema apresenta a tela (DV1).
4. O usuário seleciona a categoria desejada (A1) (A2) (A3).
5. O usuário visualiza a categoria.
6. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. Botão adicionar categoria é pressionado:

1. O sistema mostra um pop-up (DV2).
2. O sistema preenche a lista de ícones disponíveis.
3. O usuário preenche os campos.
4. O usuário pressiona o botão “Salvar”.
5. O sistema verifica os dados do usuário. (E1)
6. O sistema grava os dados enviados.
7. O Caso de Uso é encerrado.

A3. Ícone Excluir Categoria é pressionado.

1. O sistema apresenta pop-up para confirmar exclusão (DV3).
2. O usuário pressiona o botão “Confirmar”. (A5)
3. O sistema grava os dados do usuário.
4. O caso de uso é reiniciado.

A4. Item da lista de categoria é pressionado

1. O sistema preenche os campos conforme as informações da categoria selecionada.
2. O sistema mostra um pop-up (DV2).
3. O usuário atualiza as informações.
4. O usuário pressiona o botão “Salvar”.
5. O sistema verifica os dados do usuário (E1).
6. O sistema grava os dados enviados.
7. O Caso de Uso é encerrado.

A5. Botão Cancelar é pressionado

1. O Caso de Uso é reiniciado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem “Preencha os campos obrigatórios”.
2. O Caso de Uso é reiniciado.

UC010 – Manter Subcategorias Despesa

Descrição

Este caso de uso permite visualizar, adicionar ou editar as subcategorias de despesa.

Data View

DV1 - Tela Visualizar Subcategoria Despesa

FIGURA 61 – APP UPMONEY: VISUALIZAR SUBCATEGORIA DESPESA

FONTE: Os Autores (2020).

DV2 - Tela Editar Subcategoria Despesa

FIGURA 62 – APP UPMONEY: EDITAR SUBCATEGORIA DESPESA

FONTE: Os Autores (2020).

DV3 - Tela Excluir Subcategoria Despesa

FIGURA 63 – APP UPMONEY: EXCLUIR SUBCATEGORIA DESPESA

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema busca a lista de subcategorias de despesa do usuário.
2. O sistema preenche a lista de subcategorias, divididas nas categorias principais.

3. O sistema apresenta a tela (DV1).
4. O usuário expande a categoria principal desejada.
5. O usuário seleciona a subcategoria desejada (A1) (A2) (A3).
6. O usuário visualiza a subcategoria.
7. O Caso de Uso é encerrado.

Fluxos Alternativos

A1. Botão adicionar subcategoria é pressionado:

1. O sistema mostra um pop-up (DV2).
2. O sistema preenche o combo de categorias principais.
3. O sistema preenche a lista de ícones disponíveis.
4. O usuário preenche os campos. (R1)
5. O usuário pressiona o botão “Salvar”.
6. O sistema verifica os dados do usuário. (E1)
7. O sistema grava os dados enviados.
8. O Caso de Uso é encerrado.

A3. Ícone excluir subcategoria é pressionado.

1. O sistema apresenta pop-up para confirmar exclusão (DV3).
2. O usuário pressiona o botão “Confirmar”. (A5)
3. O sistema grava os dados do usuário.
4. O caso de uso é reiniciado.

A4. Item da lista de subcategoria é pressionado

1. O sistema preenche os campos conforme as informações da subcategoria selecionada.
2. O sistema mostra um pop-up (DV2).
3. O usuário atualiza as informações.
4. O usuário pressiona o botão “Salvar”.
5. O sistema verifica os dados do usuário (E1).
6. O sistema grava os dados enviados.
7. O Caso de Uso é encerrado.

A5. Botão Cancelar é pressionado

1. O Caso de Uso é reiniciado.

Fluxos de Exceção

E1. Campos vazios:

1. O sistema retorna a mensagem “Preencha os campos obrigatórios”.
2. O Caso de Uso é reiniciado.

Regras de Negócio

R1. O usuário deve selecionar uma categoria principal para cadastrar a subcategoria.

UC012 – Visualizar Relatórios

Descrição

Este caso de uso permite a visualização de relatórios.

Data View

DV1 - Tela visualização de relatórios das despesas das contas.

FIGURA 64 – APP UPMONEY: VISUALIZAÇÃO DE RELATÓRIOS DAS DESPESAS DAS CONTAS

FONTE: Os Autores (2020).

DV2 - Tela relatório subcategoria detalhada

FIGURA 65 – APP UPMONEY: RELATÓRIO SUBCATEGORIA DETALHADA

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema busca informações de contas.
2. O sistema preenche o combo de contas.

3. O sistema preenche o combo de meses (R1).
4. O sistema apresenta a tela (DV1).
5. O usuário seleciona uma conta.
6. O sistema apresenta o relatório das subcategorias dessa conta (A1)
7. O caso de uso é encerrado.

Fluxos Alternativos

A1. A subcategoria é pressionada.

1. O sistema busca informações da subcategoria pressionada.
2. O sistema constrói o relatório detalhado da subcategoria pressionada.
3. O sistema chama a tela (DV2) (A2).
4. O caso de uso é encerrado

A2. Botão Voltar é pressionado.

1. O sistema chama a tela (DV1).
2. O Caso de Uso é encerrado.

UC013 – Fazer Logout

Descrição

Este caso de uso permite realizar o logout da conta.

Data View

DV1 - Tela Fazer Logout.

FIGURA 66 – APP UPMONEY: FAZER LOGOUT

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Fazer Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema apresenta a tela (DV1).
2. O usuário pressiona o botão Sair.
3. O sistema realiza o logout do aplicativo.
4. O sistema chama o caso de uso UC001 - Fazer Login.
5. O Caso de Uso é encerrado.

Fluxos Alternativos

UC014 – Análise de gastos

Descrição

Este caso de uso apresenta uma análise de gastos do usuário, e como está sua saúde financeira dentro da regra 50/35/15.

Data View

DV1 - Tela de visualização da organização de gastos nas categorias principais.

FIGURA 67 – APP UPMONEY: TELA DE VISUALIZAÇÃO DA ORGANIZAÇÃO DE GASTOS

FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Login.
2. O usuário deve ter cadastrado despesas e receitas.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema apresenta a tela (DV1).
2. O sistema faz o cálculo de análise dos gastos do usuário.
3. O sistema apresenta o gráfico demonstrativo e o texto explicativo. (A1)(A2)
4. O Caso de Uso é encerrado.

Fluxos Alternativos

A1:Botão Remanejar Gastos é pressionado.

1. O sistema chama o Caso de Uso UC015 – Remanejar Gastos.
2. O Caso de Uso é encerrado.

A2: Uma fatia do gráfico é selecionado.

1. O sistema busca os gastos que o usuário possui na categoria selecionada.
2. O sistema apresenta os dados.
3. O Caso de Uso é encerrado.

Regras de Negócio

R1. O sistema verifica se o usuário possui despesas e receitas cadastradas. Caso não possua o sistema desativa o botão de “Remanejar Gastos”.

UC015 – Remanejar Gastos

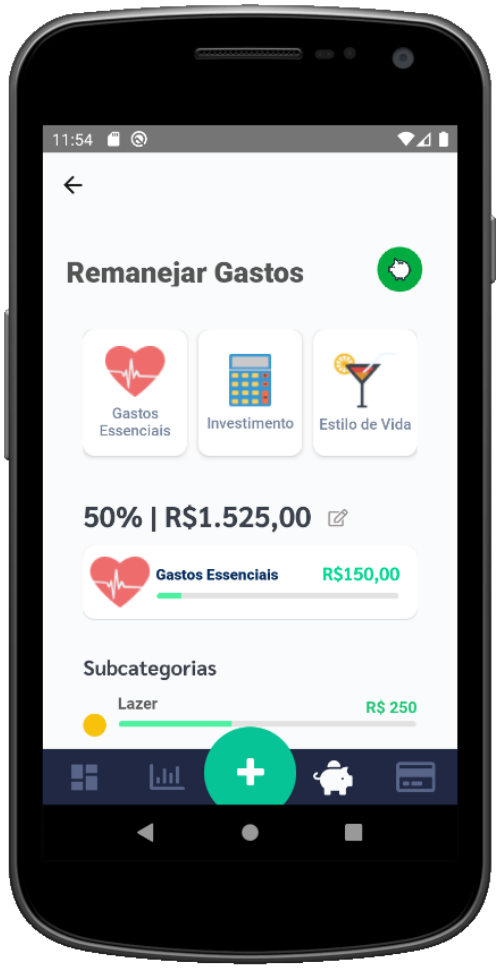
Descrição

Este caso de uso permite alterar os valores limites para cada categoria e subcategoria.

Data View

DV1 - Tela Remanejar Despesas

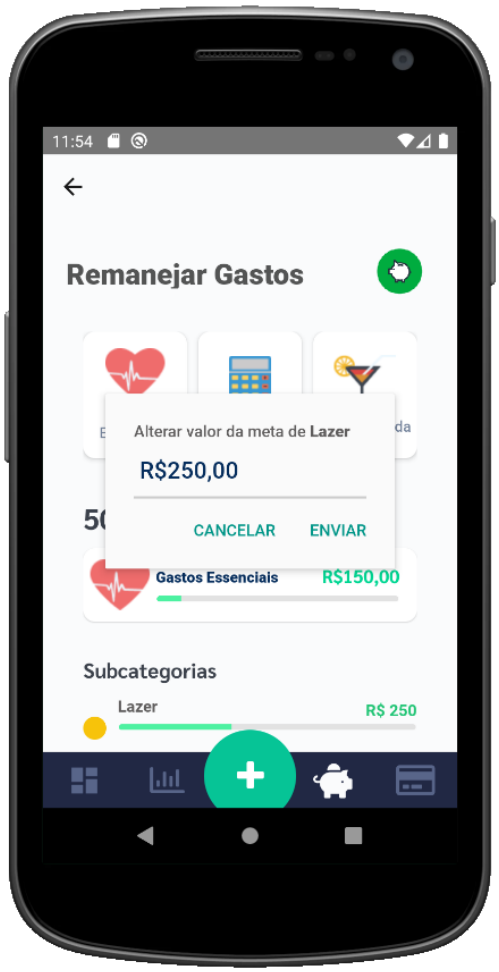
FIGURA 68 – APP UPMONEY: REMANEJAR GASTOS



FONTE: Os Autores (2020).

DV2 - Tela Alterar valor de meta de subcategoria

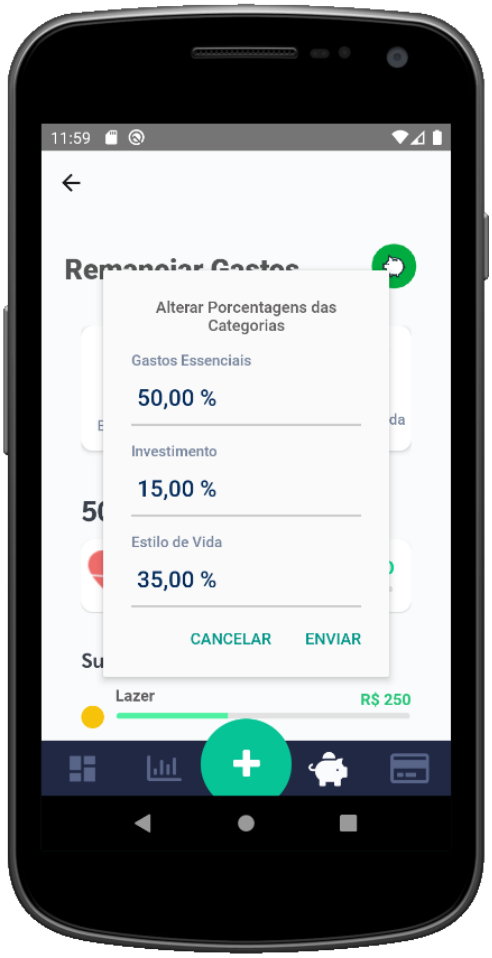
FIGURA 69 – APP UPMONEY: ALTERAR VALOR DE META DE SUBCATEGORIA



FONTE: Os Autores (2020).

DV3 - Tela Alterar porcentagem da categoria principal.

FIGURA 70 – APP UPMONEY: ALTERAR PORCENTAGEM DA CATEGORIA PRINCIPAL



FONTE: Os Autores (2020).

Pré-condições

Este caso de uso pode iniciar somente se:

1. O sistema tiver executado o UC001 – Login.

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema calcula os gastos com base nos cadastros das subcategorias nas categorias principais. (R1)
2. O sistema preenche o valor ideal que deveria ser gasto nas categorias principais e o quanto já foi gasto.
3. O sistema preenche lista de subcategorias.
4. O sistema apresenta a tela (DV1).
5. O usuário clica no menu da subcategoria Lazer (A1) (A2) (A3) (A4).
6. O sistema apresenta o pop-up para preencher o novo valor meta ser preenchido (DV2).
7. O usuário preenche o valor desejado.
8. O usuário pressiona “Confirmar” (A5).
9. O sistema grava os dados.
10. O Caso de Uso é reiniciado.

Fluxos Alternativos

A1. Botão Investimento é pressionado.

1. O sistema busca informações de subcategorias de investimento.
2. O sistema preenche lista de subcategorias.
3. O sistema apresenta a tela (DV1).
4. O usuário seleciona um item da lista de subcategorias de investimento.
5. O sistema apresenta o pop-up para preencher o novo valor meta ser preenchido (DV2).
6. O usuário preenche o valor desejado.
7. O usuário pressiona Confirmar (A5).
8. O sistema grava os dados.
9. O Caso de Uso é reiniciado.

A2. Botão Estilo de Vida é pressionado.

1. O sistema busca informações de subcategorias de estilo de vida.

2. O sistema preenche lista de subcategorias.
3. O sistema apresenta a tela (DV1).
4. O usuário seleciona um item da lista de subcategorias de estilo de vida.
5. O sistema apresenta o pop-up para preencher o novo valor meta ser preenchido (DV2).
6. O usuário preenche o valor desejado.
7. O usuário pressiona Confirmar (A5).
8. O sistema grava os dados.
9. O Caso de Uso é reiniciado.

A3. Botão Gastos Essenciais é pressionado.

1. O sistema busca informações de subcategorias de gastos essenciais.
2. O sistema preenche lista de subcategorias.
3. O sistema apresenta a tela (DV1).
4. O usuário seleciona um item da lista de subcategorias de gastos essenciais.
5. O sistema apresenta o pop-up para preencher o novo valor meta ser preenchido (DV2).
6. O usuário preenche o valor desejado.
7. O usuário pressiona “Confirmar” (A5).
8. O sistema grava os dados.
9. O caso de uso é reiniciado.

A4. Ícone de editar porcentagem da categoria principal é pressionado.

1. O sistema busca as informações das categorias principais.
2. O sistema preenche os campos de porcentagens.
3. O usuário altera alguma porcentagem (R3).
4. O usuário pressiona “Confirmar” (A5).
5. O sistema grava os dados.
6. O caso de uso é reiniciado.

A5. Cancelar é pressionado.

1. O sistema reinicia o Caso de Uso.

Regras de Negócio

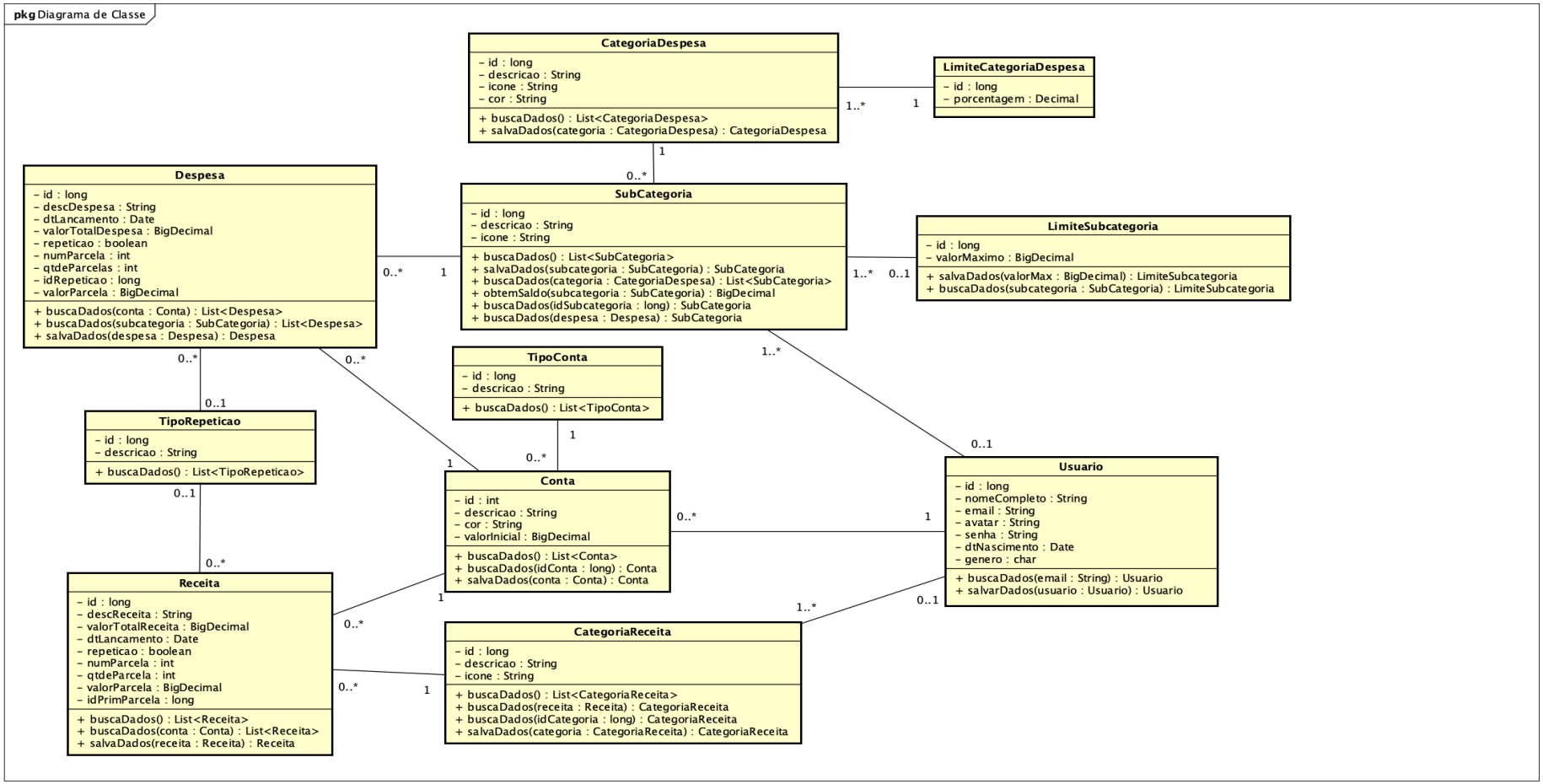
R1. Ao iniciar o Caso de Uso, o sistema deve buscar dados da categoria gastos essenciais para o preenchimento da lista de subcategorias.

R2. Todas as subcategorias já cadastradas pelo usuário serão apresentadas na lista das suas respectivas categorias principais. Se elas não possuírem metas já cadastradas, o sistema considerará essa meta como 100%.

R3. Ao alterar o valor de porcentagem de uma categoria principal, é preciso alterar as 3, para que a soma dos valores seja 100%.

APÊNDICE E – DIAGRAMA DE CLASSES

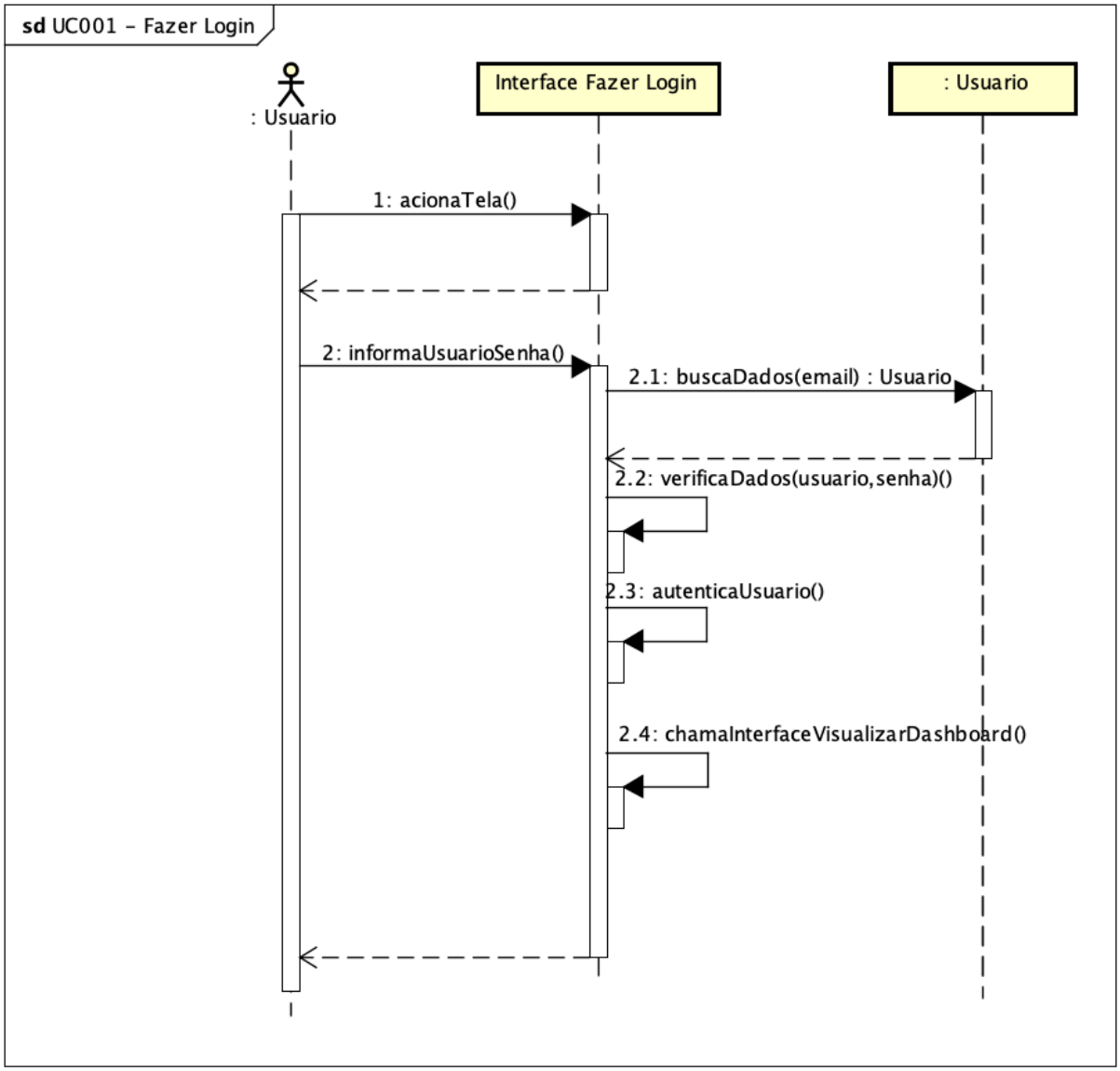
FIGURA 71 – DIAGRAMA DE CLASSES



FONTE: Os Autores (2020).

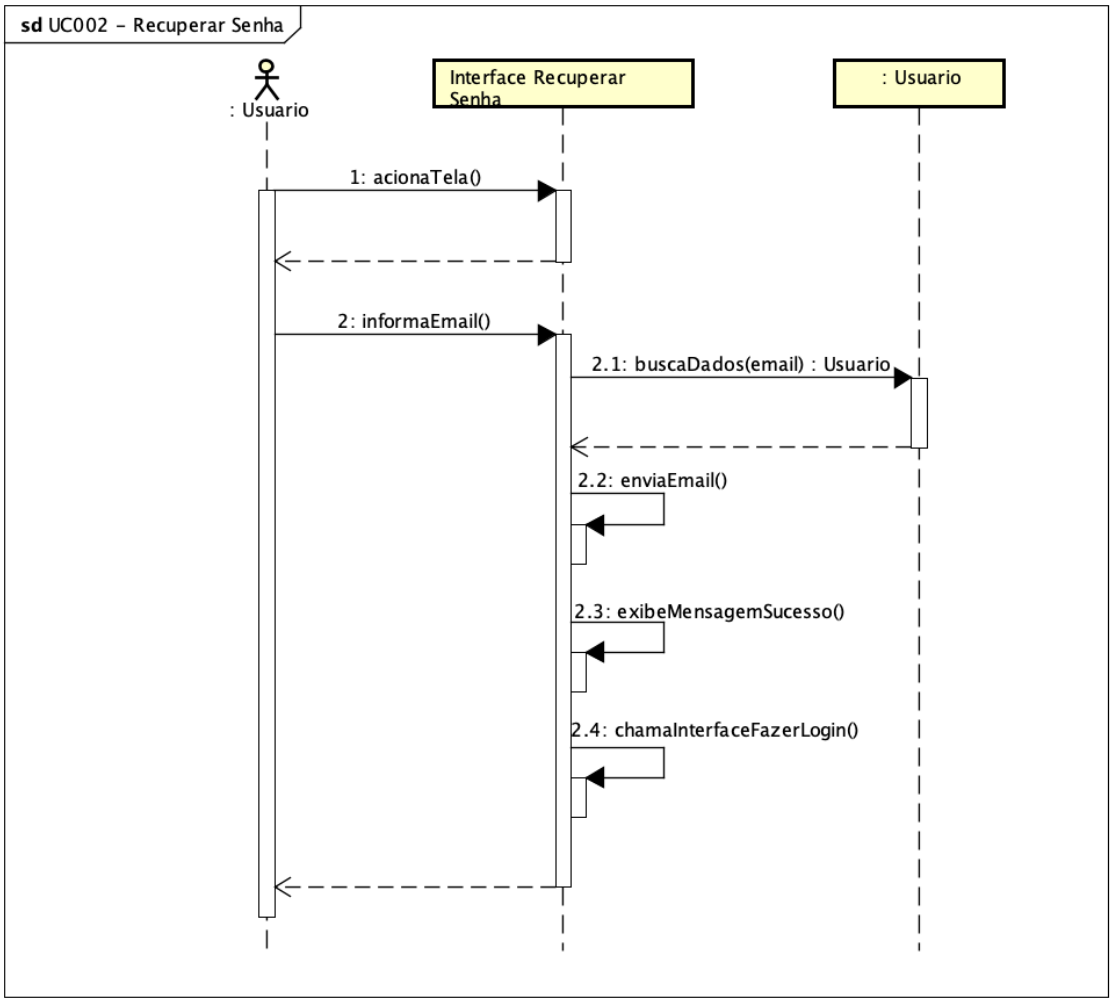
APÊNDICE F – DIAGRAMAS DE SEQUÊNCIA

FIGURA 72 – DIAGRAMA DE SEQUENCIA – FAZER LOGIN



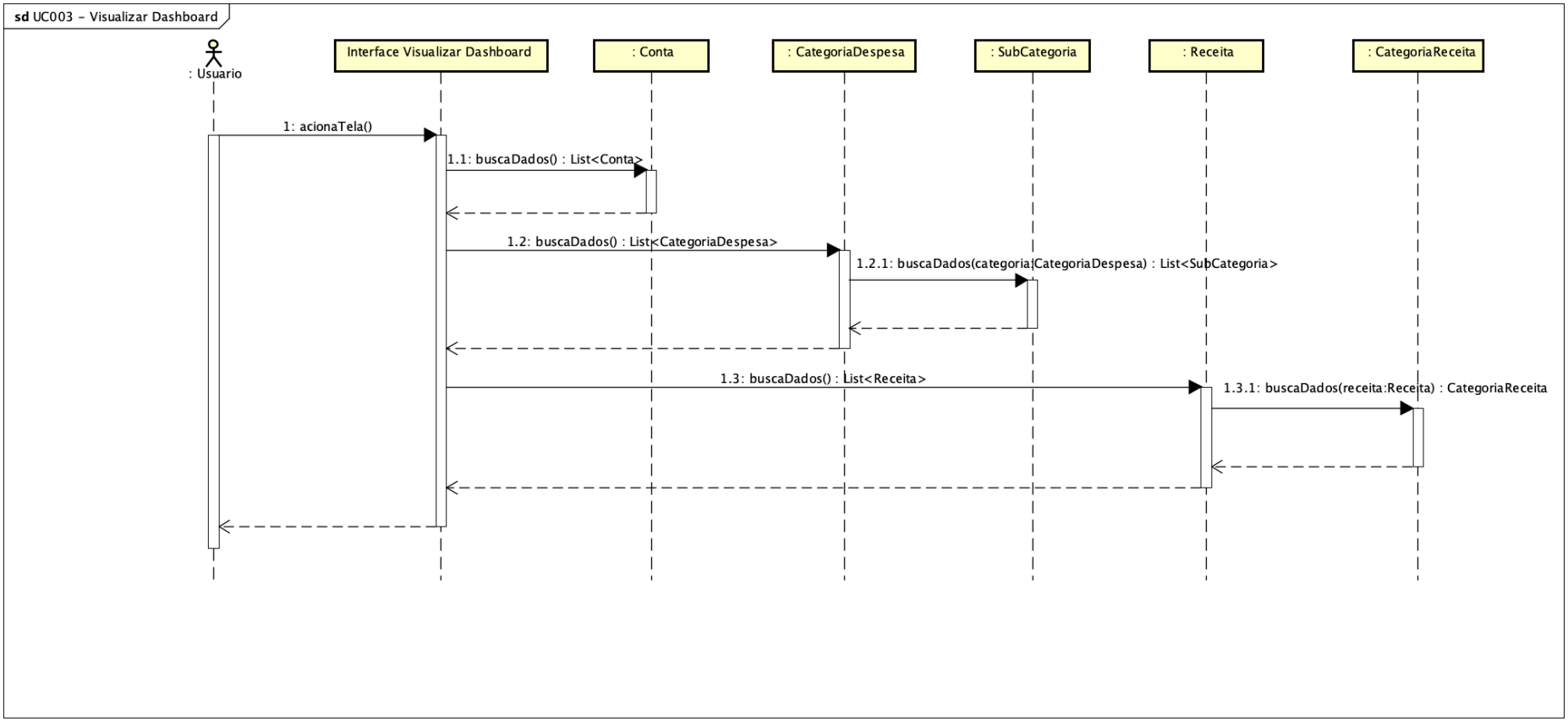
FONTE: Os Autores (2020).

FIGURA 73 – DIAGRAMA DE SEQUENCIA – RECUPERAR SENHA



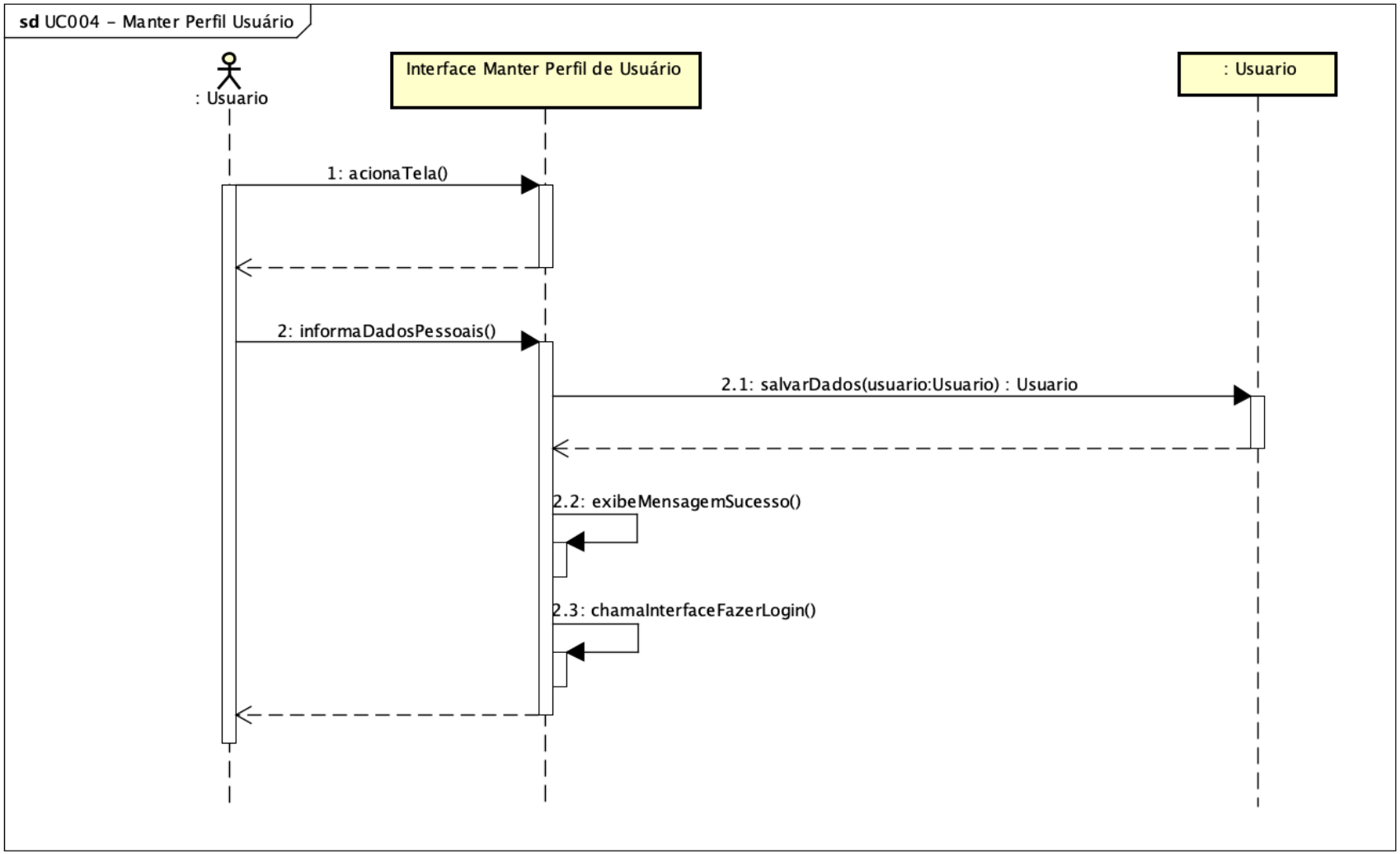
FONTE: Os Autores (2020).

FIGURA 74 – DIAGRAMA DE SEQUENCIA – VISUALIZAR DASHBOARD



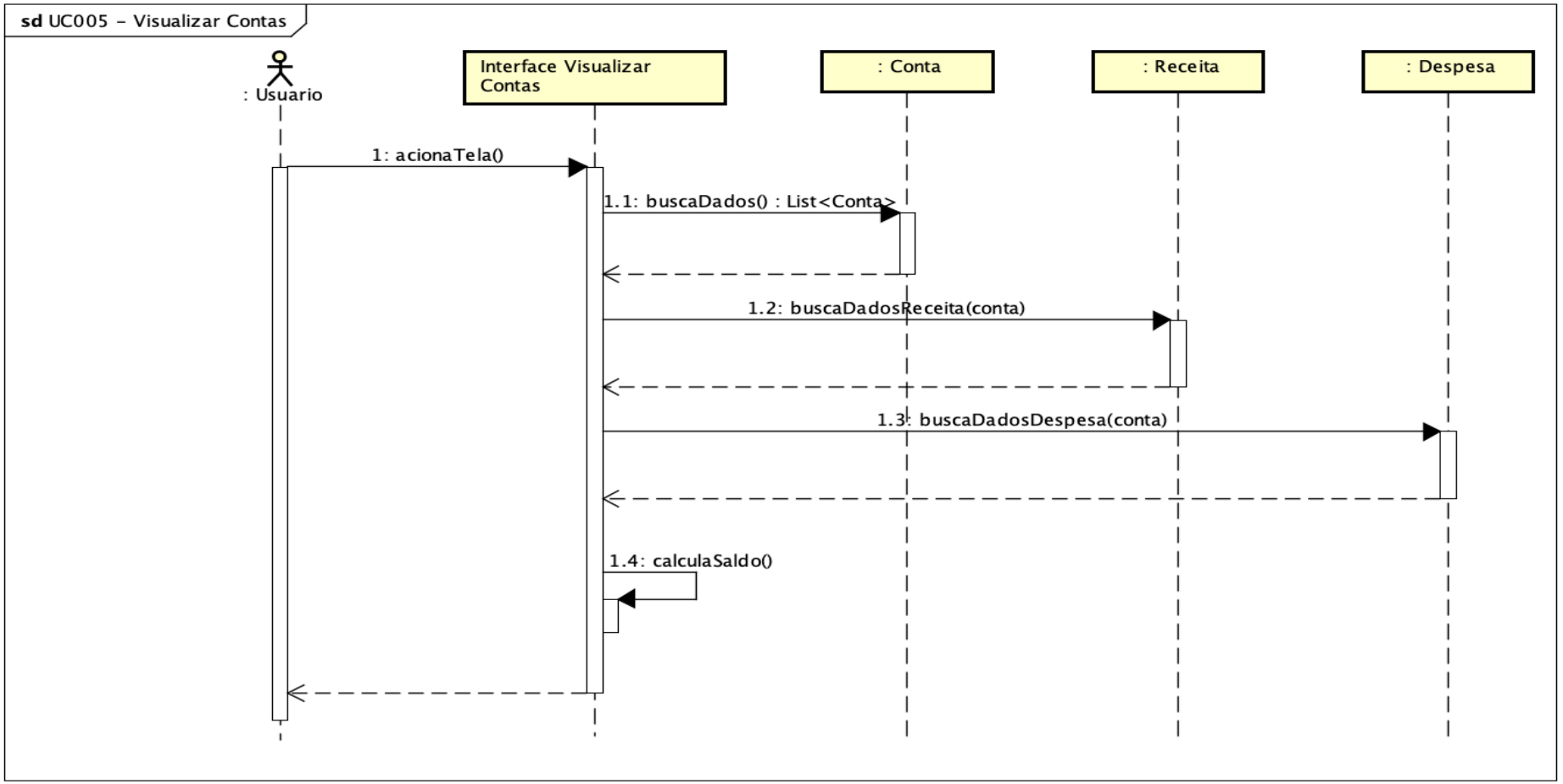
FONTE: Os Autores (2020).

FIGURA 75 – DIAGRAMA DE SEQUENCIA – MANTER PERFIL DE USUÁRIO



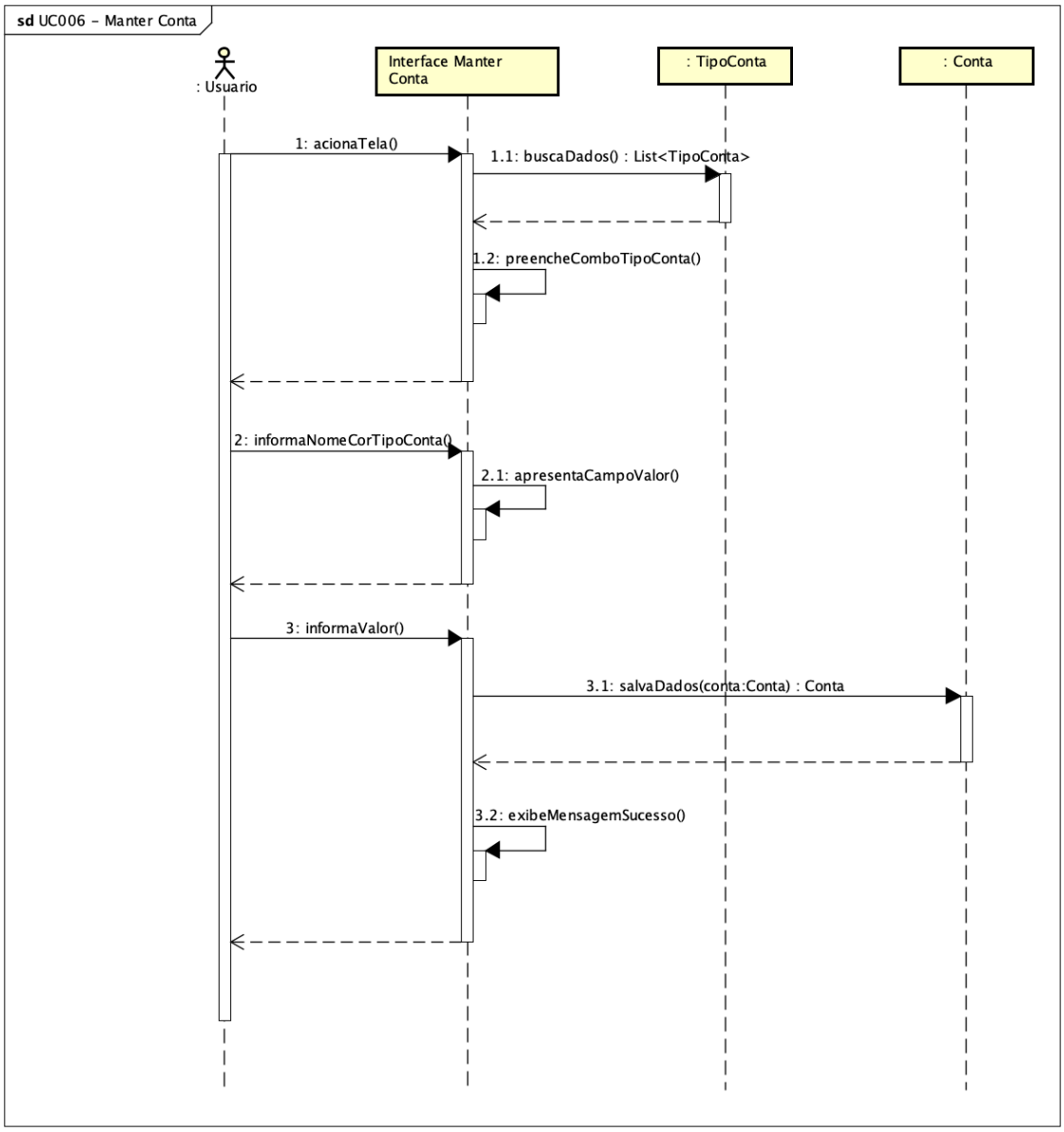
FONTE: Os Autores (2020).

FIGURA 76 – DIAGRAMA DE SEQUENCIA – VISUALIZAR CONTAS



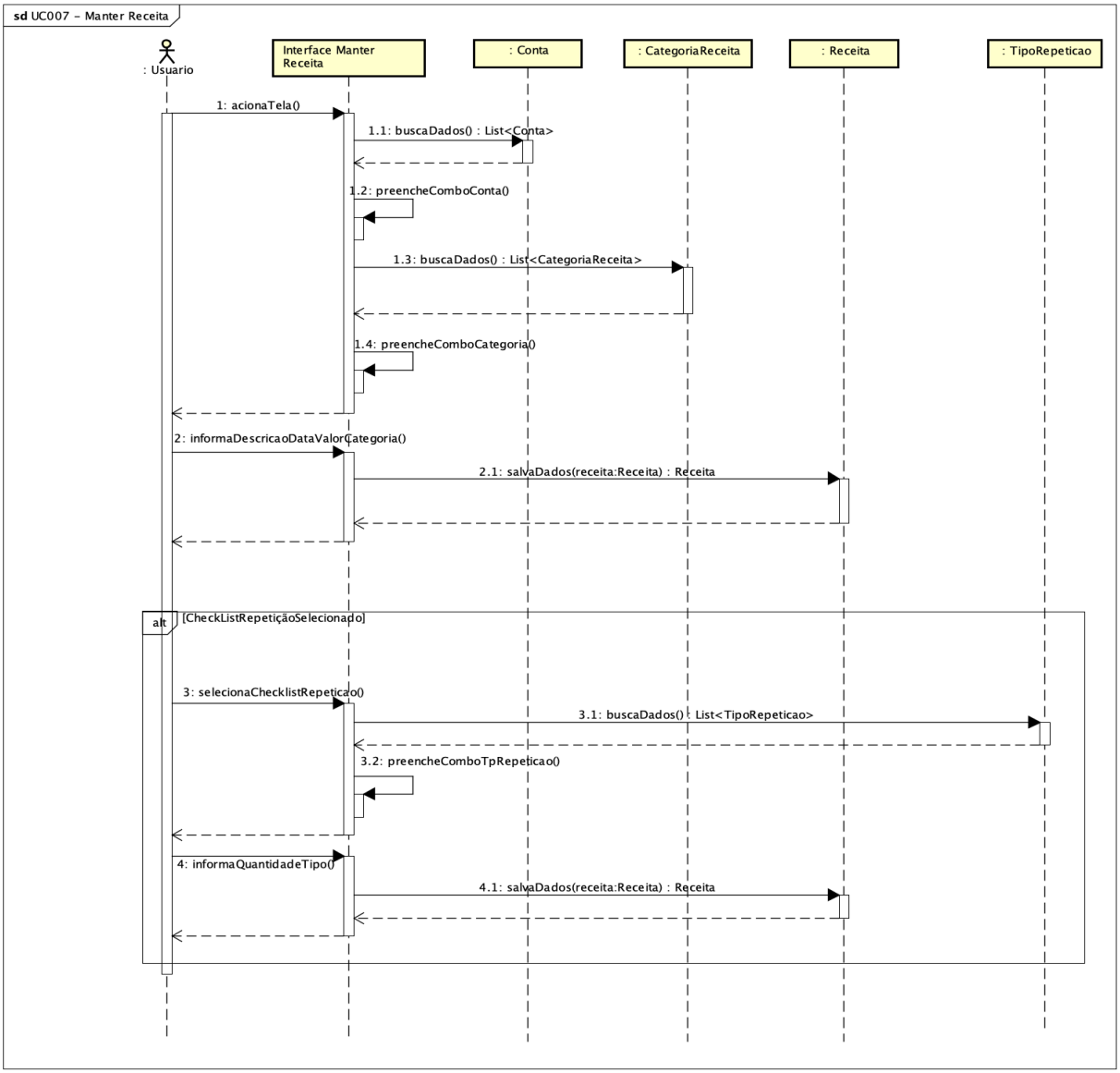
FONTE: Os Autores (2020).

FIGURA 77 – DIAGRAMA DE SEQUENCIA – MANTER CONTA



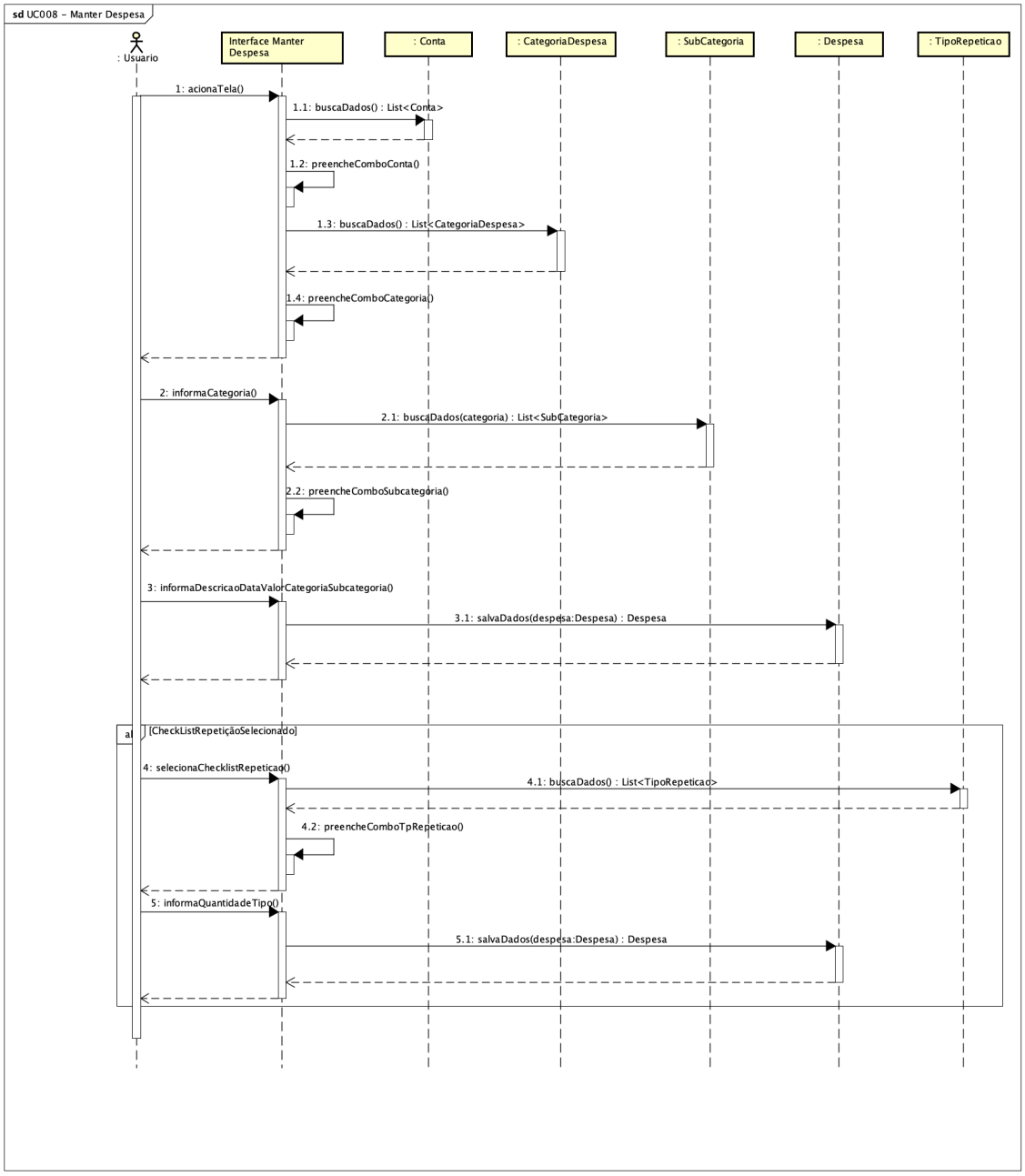
FONTE: Os Autores (2020).

FIGURA 78 – DIAGRAMA DE SEQUENCIA – MANTER RECEITA



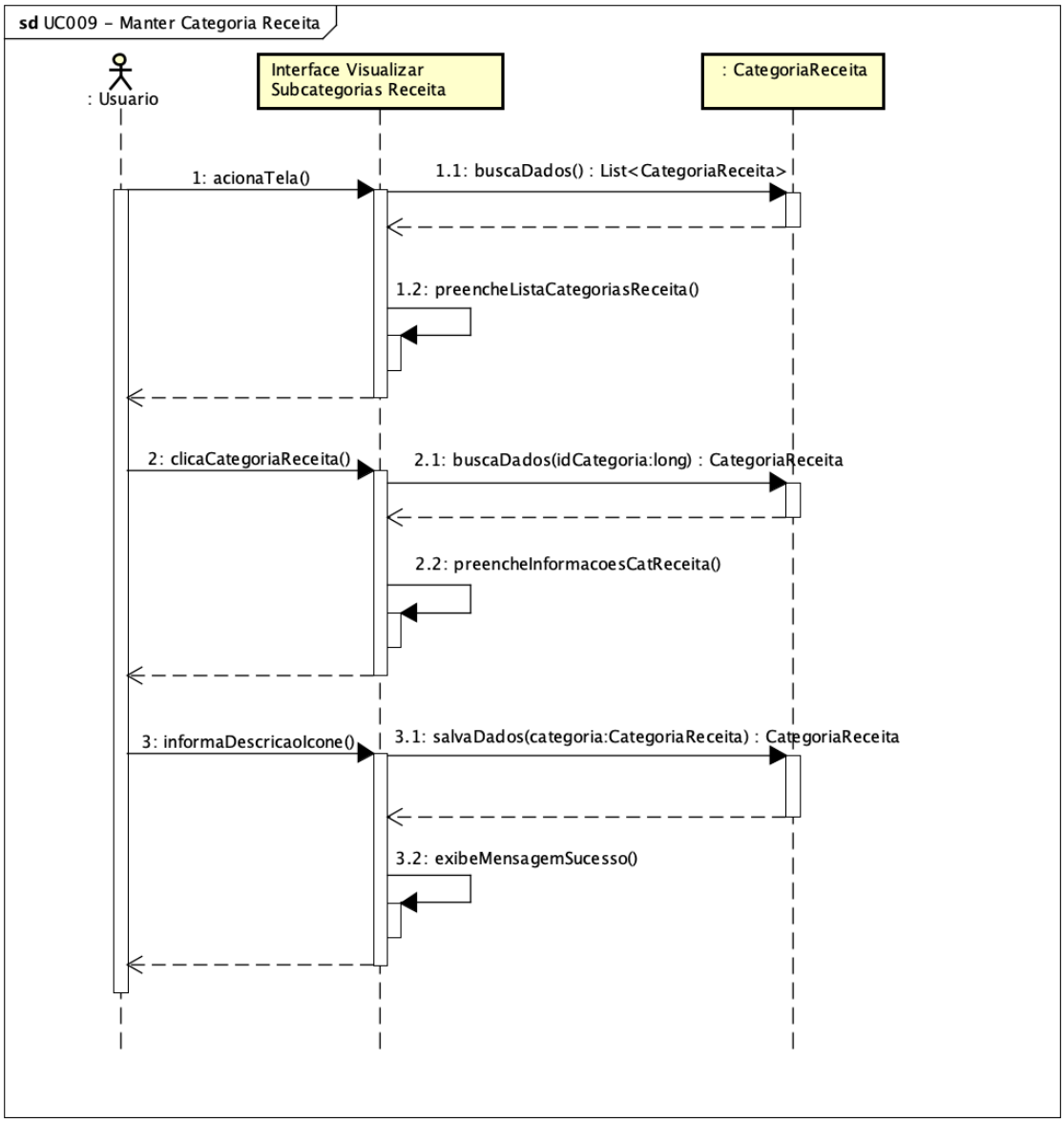
FONTE: Os Autores (2020)

FIGURA 79 – DIAGRAMA DE SEQUENCIA – MANTER DESPESA



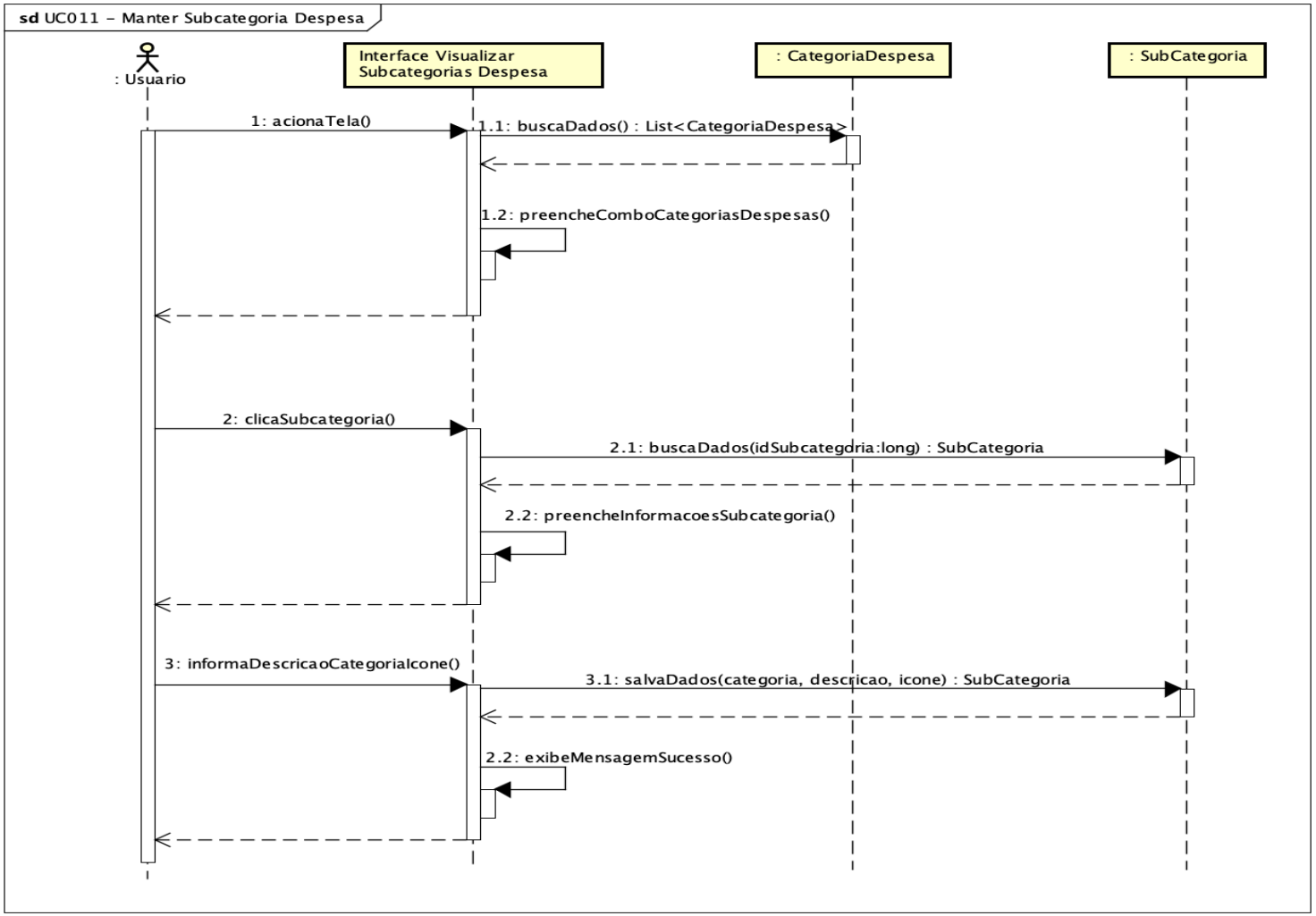
FONTE: Os Autores (2020).

FIGURA 80 – DIAGRAMA DE SEQUENCIA – MANTER CATEGORIA RECEITA



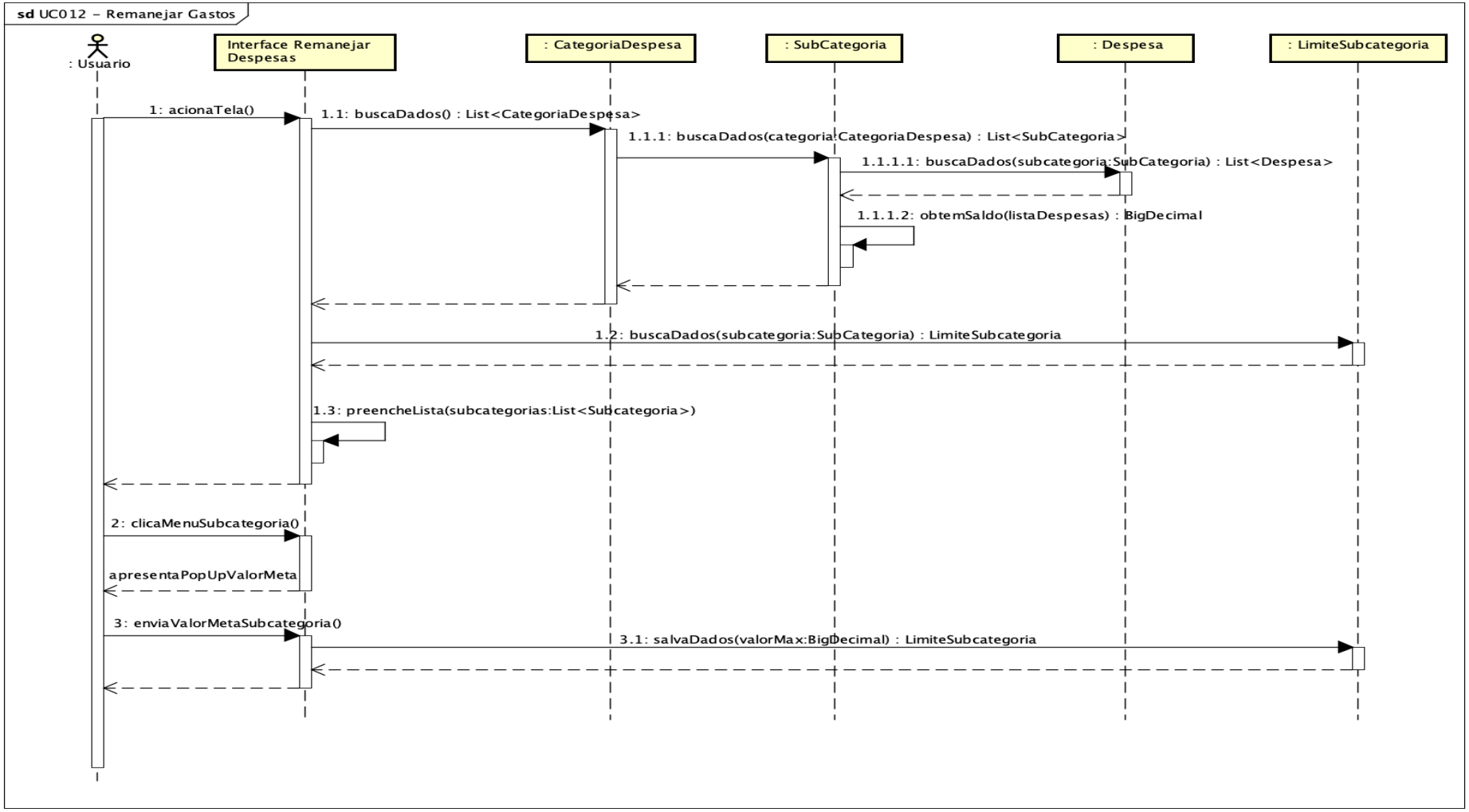
FONTE: Os Autores (2020).

FIGURA 81 – DIAGRAMA DE SEQUENCIA – MANTER SUBCATEGORIA DESPESA



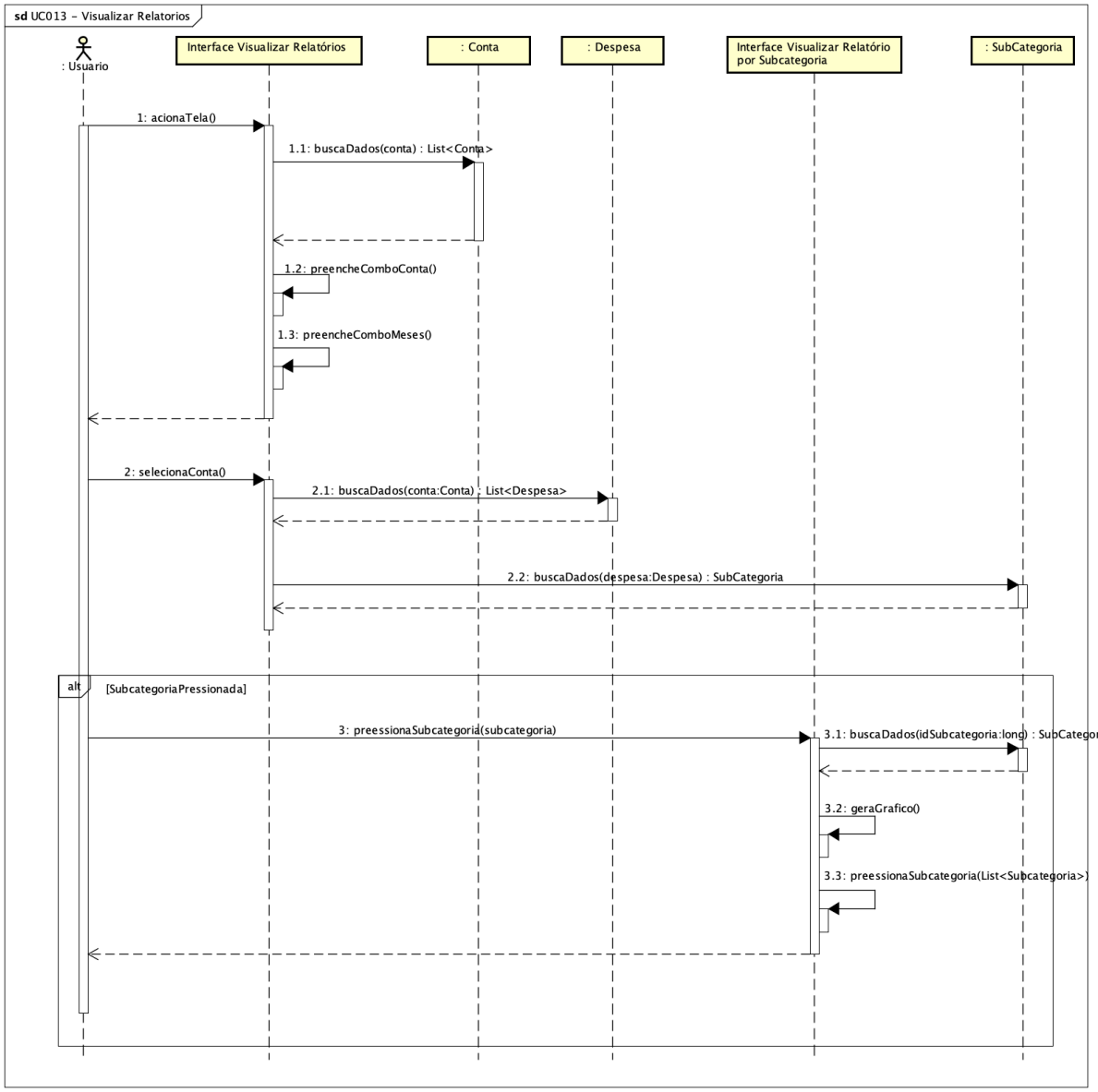
FONTE: Os Autores (2020).

FIGURA 82 – DIAGRAMA DE SEQUENCIA – REMANEJAR GASTOS



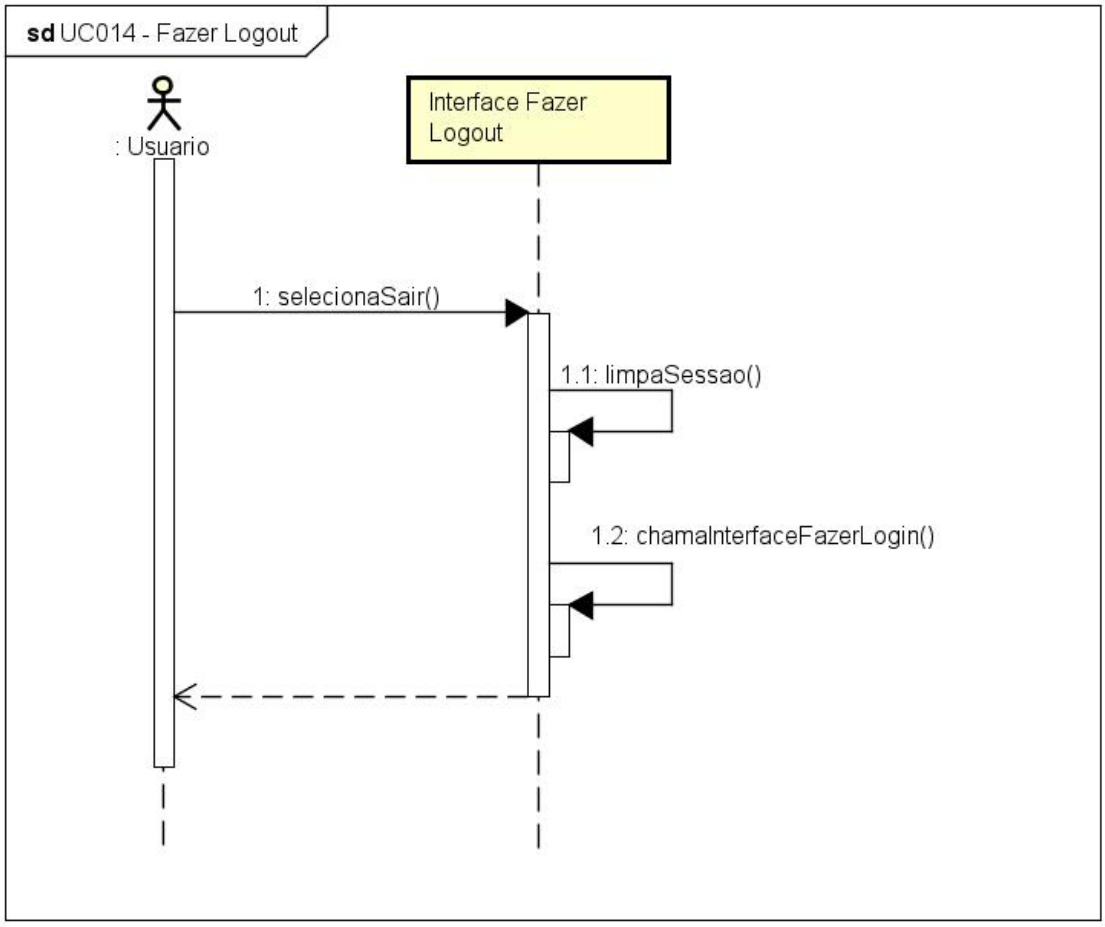
FONTE: OS AUTORES (2020).

FIGURA 83 – DIAGRAMA DE SEQUENCIA – VISUALIZAR RELATÓRIOS



FONTE: Os Autores (2020).

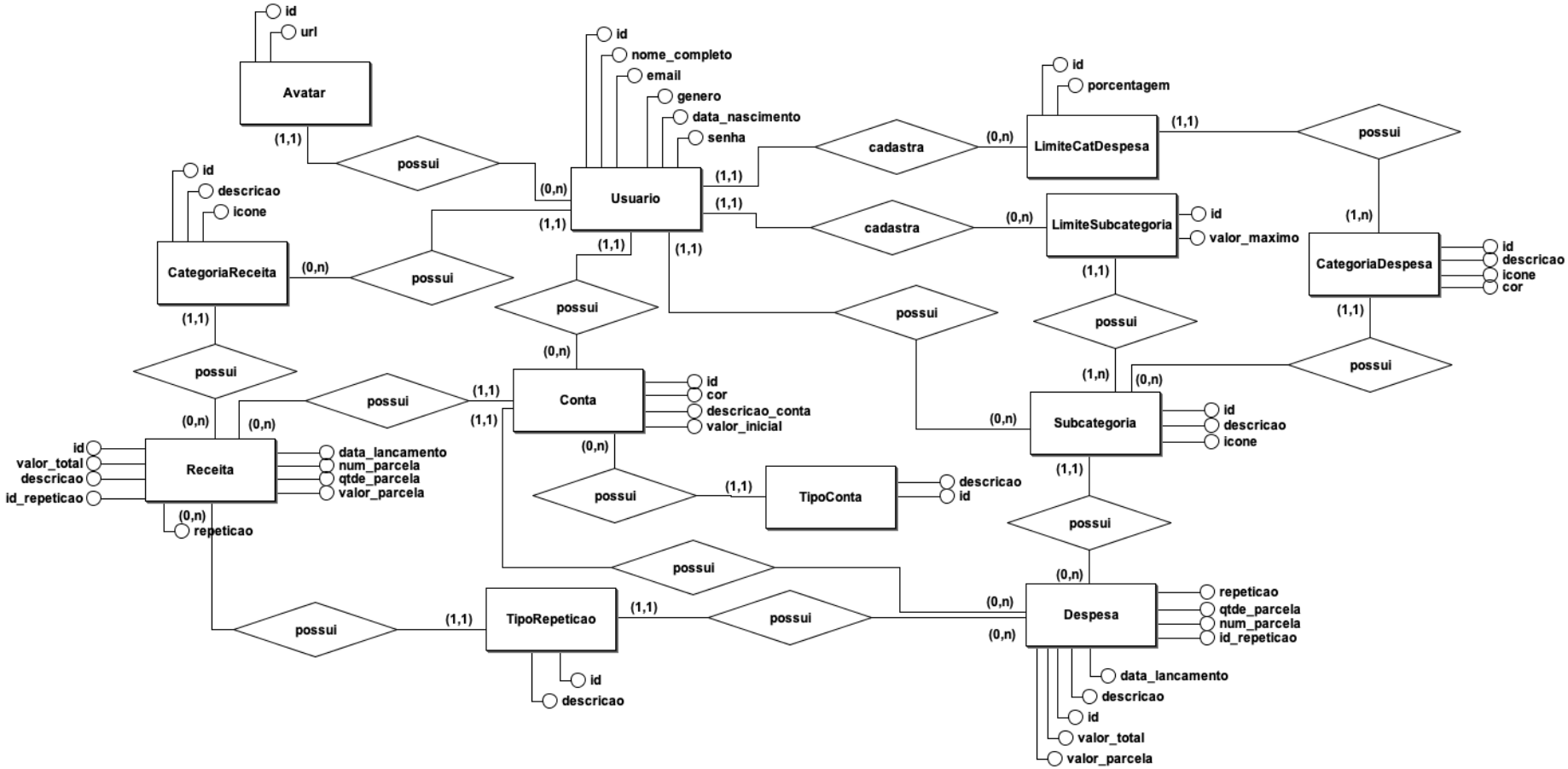
FIGURA 84 – DIAGRAMA DE SEQUENCIA – FAZER LOGOUT



FONTE: Os Autores (2020).

APÊNDICE G – DIAGRAMA CONCEITUAL

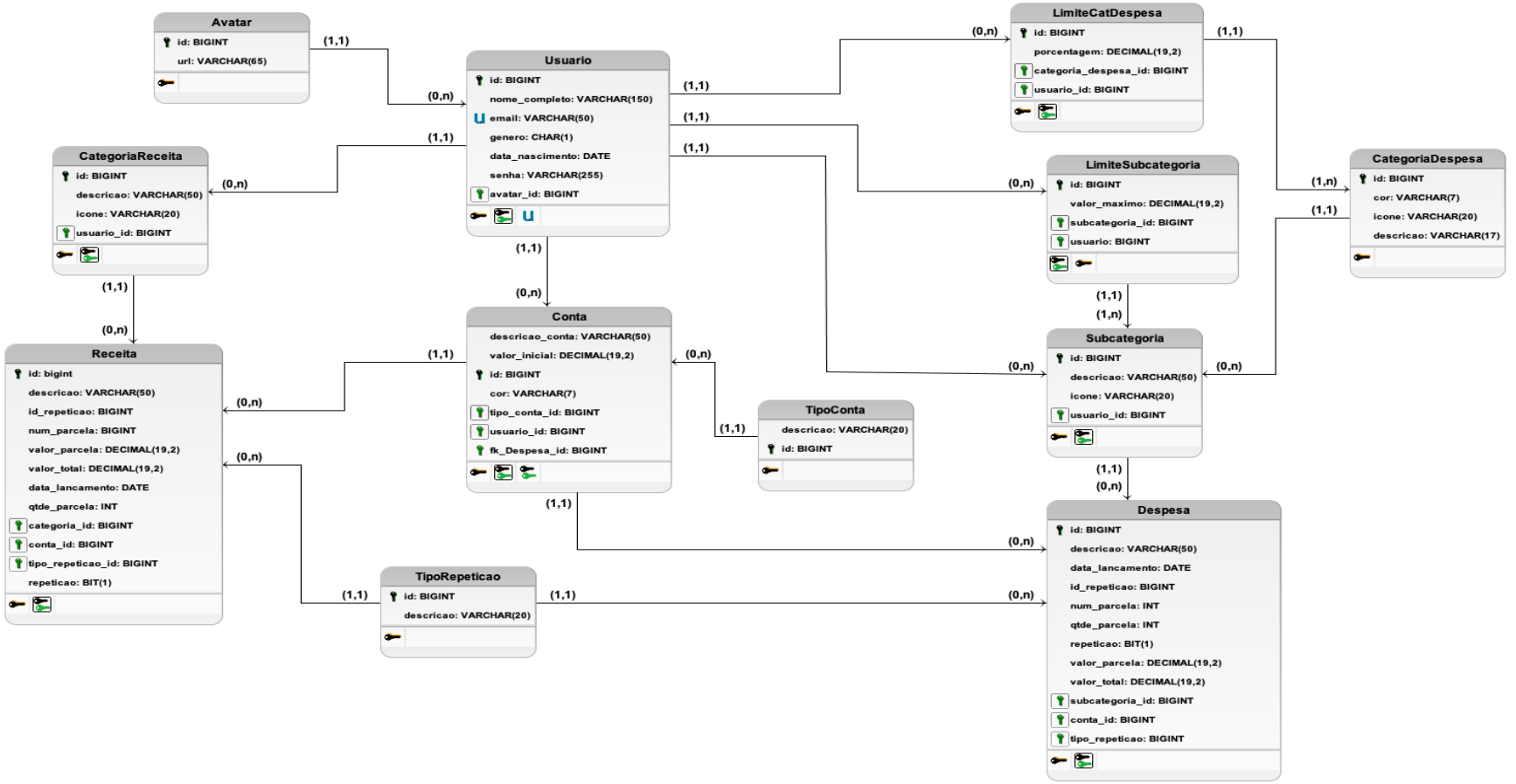
FIGURA 85 – DIAGRAMA CONCEITUAL BANCO DE DADOS



FONTE: Os Autores (2019).

APÊNDICE H – DIAGRAMA LÓGICO

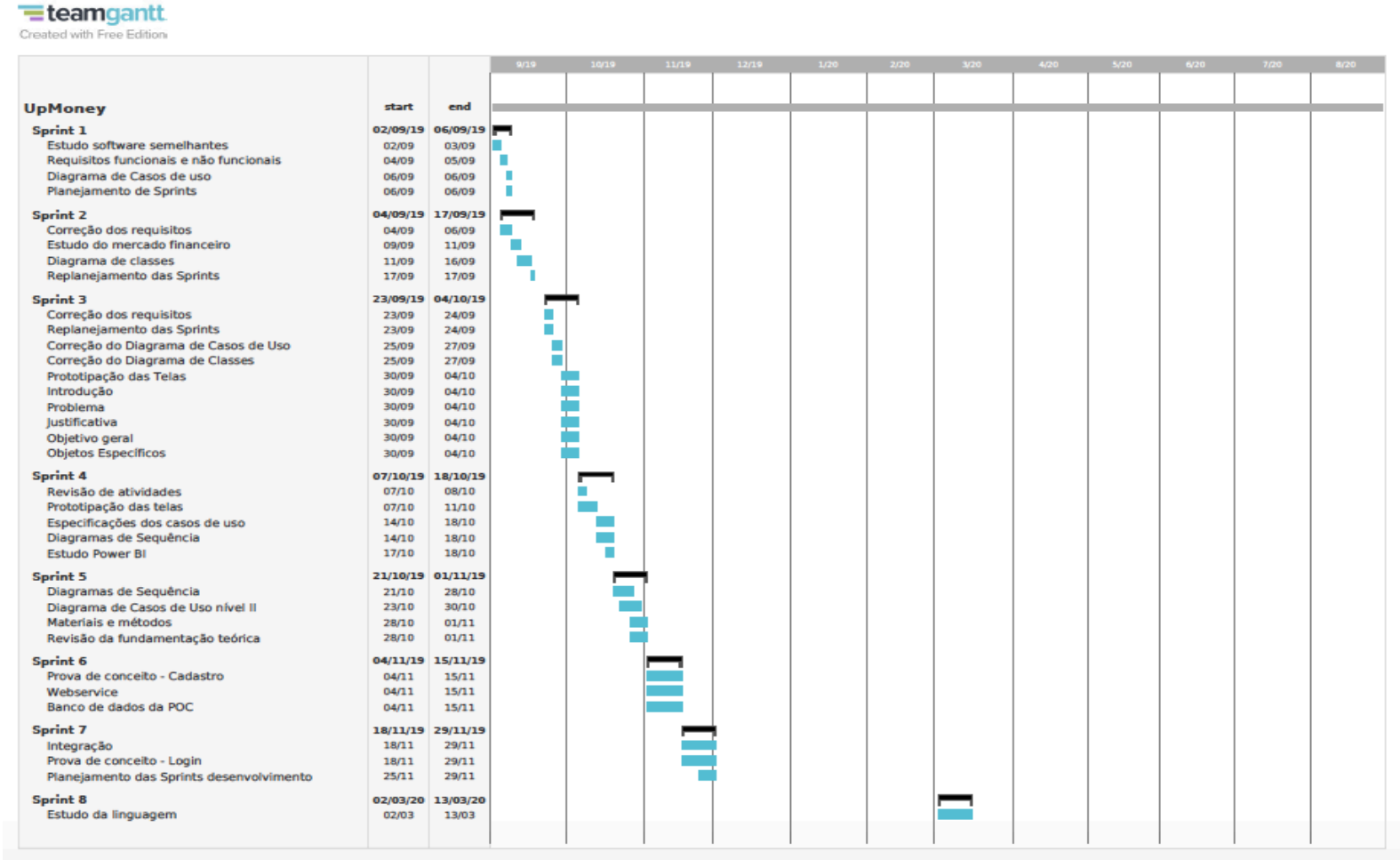
FIGURA 86 – DIAGRAMA LÓGICO BANCO DE DADOS



FONTE: Os Autores (2019).

APÊNDICE I – DIAGRAMA DE GANTT

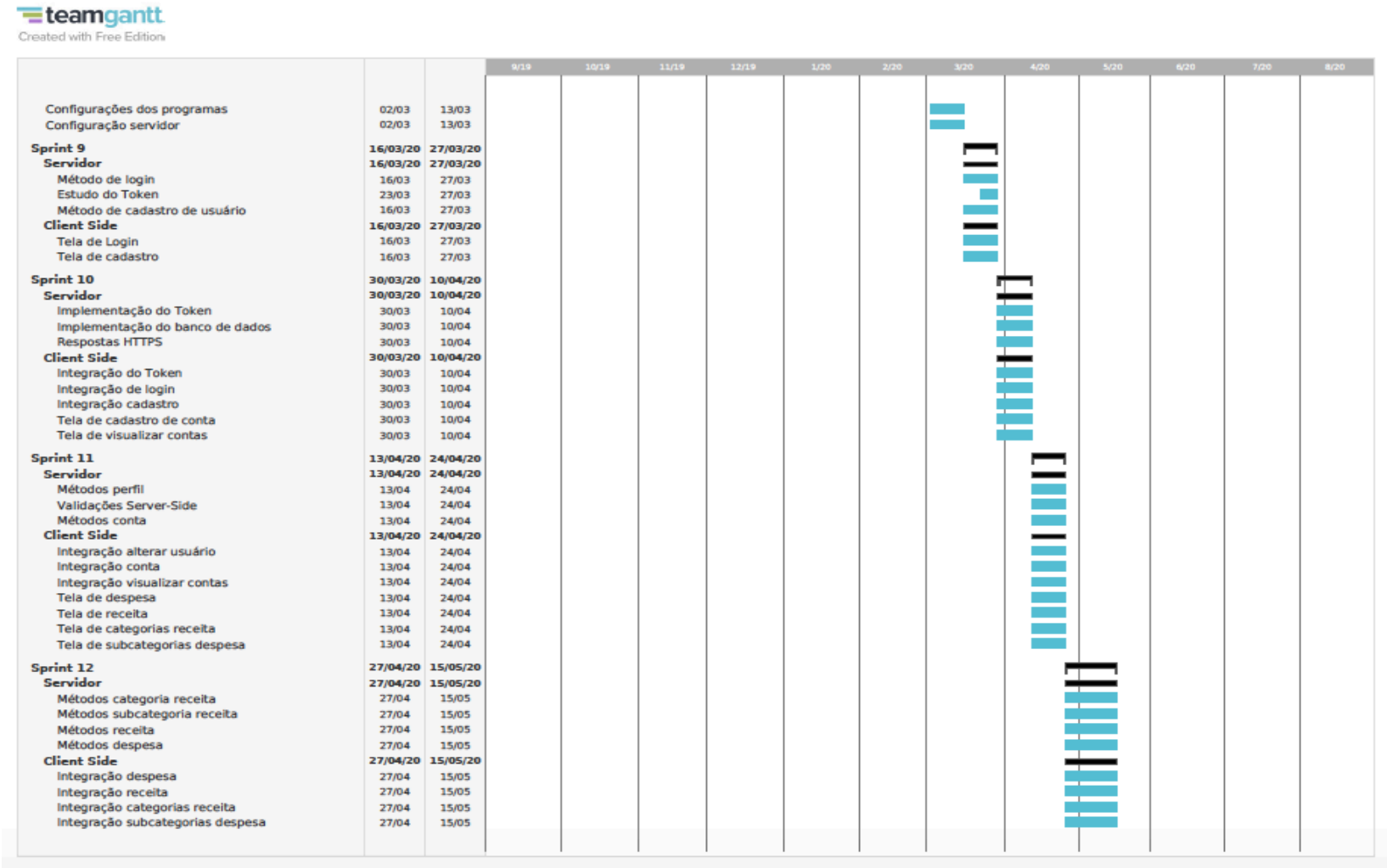
FIGURA 87 – DIAGRAMA GANTT – PRIMEIRA FASE



UpMoney	start	end
Sprint 1	**02/09/19**	**06/09/19**
Estudo software semelhantes	02/09	03/09
Requisitos funcionais e não funcionais	04/09	05/09
Diagrama de Casos de uso	06/09	06/09
Planejamento de Sprints	06/09	06/09
Sprint 2	**04/09/19**	**17/09/19**
Correção dos requisitos	04/09	06/09
Estudo do mercado financeiro	09/09	11/09
Diagrama de classes	11/09	16/09
Replanejamento das Sprints	17/09	17/09
Sprint 3	**23/09/19**	**04/10/19**
Correção dos requisitos	23/09	24/09
Replanejamento das Sprints	23/09	24/09
Correção do Diagrama de Casos de Uso	25/09	27/09
Correção do Diagrama de Classes	25/09	27/09
Prototipação das Telas	30/09	04/10
Introdução	30/09	04/10
Problema	30/09	04/10
Justificativa	30/09	04/10
Objetivo geral	30/09	04/10
Objetos Específicos	30/09	04/10
Sprint 4	**07/10/19**	**18/10/19**
Revisão de atividades	07/10	08/10
Prototipação das telas	07/10	11/10
Especificações dos casos de uso	14/10	18/10
Diagramas de Sequência	14/10	18/10
Estudo Power BI	17/10	18/10
Sprint 5	**21/10/19**	**01/11/19**
Diagramas de Sequência	21/10	28/10
Diagrama de Casos de Uso nível II	23/10	30/10
Materiais e métodos	28/10	01/11
Revisão da fundamentação teórica	28/10	01/11
Sprint 6	**04/11/19**	**15/11/19**
Prova de conceito - Cadastro	04/11	15/11
Webservice	04/11	15/11
Banco de dados da POC	04/11	15/11
Sprint 7	**18/11/19**	**29/11/19**
Integração	18/11	29/11
Prova de conceito - Login	18/11	29/11
Planejamento das Sprints desenvolvimento	25/11	29/11
Sprint 8	**02/03/20**	**13/03/20**
Estudo da linguagem	02/03	13/03

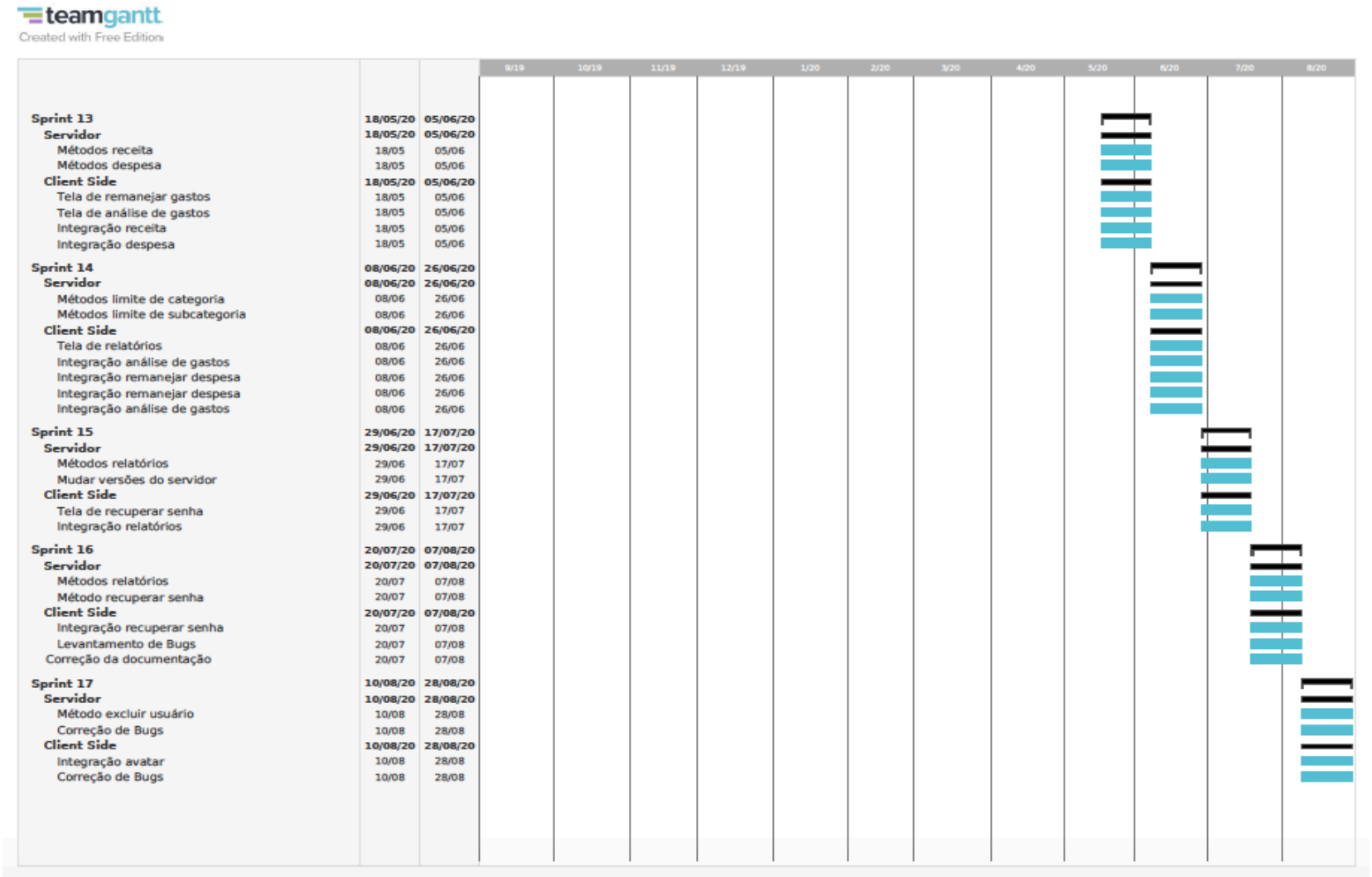
FONTE: Os Autores (2019).

FIGURA 88 – DIAGRAMA GANTT – SEGUNDA FASE (PARTE 1)



FONTE: Os Autores (2019).

FIGURA 89 – DIAGRAMA GANTT – SEGUNDA FASE (PARTE 2)



FONTE: Os Autores (2019).