UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

CONTROLE MULTIUSUÁRIO DE DESPESAS DIÁRIAS PARA A PLATAFORMA ANDROID

BRUNO MUEHLBAUER DE SOUZA

BLUMENAU
2016

BRUNO MUEHLBAUER DE SOUZA

CONTROLE MULTIUSUÁRIO DE DEPESAS DIÁRIAS PARA A PLATAFORMA ANDROID

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Francisco Adell Péricas, Mestre - Orientador

BLUMENAU
2016

CONTROLE MULTIUSUÁRIO DE DESPESAS DIÁRIAS

PARA A PLATAFORMA ANDROID

Por

BRUNO MUEHLBAUER DE SOUZA

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Mestre – Orientador, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. Paulo Fernando da Silva, Mestre – FURB

Blumenau, 04 de julho de 2016

Dedico este trabalho aos meus familiares e amigos que estiveram ao meu lado durante os anos de graduação e de certa forma colaboraram para o desenvolvimento deste trabalho.

AGRADECIMENTOS

Aos meus familiares pela força e apoio em todos os momentos.

Aos meus amigos, por lembrarem sempre da importância e dificuldade da graduação bem como seus benefícios.

Ao meu mestre-orientador, Francisco Adell Péricas pelo apoio, ajuda, e principalmente por confiar no meu trabalho e dedicação.

Ao professor Mauricio Copabianco Lopes, responsável pela disciplina de Trabalho de Conclusão de Curso II.

Tente uma, duas, três vezes e se possível tente a quarta, a quinta e quantas vezes for necessário. Só não desista nas primeiras tentativas, a persistência é amiga da conquista. Se você quer chegar onde a maioria não chega, faça o que a maioria não faz.

Bill Gates

RESUMO

Este trabalho apresenta um aplicativo de controle de despesas diárias multiusuário que foi desenvolvido para controle de gastos financeiros entre um ou mais usuários. Foi desenvolvido para a plataforma Android tendo como principal objetivo tornar eficiente o controle financeiro dos usuários que possuem contas em banco que são utilizadas em conjunto com outras pessoas. As despesas ficam armazenadas num ambiente único e cada usuário se torna responsável pela correta alimentação do aplicativo. O desenvolvimento ocorreu utilizando ambiente Windows com Android Studio para o aplicativo Android e Eclipse para o webservice. O banco de dados utilizado foi o PostgreeSQL para o webservice e SQLite nos dispositivos Android.

Palavras-chave: Multiusuário. Controle de despesas. Android.

ABSTRACT

This work describes a multi-user daily expense tracking application that was developed to control financial expenses between one or more users. It was developed for the Android platform with the main objective to make effective financial control of users who have bank accounts that are used in conjunction with others. The costs are stored in a single environment and each user becomes responsible for the correct application of power. The development took place using Windows with Android Studio for Android and Eclipse application to the webservice. The database used was PostgreSQL and SQLite for webservice on Android devices.Keywords: Multi-User. Control of expenses. Android.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Os jovens, na grande maioria, já possuem cartão de crédito e no entanto, nunca tiveram formação sobre investimentos, economia, impostos, etc... (PERETTI, 2008). Porém, uma boa administração financeira é capaz de proporcionar às pessoas melhor bem-estar e melhor qualidade de vida. Segundo Peretti (2008), a educação financeira desenvolve o caráter, a personalidade e afasta o medo, fazendo com que se assuma e crie coragem para resolver os problemas.

Segundo Braga (2013), as despesas diárias ou aquelas esporádicas que passam despercebidas são o grande vilão do orçamento financeiro das pessoas. Essas pequenas despesas são denominadas gastos fantasmas.

> Quando falamos em fantasmas alguns acreditam, outros não. O fato é que os gastos são reais e, assim como os fantasmas que nos causaram espanto, nos assustarão muito no momento em que descobrirmos que existem e, principalmente, ao nos darmos conta do tamanho do estrago que causam no orçamento. (BRAGA, 2013).

Gerir os gastos e despesas pessoais são tarefas difíceis pois requerem prioridades bem definidas, comprometimento e disciplina (NAVARO, 2014). De acordo com Braga (2013), não ter controle das despesas diárias pode atingir drasticamente o orçamento pessoal.

Padilha (2013), diz que um dos principais passos para elaborar um plano de economia é anotar numa planilha todas essas despesas extras não-fixas, desde as mais altas, até as mais baixas.

Papel e caneta, planilhas ou outras formas de elaborar um plano de economia são alimentados de forma assíncrona, horas ou até dias após a despesa ter acontecido. E, de acordo com Navarro (2014), é preciso ir além dessas planilhas.

> É muito comum que nos esqueçamos de listar todos os custos que uma aquisição ou um consumo possa nos trazer. Quando não estamos atentos para esta apuração "cirúrgica", geralmente nossos levantamentos de gastos limitam-se a despesa principal na análise. Por exemplo: quando vamos a um restaurante, contabilizamos apenas o custo da refeição e deixamos de ponderar o custo do serviço, do estacionamento, do combustível/táxi, etc. (BASTOS, 2011).

Com base no que foi apresentado, este trabalho visa disponibilizar uma solução de controle financeiro pessoal fazendo com que usuários que compartilham um mesmo recurso financeiro possam utilizar uma solução automatizada para seus registros diários.

1.1 OBJETIVOS

O objetivo deste trabalho é o desenvolvimento de um sistema de controle multiusuário de despesas diárias para a plataforma Android.

Os objetivos específicos são:

a) sincronizar informações entre um dispositivo móvel e base de dados remota por meio de um *webservice*;
b) lançar despesas e receitas através de uma interface Android;
c) permitir mais de um usuário controlar uma mesma conta compartilhada de despesas.

1.2 ESTRUTURA

Este trabalho está estruturado em quatro capítulos. O primeiro capítulo trata da introdução sobre o tema do trabalho, seus objetivos e estrutura.

O segundo capítulo apresenta a fundamentação teórica, aprofundando o tema escolhido afim de proporcionar um melhor entendimento do trabalho como um todo.

O terceiro capítulo contempla o desenvolvimento da aplicação propriamente dita. Inclui diagramas de caso de uso, diagramas de classe, persistência e as etapas de desenvolvimento bem como ferramentas e resultados obtidos.

O quarto capítulo apresenta as conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda assuntos a serem apresentados nas seções a seguir, tais como independência financeira, planejamento financeiro, controle de gastos, Android, Android Studio, dispositivo virtual Android (AVD), Jboss Forge, webservice, REST e trabalhos correlatos.

2.1 INDEPENDÊNCIA FINANCEIRA

Para atingir a dependência financeira é necessário poupar, fazer sobrar dinheiro, gastar menos do que se ganha. Ao mesmo tempo, deve-se saber administrar as aplicações com eficiência. A Tabela 1 apresenta os montantes calculados com base em uma poupança mensal de R$ 50,00, baseado na idade atual do investidor e diferentes taxas médias de retorno (juro) anual. Ou seja, se o investidor, hoje, com 20 anos aplicar R$ 50,00 com retorno médio anual de 15%, será um milionário com 60 anos (HOJI, 2007).

Tabela 1 - Montante acumulado aos 60 anos de idade, com poupança mensal de R$ 50,00.

	Idade atual							
	20	25	30	35	40	45	50	55
Taxa de retorno	(40)	(30)	(30)	(25)	(20)	(15)	(10)	(5)
10% a.a	277.517	169.940	103.142	61.666	35.913	19.922	9.993	3.828
11% a.a	366.364	215.082	125.319	72.044	40.427	21.664	10.530	3.922
12% a.a	485.051	272.952	152.601	84.310	45.561	23.573	11.097	4.017
13% a.a	643.663	347.121	186.170	98.812	51.398	25.664	11.696	4.115
14% a.a	855.661	442.215	227.484	115.959	58.037	27.953	12.329	4.215
15% a.a	1.138.990	564.133	278.328	136.323	65.585	30.461	12.999	4.317

Fonte: Hoji (2007, p. 26).

Com idade em torno de 30 anos e estabilidade em uma profissão o investidor pode aplicar mensalmente um valor de R$ 710,00 em uma aplicação com taxa média de 8% ao ano. Também seria suficiente para atingir R$ 1 milhão aos 60 anos.

Em algumas famílias de classe média brasileira é comum o pai dar ao seu filho um automóvel de presente, assim que ingressar na faculdade. Se ingressar em uma federal, reforça o motivo, pois não terá de pagar a mensalidade escolar durante quatro ou cinco anos. Se o valor do automóvel for de um modelo simples de R$ 32 mil, aplicado em uma renda de 9% ao ano, o indivíduo terá seu primeiro milhão aos 60 anos (HOJI, 2007).

2.1.1 Planejamento financeiro

Segundo Hoji (2007), a situação financeira atual de uma família é a consequência das decisões ao longo da vida e do seu planejamento financeiro (ou falta dele). Quando uma família está com muitas dívidas é comum acharem que tiveram "azar" e acabaram endividadas ao invés de acumular riqueza. A culpa do endividamento não é do "azar" e sim falta de técnicas de gestão financeira.

O planejamento financeiro familiar não exige cálculos complexos, mas sim muita disciplina, sacrifícios e renúncias temporárias, pois as escolhas feitas no presente determinam o futuro. Se o salário não é suficiente para cobrir os gastos, então eles devem ser reduzidos. Cada centavo economizado é capaz de virar grandes somas de dinheiro ao serem acumulados (HOJI, 2007).

> A receita é a importância recebida, geralmente de forma periódica, como o salário, aluguel, lucro de operações comerciais, etc... As despesas são gastos que uma vez consumidos, "perdem-se" para sempre. Despesas não podem ser vendidas ou convertidas em dinheiro, porém, são necessárias para a sobrevivência familiar como pagamento de aluguel, pagamento de tratamentos médicos ou odontológicos, impostos, etc. (HOJI, 2007)

Se a receita é maior que a despesa, tem-se uma sobra de dinheiro, que pode ser aplicada em investimentos. Caso contrário, apresenta-se uma situação de falta de dinheiro, obrigando a família a recorrer a um empréstimo ou resgatar parte dos investimentos (caso possua) (HOJI, 2007).

Segundo Torralvo et al. (2012), o planejamento financeiro é uma forma racional de organização das finanças. Pode ser associado a investimentos, bolsas de valores e negociações envolvendo ações e outros ativos com significativo grau de risco.

> O planejamento financeiro é desenvolvido justamente para que a família possa então atingir seus objetivos. Para que isso ocorra, é necessário, primeiramente, que o planejador financeiro consiga avaliar em detalhes o momento atual das finanças da família para que, então, possa traçar suas recomendações (TORRALVO et al., 2012).

2.1.2 Controle de gastos

O primeiro passo para iniciar o processo de enriquecimento é fazer sobrar dinheiro, ou seja, ter uma diferença positiva entre receita e despesa. Segundo Leitão (2015), para que isso aconteça de uma maneira suave o controle de gastos é indispensável. Uma enorme quantidade de gastos poderia ser retirada do orçamento sem qualquer tipo de prejuízo. Isso acontece, pois, a falta de um controle eficiente impossibilita uma visão geral do dinheiro o que gera um impulso descontrolado agravado pela influência dos vendedores ou ocasiões.

Os conceitos iniciais para realizar o controle de gastos com eficiência segundo Navarro (2015) são:

a) entradas: salários, proventos de renda extra, 13º salário e outros;
b) despesas fixas: despesas que sempre virão no final do mês sem sofrer oscilações. São elas aluguel, financiamento de imóvel, faculdade, financiamento de veículo, entre outros.
c) despesas variáveis: que oscilam de acordo com o mês ou situação. Como conta de luz, água, telefone, alimentação, gastos com o carro, troca de óleo, etc.

Segundo Ávila (2015), uma planilha de controle de gastos é essencial para a tomada de decisão do dia. Ela responde algumas perguntas que podem acontecer no dia-a-dia de uma família, como:

a) é possível comprar este eletrodoméstico?
b) quanto podemos investir este mês?
c) será necessário pedir um adiantamento ou empréstimo?

Segundo Navarro (2015), os motivos para o controle são:

a) impulsividade/consumismo;
b) falta de objetivos com o dinheiro;
c) gastos excessivos com festas/eventos;
d) não tomar devidos cuidados com cheques e cartões de crédito.

2.2 ANDROID

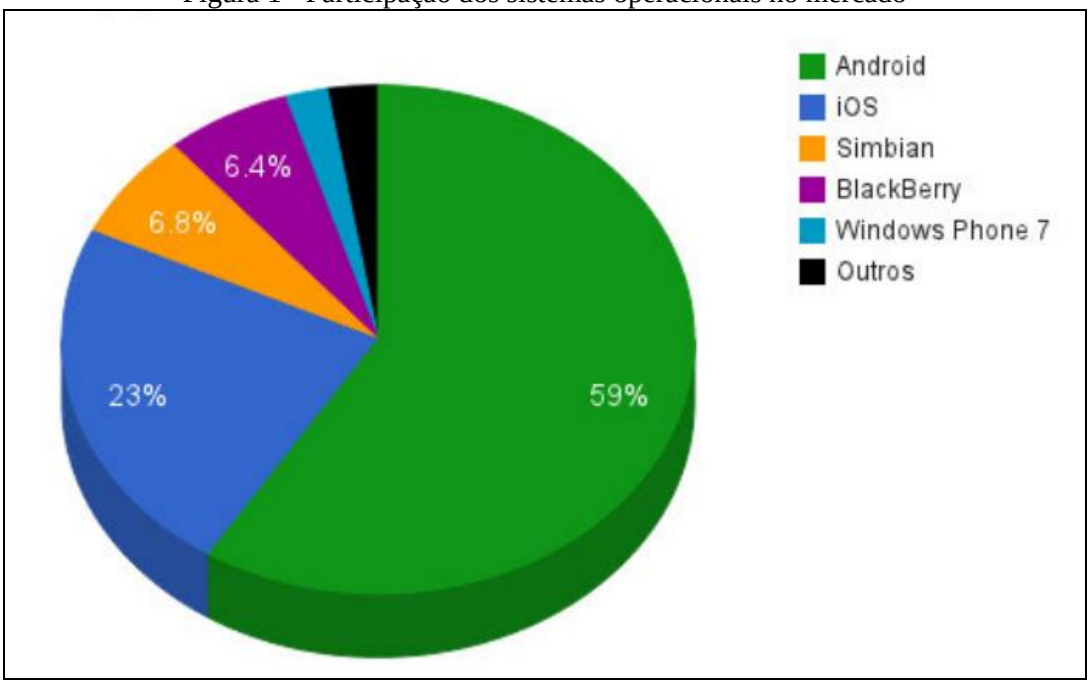
Segundo Cidral (2012), Android é o sistema operacional do Google para smarthphones baseado no Linux. Além disso, a loja virtual Google Play tem aplicativos e jogos tanto gratuitos quanto pagos para os *smarthphones* e *tablets* com Android.

> A plataforma Android foi concebida inicialmente pelo Google. A plataforma está sendo e será mantida pelo Open Handset Alliance, que é um grupo formado por mais

> de 30 empresas (de tecnologias de dispositivos móveis, provedoras de serviços móveis, fabricantes, etc) as quais se uniram para inovar e acelerar o desenvolvimento de aplicações, serviços, trazendo aos consumidores uma experiência mais rica em termos de recursos, menos dispendiosa em termos financeiros para o mercado móvel. Pode-se dizer que a plataforma Android é a primeira plataforma móvel completa, aberta e livre (RABELLO, 2013).

Segundo Monteiro (2013) o Android possui 59% do mercado de *smarthphones* e soma uma quantia de 89,9 milhões de aparelhos. A Figura 1 demonstra a participação dos principais sistemas operacionais presentes no mercado.

Figura 1 - Participação dos sistemas operacionais no mercado



Fonte: Monteiro (2013, p. 2).

2.3 ANDROID STUDIO

Segundo Carvalho (2013), o Android Studio é uma IDE para desenvolvimento na plataforma Android baseado no InteliJ Community Version. Tem o mesmo objetivo do Eclipse + Android Developer Tools (*ADT*) e provê um ambiente de desenvolvimento, depuração, testes e perfil multiplataforma para Android.

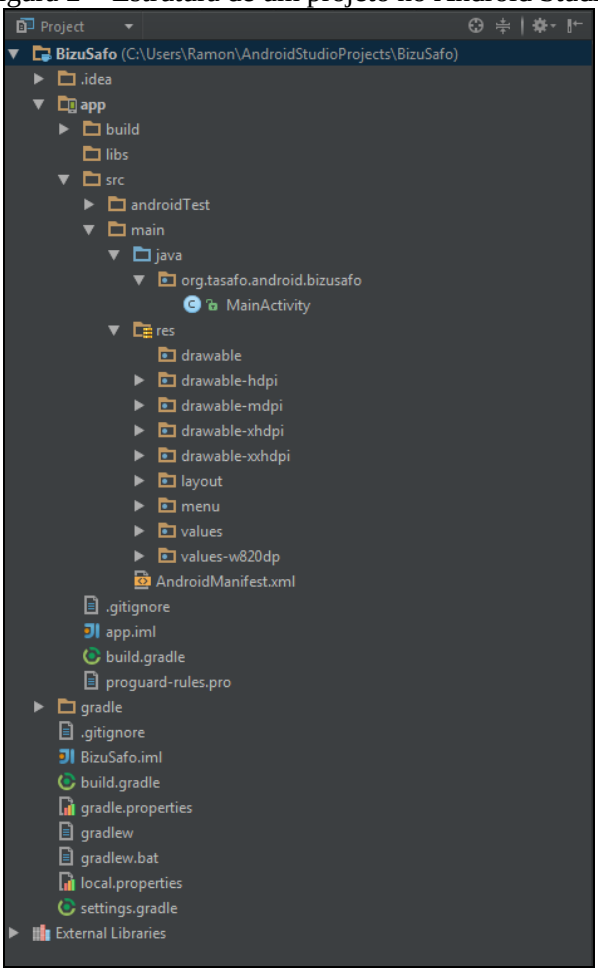
O Android Studio já está muito evoluído com relação ao Eclipse + ADT pois já conta com extensões para vários dispositivos, como relógios inteligentes (AndroidWear), TV's (AndroidTV) e carros (AndroidAuto) o que não acontecia com o seu antecessor pois o mesmo não obtinha suporte a geração de múltiplos *Android Packages* (*APK*) , arquivo compilado do Android (RABELLO, 2015).

Sua interface é bem atraente e também é possível customizar atalhos de forma que sejam iguais às de outras IDE's, como o Eclipse, o que torna menor a curva de aprendizado. A funcionalidade *Injection Language* permite que a IDE consiga validar até mesmo as *strings* do projeto. Fornece também integração com *Mercurial*, *Git* e *Subversion*, além de interface gráfica para as principais operações como *commits*, *pushs* e *diffs* (CARVALHO, 2013).

2.3.1 A estrutura de um projeto no Android Studio

O Android Studio mudou muito em relação ao Android ADT. Com a incorporação do *Gradle* como sistema de *build* introduz o conceito de módulo para cada subprojeto criado, ao contrário do ADT, onde eram criados vários projetos com suas respectivas bibliotecas (RABELLO, 2015). No caso da Figura 2 o nome do projeto é BizuSafo e o nome do módulo é app.

Figura 2 - Estrutura de um projeto no Android Studio



Fonte: Rabello (2015)

De acordo com Rabello (2015), em um projeto Android tem-se as pastas:

a) `src/androidTest/java`: pacote principal contendo as pastas de teste;

b) `scr/main/java`: pacote principal contendo as classes Java do módulo;

c) `src/main/res`: pasta de recursos (layouts, imagens, animações, etc.);

d) `scr/main/AndroidManifest.xml`: XML descritor do app que contém as permissões e declaração de componentes;

e) `app/build.gradle`: *script Gradle* contendo as configurações de módulo, versões de SDK e configuração de build;

f) `nomedoprojeto/gradlew` e `gradlew.bat`: são *scripts* para rotinas do *Gradle*. Por exemplo, se executar o comando *gradlew assembleDebug* irá gerar um aquivo *.apk* do projeto para o *build* tipo depuração;

g) `nomedoprojeto/settings.gradle`: arquivo de configuração das dependências do projeto.

2.3.2 Dispositivo Virtual Android (AVD)

Segundo Rabello (2015), o ideal é desenvolver utilizando o próprio dispositivo físico. Porém este tipo de abordagem representa ter um bom investimento. Como em algumas situações não é possível ter o dispositivo físico, é possível criar dispositivos virtuais conhecidos como *Android Virtual Devices* (AVD's). Com um AVD é possível emular um *smarthphone* com arquitetura Intel x86, Android 5.0 Lollipop, suporte a SD Card, 3G, etc.

De acordo com Rabello (2015), existem duas formas de executar um aplicativo no emulador. Pode-se iniciar um AVD já criado e clicar no ícone de início ao lado da descrição do módulo do projeto ou então executar o módulo diretamente que abrirá uma janela para selecionar um AVD já existente ou iniciar a criação de um novo. A Figura 3 exibe o projeto BizuSafo em um dispositivo virtual.

Figura 3 - Exemplo de dispositivo virtual Android simulando uma aplicação.

Fonte: Rabello (2015).

2.4 WEBSERVICE

Segundo Santos (2015), *webservice* define uma arquitetura de comunicação entre softwares independentemente da plataforma que foram desenvolvidos. Sua característica é a comunicação com rede sempre disponível. Este tipo de abordagem tem sido muito utilizado na integração de ferramentas.

> O conceito de web service solucionou dois grandes problemas no mundo do desenvolvimento de software. Primeiro, permitiu que aplicações — independentemente de plataforma — trocassem informações. Segundo, mudou o conceito que temos sobre reutilização (SANTOS, 2015).

Na prática, *webservice* é uma arquitetura de comunicação entre softwares que sejam da mesma plataforma ou não. Os principais benefícios segundo Santos (2015), são:

a) protocolos baseados em um padrão;

b) protocolos baseados em texto;

c) utilização da porta 80, sem bloqueios pelo *firewall*.

2.4.1 REST

Segundo Saudate (2014), REST significa *REpresentational State Transfer*, e teve origem na tese de doutorado de Roy Fielding. Todo serviço REST é baseado nos chamados recursos, que são entidades bem definidas em sistemas, que possuem identificadores e

endereços (URLs) próprios. O protocolo REST é guiado pelas boas práticas de uso do HTTP. São elas:

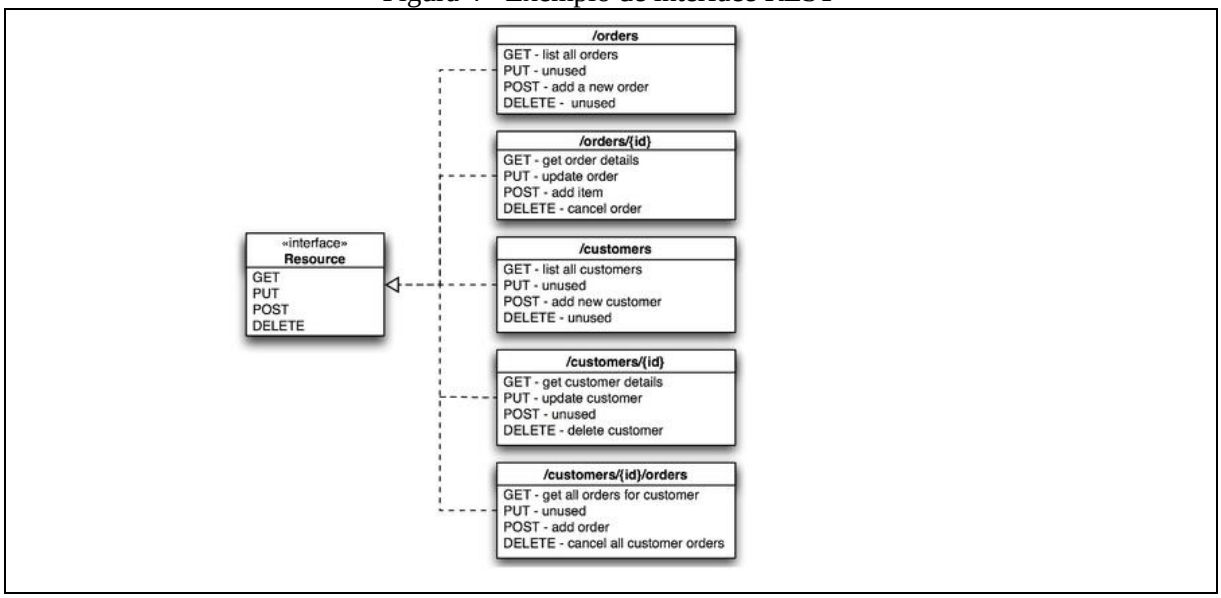
a) uso adequado dos métodos HTTP;
b) uso adequado de URLs;
c) uso de códigos de *status* padronizado;
d) uso adequado de cabeçalhos HTTP;
e) interligação de recursos.

Ainda de acordo com Saudate (2014), o protocolo HTTP é o único protocolo totalmente compatível conhecido. A possível causa para esta preferência seria por que o autor do protocolo HTTP também é autor do REST.

2.4.2 Métodos HTTP

Segundo Saudate (2014) cada método possui particularidades e aplicações de acordo com as necessidades. A Figura 4 ilustra um exemplo de interface genérica utilizando REST.

Figura 4 - Exemplo de interface REST



Fonte: Saudade (2014)

Os métodos GET, PUT, POST e DELETE são métodos padrão utilizados em todas as classes. Ao utilizar o método GET de */customers* por exemplo, a resposta será todos os *customers* da aplicação. Ainda utilizando o método GET, mas colocando um parâmetro *id* no final da *URL* é possível retornar apenas o *customer* responsável pelo parâmetro.

> Para interagir com as URLs, os métodos HTTP são utilizados. A regra de ouro para esta interação é que URLs são substantivos, e métodos HTTP são verbos. Isto quer dizer que os métodos HTTP são os responsáveis por provocar alterações nos recursos identificados pelas URLs. (SAUDATE, 2014).

É importante notar que segundo Saudate (2014), os métodos GET, PUT, POST e DELETE podem ser diretamente relacionados com operações com banco de dados. Ou seja, para recuperar uma informação é utilizado o método GET. Para criar um novo objeto é utilizado o método POST, e para deletar o método DELETE. Ainda de acordo com Saudate (2014) todas essas operações são lógicas, o que não significa que o método DELETE deve realmente deletar o objeto, apenas poderia indisponibilizar o recurso.

De acordo com Saudate (2014) toda requisição enviada retorna um código de status. Estes códigos são divididos em cinco famílias: 1xx, 2xx, 3xx, 4xx e 5xx sendo:

a) 1xx – Informacionais;
b) 2xx – Códigos de sucesso;
c) 3xx – Códigos de redirecionamento;
d) 4xx – Erros causados pelo cliente
e) 5xx – Erros originados no servidor.

Os códigos mais utilizados são:

a) 200 – *OK*;
b) 201 – *Created*;
c) 202 – *Accepted*;
d) 400 – *Bad Request*;
e) 404 – *Not Found*.

2.5 TRABALHOS CORRELATOS

A seguir serão descritos 2 trabalhos correlatos ao proposto, sendo como principal semelhança o tema de controle de despesas. O item 2.6.1 descreve o aplicativo Minhas Economias que é *mobile-web* e controla gastos, possuindo extensão para dispositivos móveis. O item 2.6.2 descreve o MOBILLS, um aplicativo também *mobile-web* de controle de gastos para Android, iPhone e WindowsPhone.

2.5.1 Minhas Economias

Segundo o site TodoCelular (2015), o software Minhas Economias é um serviço online para administração de finanças pessoais. Possui versões para web, iOS e Android. Com ele é possível verificar saldo de contas, cadastrar despesas e receitas, fazer transferências, parcelamentos, anotações e transações *online*. A Figura 5 ilustra a tela do aplicativo para *smarthphone*.

Figura 5 - Telas de visualização de contas no Minhas Economias.

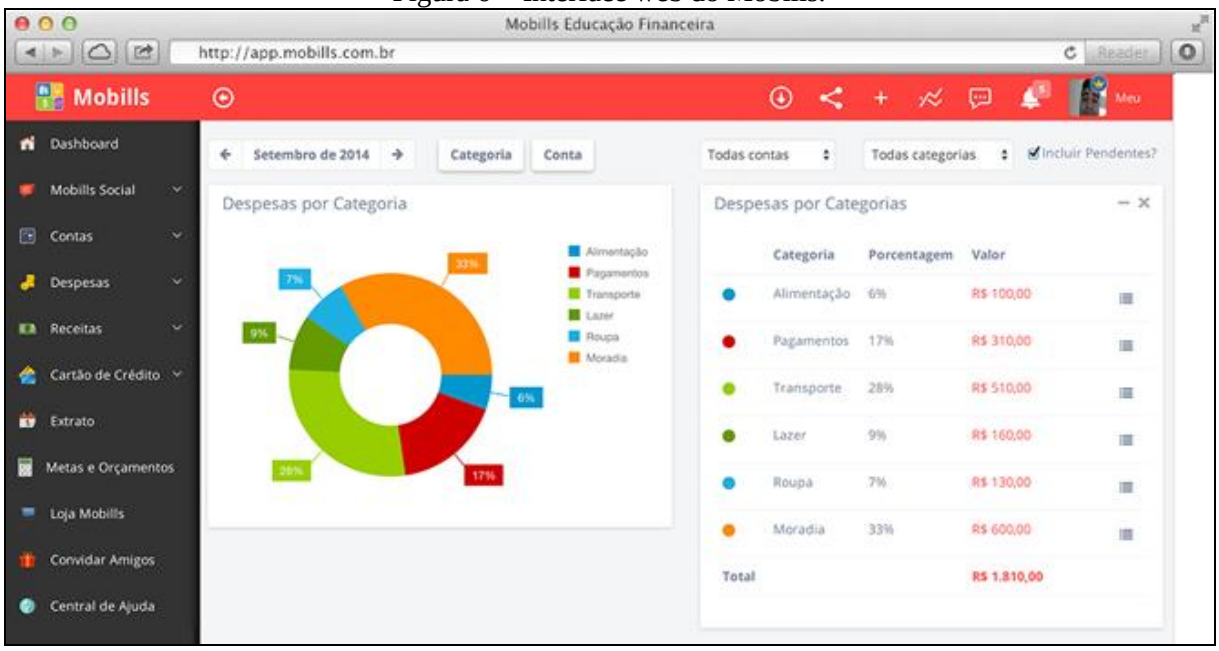
Fonte: TodoCelular (2015).

2.5.2 Mobills

Segundo Mobills (2014), o Mobills é um sistema de controle financeiro pessoal e *online*. Foi criado em 2013 por David Mosiah e Calos Terceiro e incialmente foi desenvolvido apenas para *smarthphones* Android, porém atualmente conta com interface *web* e pode ser usado no iPhone e Windows Phone.

De acordo com Vaz (2014) sua principal função é administrar as finanças pessoais. Com ele é possível assumir o controle de gastos com supermercado, lazer, cabelereiro, transporte, luz, água, etc. Possui categorização e cálculos em tempo real. A Figura 6 ilustra a interface web do Mobills.

Figura 6 - Interface web do Mobills.



Fonte: Mobills (2014).

Segundo Almeida (2014), os dados podem ser sincronizados com a nuvem e os relatórios podem ser acessados pelo computador. Também conta uma rede social onde os usuários podem interagir com a comunidade através de dicas financeiras, pedir ajuda ou conselhos sobre finanças.

> A interface na versão web é bastante parecida com a versão do aplicativo Android, prezando pela beleza e intuitividade na utilização. A sincronização pode ser configurada de forma automática, onde os lançamentos são sincronizados assim que salvos, ou de forma manual, sincronizando somente quando desejada. Todos as operações realizadas pelo aplicativo, também podem ser feitas pela web (ALMEIDA, 2014).

Algumas funcionalidades a mais, segundo Almeida (2014) são:

a) contas e transferências: cadastro de contas simples;
b) análise de despesa inteligente;
c) metas e orçamentos: é possível cadastrar orçamentos mensais para determinadas despesas;
d) calendário;
e) estatísticas;
f) exportação para excel;
g) importação de SMS.

3 DESENVOLVIMENTO

Neste capítulo estão descritas as especificações técnicas do software proposto, levantamento de informações, requisitos, especificação técnica e ferramentas de desenvolvimento, além dos diagramas utilizados no desenvolvimento: diagrama de casos de uso, diagrama de atividades e modelo de entidade e relacionamento.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Este trabalho disponibiliza uma solução para ajudar a usuário a manter o histórico de lançamentos financeiros, no seu *smarthphone* do usuário. A funcionalidade multiusuário agrega a vantagem para usuários que precisam saber exatamente toda a movimentação de uma conta compartilhada com sua família, com colegas de trabalho ou até mesmo com amigos.

O sistema foi desenvolvido usando um *webservice* para sincronização dos dados possibilitando a função de compartilhamento de conta, e uma interface *mobile*, fornecendo acessibilidade e agilidade para o lançamento de gastos. Além de manter os gastos, é possível acessar gráficos agrupados por usuários ou categorias para saber quem ou o que está consumindo mais do orçamento.

3.2 ESPECIFICAÇÃO

Neste tópico serão apresentados os Requisitos Funcionais (RF), os Requisitos Não Funcionais (RNF), os diagramas de casos de uso, o Modelo de Entidade Relacionamento (MER) e o diagrama de classes.

3.2.1 Requisitos do Sistema

O Quadro 1 apresenta os requisitos funcionais do aplicativo *mobile* e do *webservice*.

Quadro 1 - Requisitos funcionais

Requisitos Funcionais
RF001 - O sistema móvel deverá ser acessado por meio de login e senha.
RF002 - O sistema móvel deve permitir cadastrar novos usuários. Deverá exigir o login, a senha, o nome, o sobrenome e o e-mail do usuário no momento do cadastro.
RF003 - O sistema móvel permitirá o cadastro de novas carteiras. Deverá exigir um nome para a carteira no momento cadastro.
RF004 - O sistema móvel permitirá o cadastro de novas contas-corrente e deverá exigir um nome e um banco no momento do cadastro.
RF005 - O sistema móvel permitirá o cadastro de cartões de crédito e deverá exigir um nome pra o cartão no momento do cadastro.
RF006 - O sistema móvel deverá possuir categorias do tipo "R - Receita" pré-definidas. São elas: Salário, Bônus, Subsídio e Outro.
RF007 - O sistema móvel deverá possuir categorias do tipo "D - Despesa" pré-definidas. São elas: Alimentação, Eventos, Transporte, Lazer, Vestuário, Saúde e Educação.
RF008 - O sistema móvel permitirá movimentar contas por meio de lançamentos de "R - Receita" ou "D - Despesa".
RF009 - O sistema móvel deverá exigir uma data de movimento, uma conta (carteira, cartão de crédito ou conta-corrente), uma categoria do tipo "D - Despesa" e um valor para o lançamento do tipo "D - Despesa".
RF010 - O sistema móvel deverá exigir uma data de movimento, uma conta (carteira, cartão de crédito ou conta-corrente), uma categoria do tipo "R - Receita" e um valor para o lançamento do tipo "R - Receita".
RF011 - O sistema móvel deverá permitir informar uma data de movimento retroativa para um lançamento do tipo "D - Despesas" ou "R - Receita".
RF012 - O sistema móvel deverá armazenar a data e hora exatos da movimentação de uma conta.
RF013 - O sistema móvel permitirá o compartilhamento de contas (conta-corrente ou cartão de crédito) com outros usuários.
RF014 - O sistema móvel permitirá visualizar extratos de conta. Os extratos poderão ser diários, semanais ou mensais.
RF015 - O sistema móvel permitirá enviar os extratos diários, semanais ou mensais por e-mail.
RF016 - O sistema móvel deverá sincronizar seus dados com o sistema web.
RF017- O sistema móvel deverá permitir cadastrar informações sem conexão e sincronizar quando houver.
RF018 - O sistema móvel permitirá o cadastro de cartões de débito. Exigirá um nome e uma conta-corrente para movimentação no momento de cadastro de um cartão de débito.

O Quadro 2 apresenta os requisitos não funcionais para o aplicativo *mobile* e *webservice*.

Quadro 2 - Requisitos não funcionais

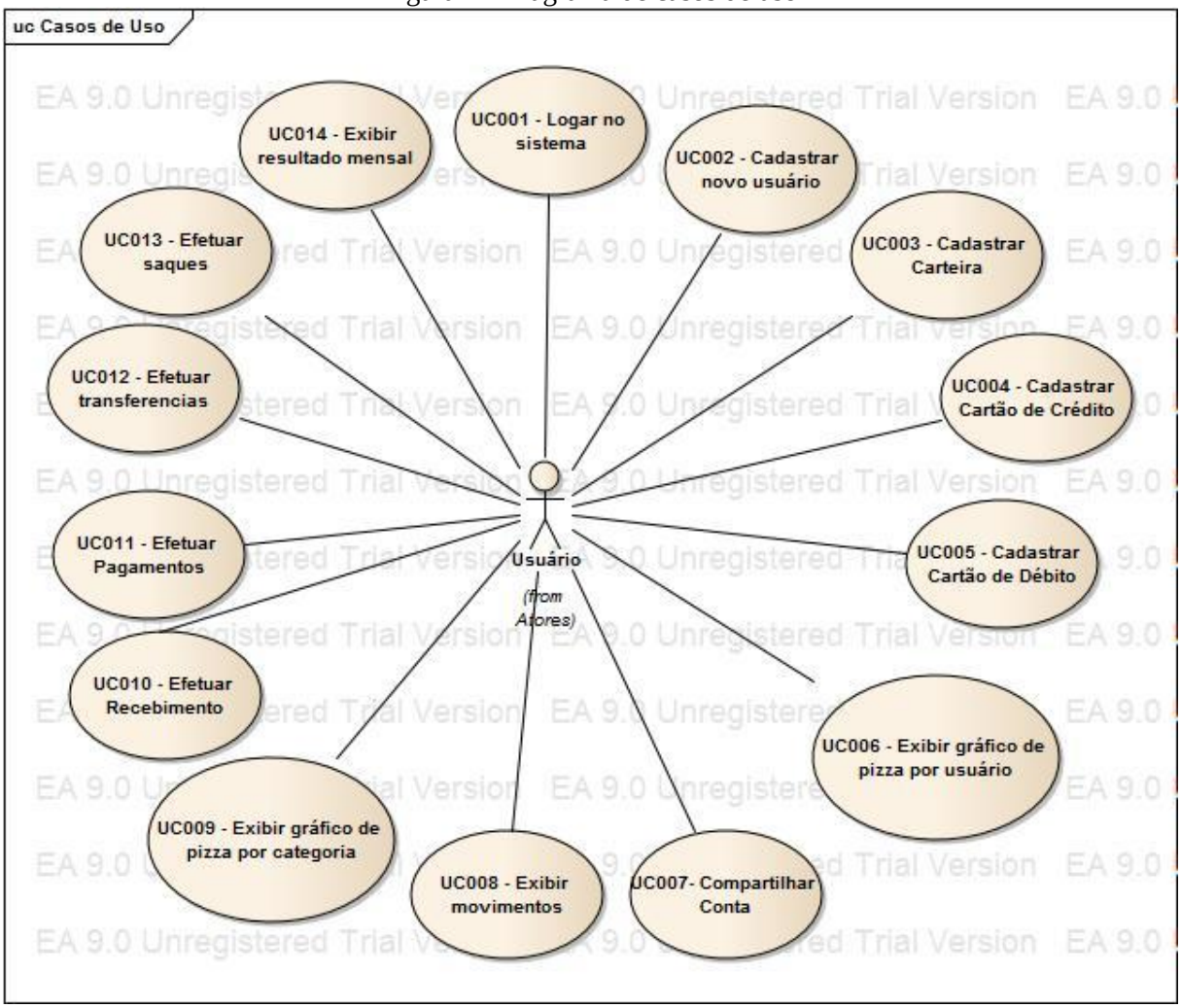
Requisitos não funcionais
RNF001 - O sistema móvel deverá rodar na plataforma Android
RNF002 - A versão mínima do Android para o sistema móvel é a 4.4.2.
RNF003 - Nenhuma informação de qualquer usuário poderá ser visualizada sem estar logado.

3.2.2 Diagrama de casos de uso

Esta seção apresenta o diagrama de casos de uso. A Figura 7 apresenta o diagrama de casos de uso do ator `Usuário`.

No Apêndice A são detalhados os principais casos de uso do aplicativo.

Figura 7 - Diagrama de casos de uso

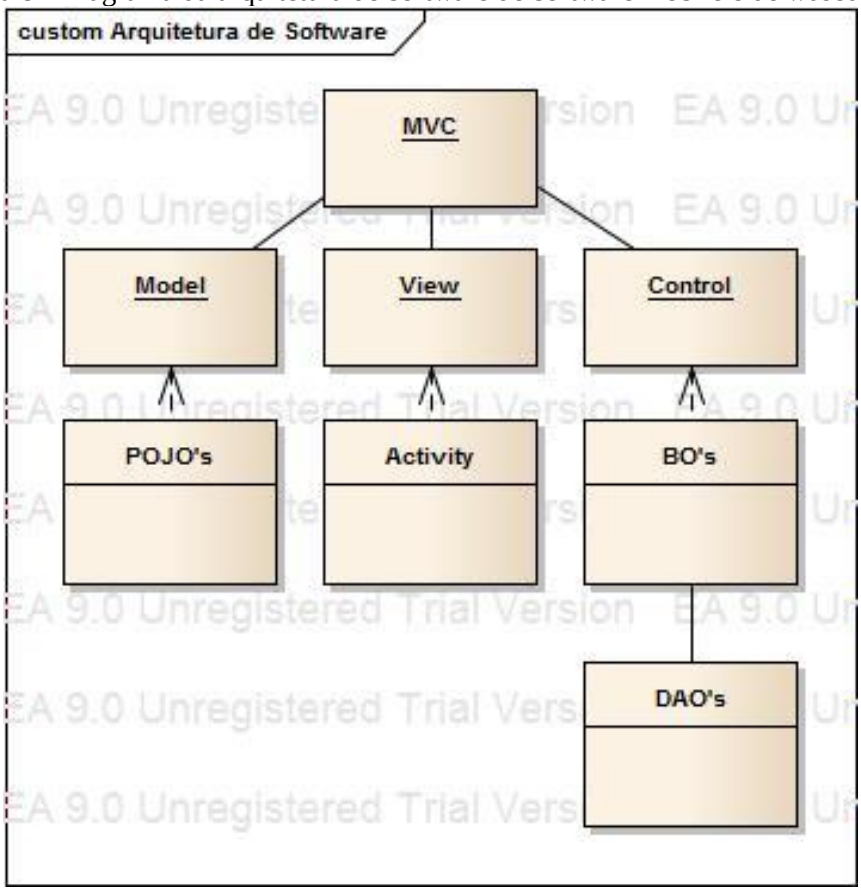


3.2.3 Projeto

Nesta seção são apresentados diagramas utilizados para elaboração do projeto do software *mobile* e do *webservice*. A arquitetura utilizada foi a MVC (Model, View e Controller) indicando que o software está dividido em camadas. Na camada `Model` há as classes responsáveis pela leitura e escrita de dados. Na camada View há as classes responsáveis pela exibição dos dados. Já a camada `Control` possui as classes responsáveis pelo controle e tráfego das informações.

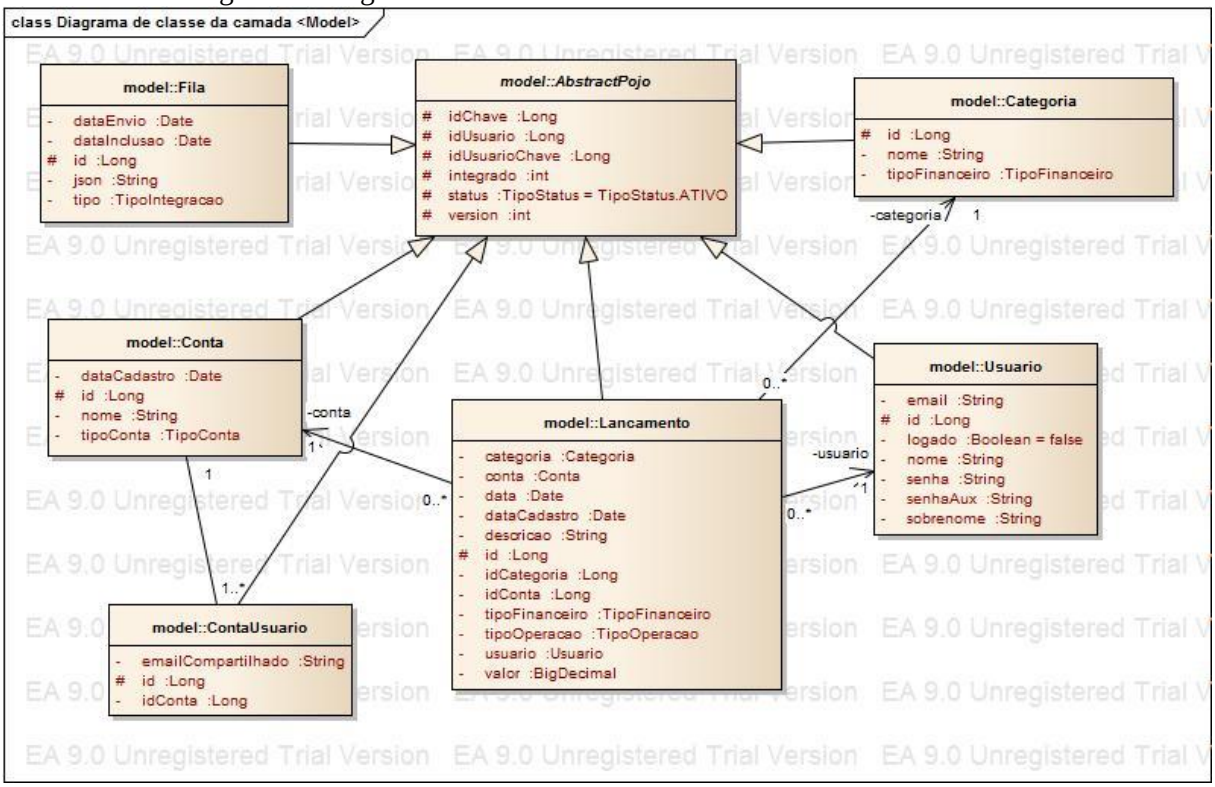
A Figura 8 mostra o padrão *Model-View-Control* (MVC) utilizado na arquitetura do software.

Figura 8 - Diagrama da arquitetura do software do software mobile e do *webservice*.



No software *mobile* a camada `Model` é representada pelas classes que estendem `AbstractPojo`, a camada `View` é representada pelas classes que estendem `Activity` e a camada `Control` é representado pelas classes que estendem `AbstractBO`. A Figura 9 ilustra o diagrama de classes da camada `Model`.

Figura 9 - Diagrama de classes da camada `Model` do software mobile



A seguir é apresentado uma breve descrição das classes criadas:

a) `AbstractPojo`: classe abstrata que contém códigos em comum de todas as classes, como por exemplo, todos os registros precisam ter um `idUsuario`;

b) `Fila`: classe que armazena os objetos para sincronização, podem ser de entrada ou saída;

c) `Categoria`: classe utilizada para categorizar os lançamentos, podem ser do tipo de entrada sou saída;

d) `Conta`: classe utilizada para gravar os dados de uma conta, as contas possuem um tipo que podem ser carteira, poupança, conta-corrente, cartão de crédito ou cartão de débito;

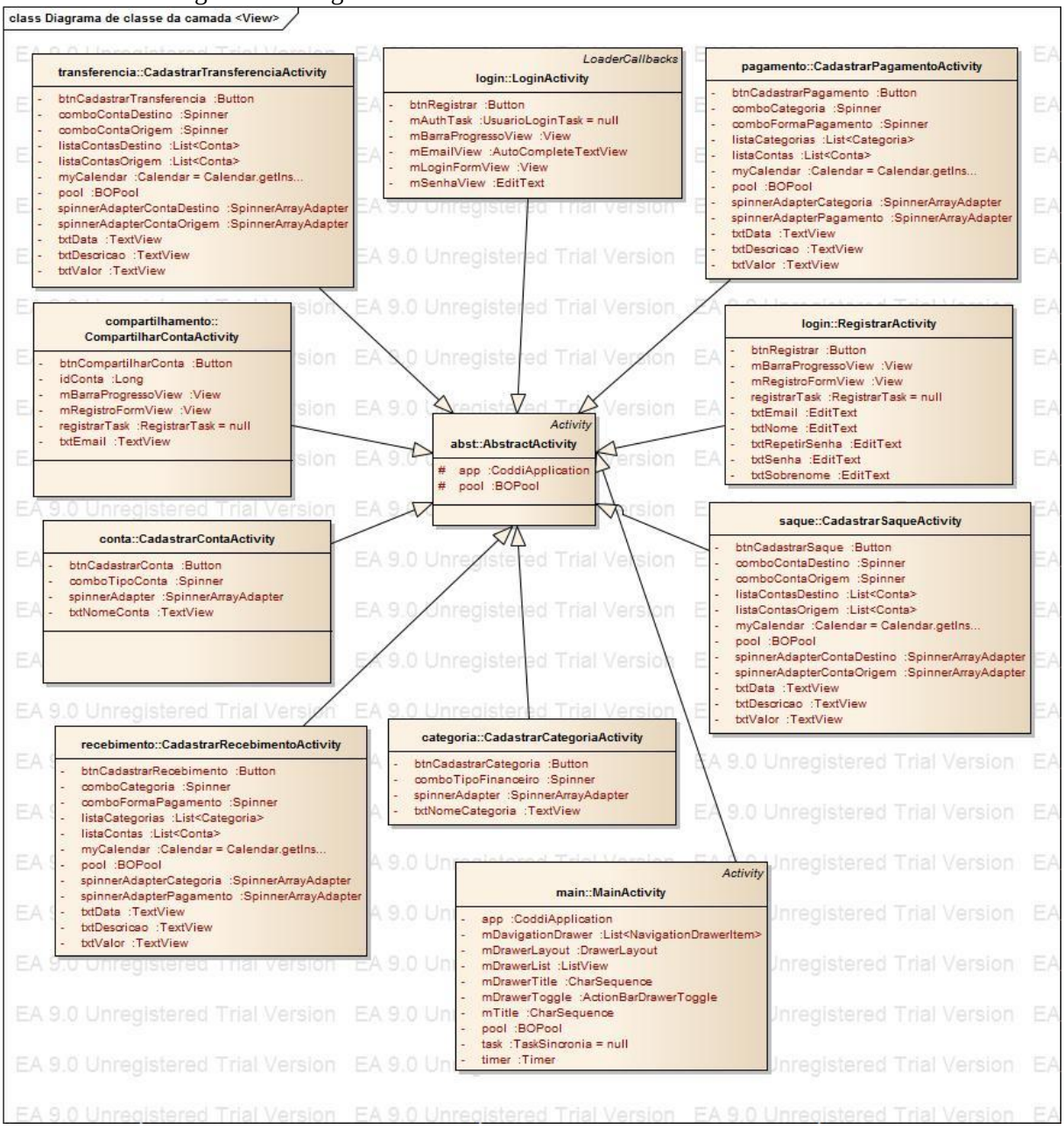
e) `Usuario`: classe utilizada para gravar os dados de um usuário;

f) `ContaUsuario`: classe intermediária que faz a ligação de contas a usuário, utilizada no compartilhamento de uma conta;

g) `Lancamento`: classe utilizada para gravar dados de um lançamento. É diferenciado pelo seu tipo que pode ser de pagamento, saque, transferência ou recebimento.

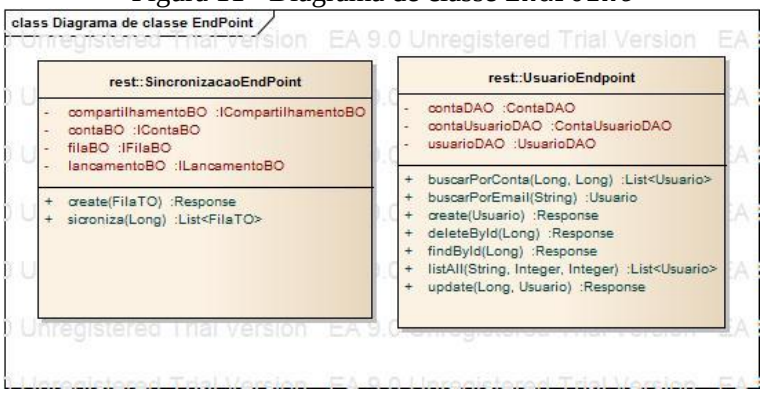
A Figura 10 ilustra o diagrama de classe da camada `View`.

Figura 10 - Diagrama de classe da camada `View` do software mobile.



A classe `MainActivity` é responsável por gerenciar as classes que serão mostradas, e também gerencia o menu do sistema. Todas as classes do modelo `View` são estendidas da classe genérica `AbstractActivity`. A classe `CompartilharContaActivity` é responsável por exibir ao usuário a interface de compartilhamento das contas. A camada `View` do *webservice* pode de certa forma ser representado pelas classes que expõe suas *URLs*. A figura Figura 11 ilustra o diagrama de classes das classes `EndPoint`.

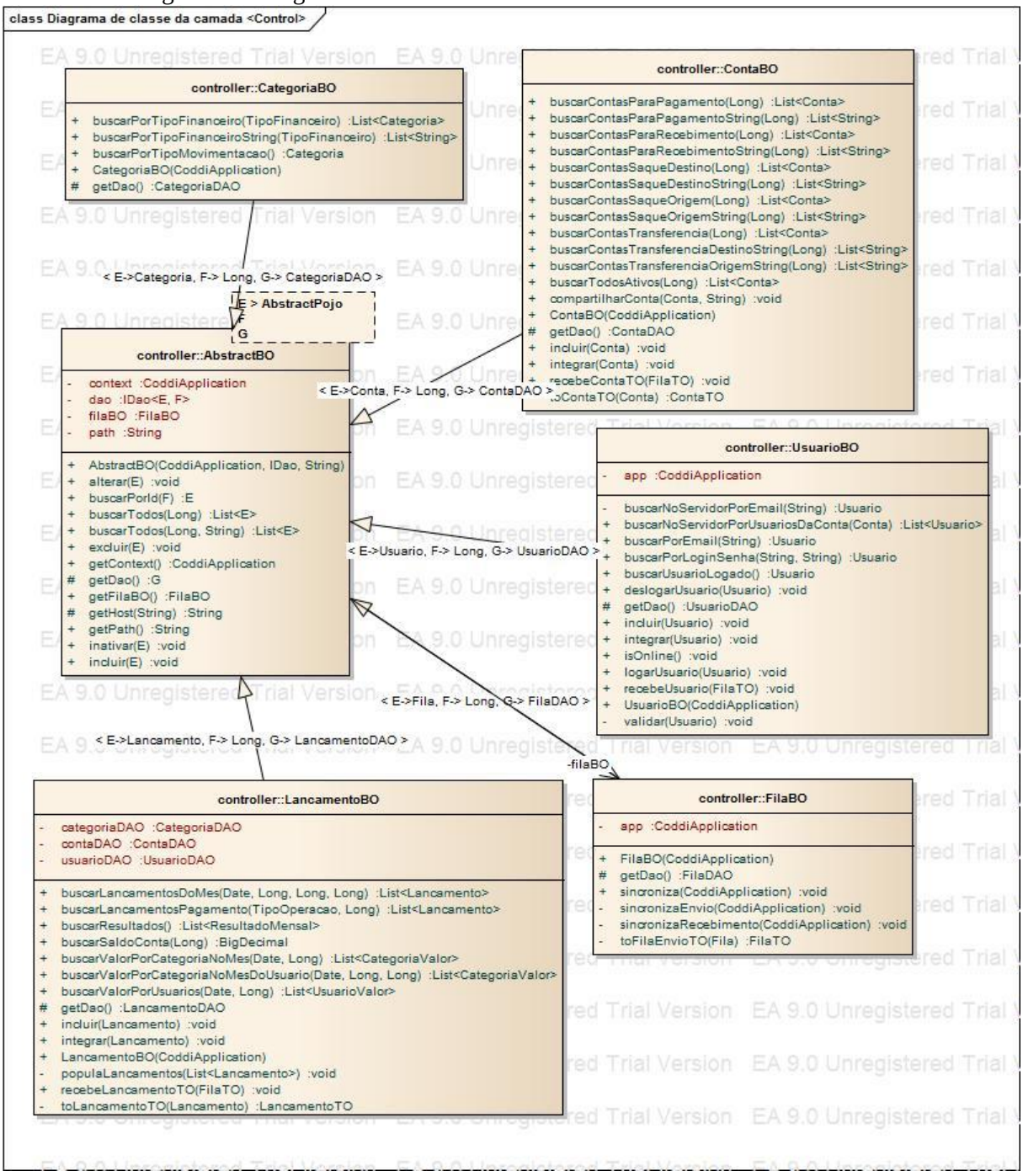
Figura 11 - Diagrama de classe `EndPoint`



Uma classe *EndPoint* é responsável por expor uma *URL* de comunicação. A classe `SincronizacaoEndPoint` é responsável pela *URL* `@Path("/sincroniza")` e a classe `UsuarioEndpoint` é responsável pela *URL* `@Path("/usuarios")`. É através destas *URLs* que o software *mobile* se comunica com os *webservices*.

A Figura 12 ilustra o diagrama de classe da camada `Control`.

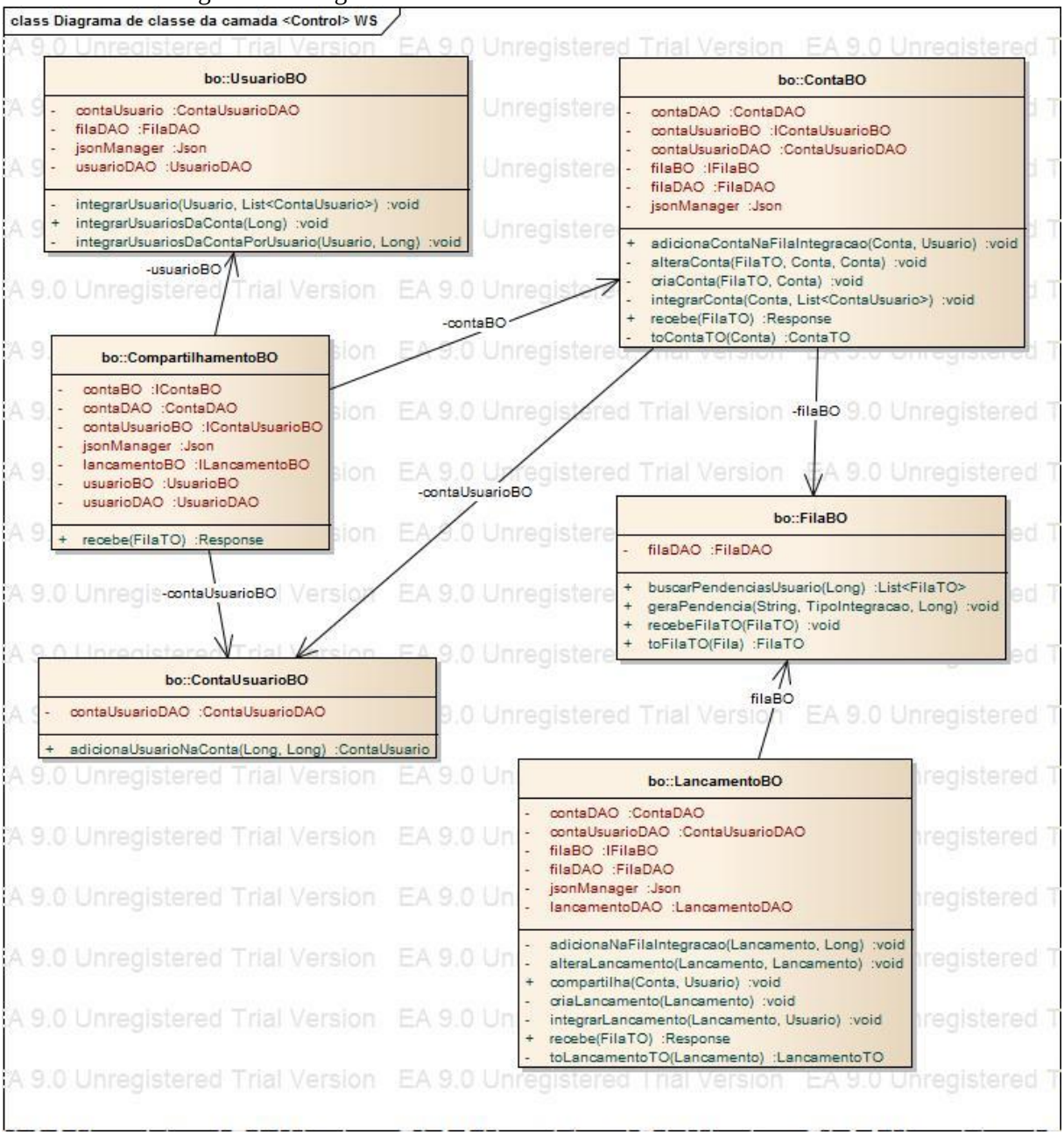
Figura 12 - Diagrama de classes da camada `Control` do software mobile.



As classes de controle são responsáveis por gerenciar a regra de negócio do sistema. São indicadas por estender a classe `AbstractBO`. Estas classes contém todo o mecanismo por trás das funcionalidades do sistema. Para cada nova entidade no sistema é criada uma classe *Business Object* (BO) para atender sua demanda de negócio.

A Figura 13 ilustra a camada de controle do *webservice*.

Figura 13 - Diagrama de classes da camada `Control` do *webservice*.



As classes da camada *Control* do *webservice* não estendem da classe *AbstractBO*, porém apresentam a mesma nomenclatura. A maior diferença entre os projetos está na forma em que são os dados são persistidos. Enquanto no software *mobile* a classe pai `IDao` é responsável pela persistência, no webservice a classe responsável é a `EntityManager`.

3.2.4 Persistência

No software *mobile* a camada de persistência fica acoplada as classes `DAO`. São responsáveis por incluir, alterar e excluir objetos do banco de dados. As tabelas do banco de

dados são criadas através de annotations nas classes modelo. O Quadro 3 apresenta a classe `Lancamento` com a devidas annotations de persistência.

Quadro 3 - Classe `Lancamento` do Software *mobile*

```
@DatabaseTable(tableName = "Lancamento")
public class Lancamento extends AbstractPojo {

   @Expose
   @DatabaseField(generatedId = true)
   protected Long id;

   @DatabaseField(width = 40)
   private String descricao;

   @DatabaseField
   private TipoFinanceiro tipoFinanceiro;

   @DatabaseField
   private Date data;

   @DatabaseField
   private Date dataCadastro;

   @DatabaseField
   private BigDecimal valor;

   @DatabaseField
   private Long idCategoria;

   @DatabaseField
   private Long idConta;
}
```

A seguir é apresentado a devida descrição de cada *annotation*:

a) `@DatabaseTable` é responsável pelo nome da tabela no banco de dados;

b) `@DatabaseField` é responsável pelo nome da coluna no banco de dados;

c) `@Expose` informa quais campos serão expostos ao json, não faz parte da persistência.

A persistência dos dados do *webservice* é feita de maneira semelhante. O Quadro 4 apresenta como exemplo a classe `Conta` do *webservice*.

Quadro 4 - Classe `Conta` do webservice

```
@Entity
@XmlRootElement
public class Conta implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "id", updatable = false, nullable = false)
	private Long id;

	@Version
	@Column(name = "version")
	private int version;

	@Column
	private String nome;

	@Convert(converter = TipoContaEnumConverter.class)
	private TipoConta tipoConta;

	@Column
	@Temporal(TemporalType.TIMESTAMP)
	private Date dataCadastro;

	@Column
	private Long idChave;

	@Column
	private Long idUsuarioChave;

	@Convert(converter = TipoStatusEnumConverter.class)
	private TipoStatus status;
}
```

A descrição de cada *annotation* utilizada é a seguinte:

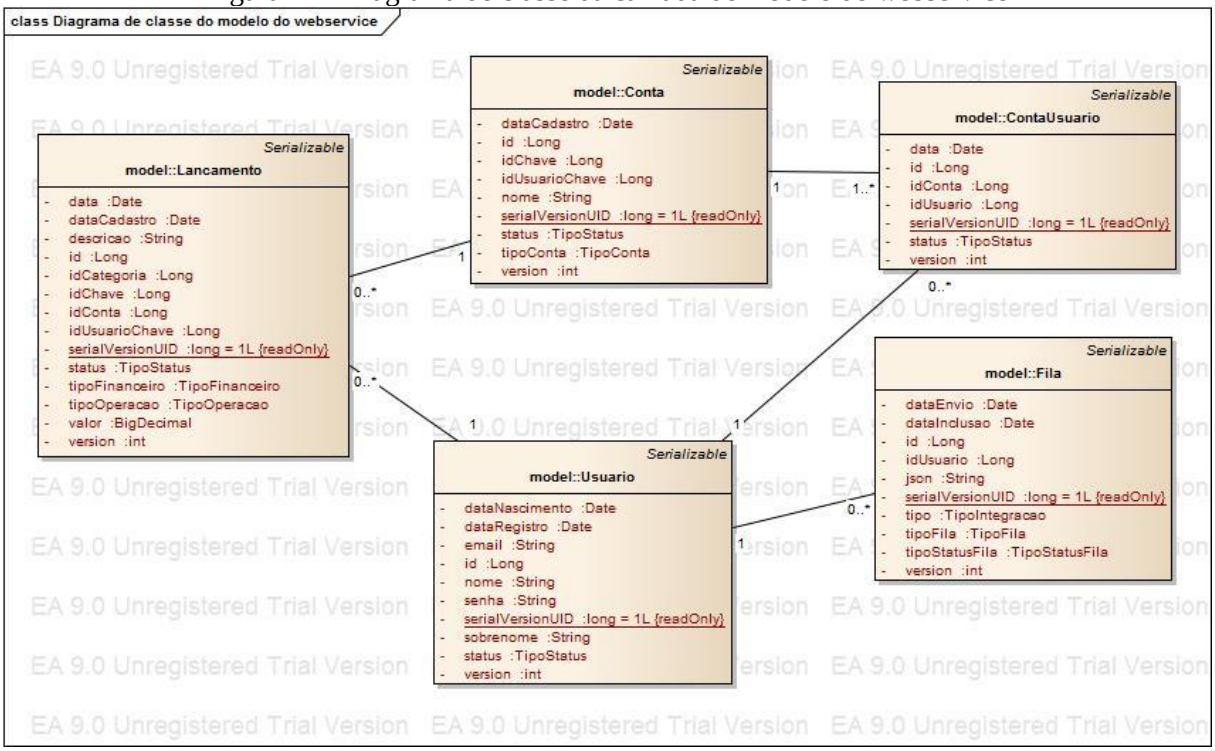
a) `@Entity` é a identificação que a classe é uma entidade;

b) `@Column` é a identificação de que o campo é uma coluna na tabela;

c) `@Id` identifica a coluna chave da tabela.

A Figura 14 ilustra o diagrama de classe da camada modelo do *webservice* que podem ser comparados as tabelas do banco de dados.

Figura 14 - Diagrama de classe da camada de modelo do webservice



3.3 IMPLEMENTAÇÃO

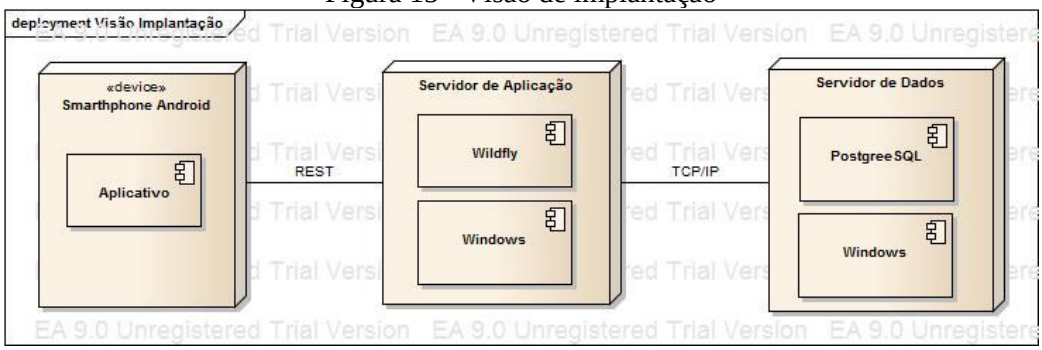
A seguir são apresentadas ferramentas e técnicas utilizadas para implementação e funcionalidades do aplicativo.

3.3.1 Técnicas e ferramentas utilizadas

Para a aplicação web foi utilizado o banco de dados PostgreeSQL juntamente com servidor de aplicação WildFly. O *webservice* foi construído em Java e os dados persistidos com auxílio do Java Persistence API (JPA).

O aplicativo Android foi construído em Java (nativo) com banco de dados SQLite. Para persistência dos dados foi utilizada a biblioteca ORMlite. A Figura 15 ilustra a visão de implantação onde é possível identificar a interação das tecnologias utilizadas.

Figura 15 - Visão de implantação



3.3.1.1 Sincronização

Quando um usuário se cadastra no sistema, ele se registra no servidor recebendo uma chave única, armazenada no campo `id` da tabela `Usuario`. Esta chave é armazenada em todos os registros do aplicativo Android, garantindo que apenas o usuário com aquela chave visualize suas informações.

Quando o usuário registra uma nova conta no sistema, um `idUsuarioChave` e um `idChave` é registrado internamente para aquele registro. O `idUsuarioChave` armazena o `id` do usuário que cadastrou aquela informação e o `idChave` é o `id` daquele registro no dispositivo do usuário, ou seja, uma chave composta de um identificador global e um identificador local. Quando esta conta é enviada para o servidor estes campos continuam iguais, não havendo necessidade de reenviar um campo do tipo código de referência para o dispositivo.

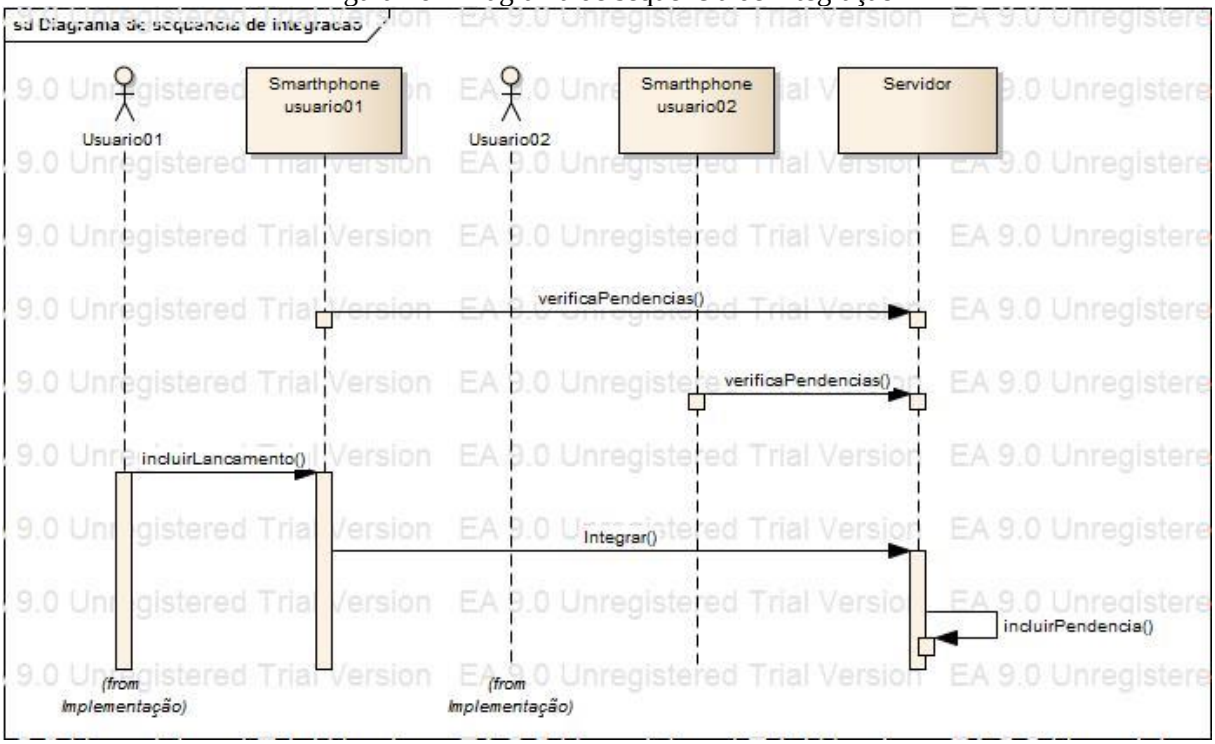
Quando o usuário dono da conta envia ao servidor uma solicitação de compartilhamento de conta, ele se torna o usuário de origem da conta, já o usuário da solicitação se torna o usuário de destino. Ao receber a solicitação, o servidor replica a conta solicitada pelo usuário de origem para o usuário de destino que recebe a conta com os mesmos `idUsuarioChave` e `idChave` da criação. Novamente nenhum tipo de código de referência precisaria ser enviado, uma vez que a chave composta continua a mesma e é enviada para todo os usuários.

Quando um lançamento é criado e replicado para os outros dispositivos fica fácil de identificar o usuário de origem do lançamento, basta identificar a coluna `idUsuarioChave`. Fazendo um `join` com a tabela `Usuario` é possível visualizar outras informações como nome e sobrenome do usuário.

3.3.1.2 Pendências e mensagens

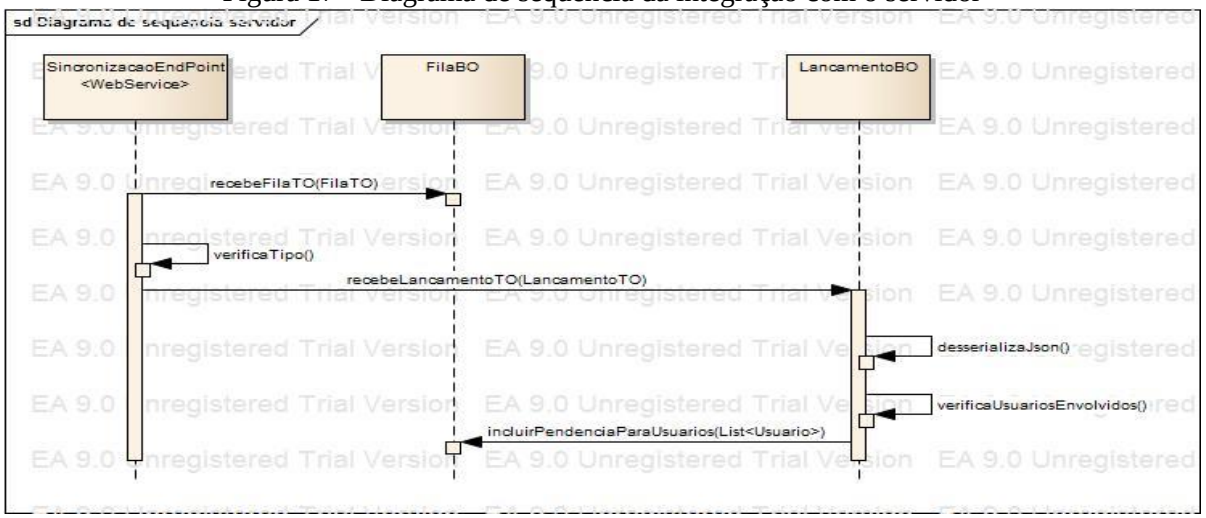
A troca de mensagens entre o *webservice* e o aplicativo Android é feita através da geração de pendência, ou seja, o responsável pela integração é sempre o aplicativo consumidor. A Figura 16 ilustra a maneira com que as integrações ocorrem.

Figura 16 - Diagrama de sequência de integração



Sem o usuário sequer ter feito operações, o aplicativo está constantemente se comunicando com o servidor a fim de efetivar a integração das pendências. Todas as pendências são serializadas em forma de texto e persistidos para o objeto `FilaTO`. Toda integração assíncrona acontece após transformar o objeto a ser integrador para um objeto do tipo `FilaTO`. A Figura 17 ilustra o recebimento de uma `FilaTO` no servidor.

Figura 17 - Diagrama de sequência da integração com o servidor



Após receber um objeto do tipo `FilaTO`, o *webservice* identifica o tipo da pendência e chama as classes *Business Object* responsáveis pelo processo. No caso de receber a pendência de lançamento, o primeiro passo é desserializar a `FilaTO` para incluí-la no banco de dados. A classe `LancamentoBO` é chamada para que as validações ocorram. Após verificar se todos os

campos estão corretos, verifica se aquele registro precisa ser replicado para algum outro usuário, sem sim, inclui as devidas pendências. O Quadro 5 apresenta o código que mostra a implementação do *webservice* de recebimento pendência.

Quadro 5 - Webservice de recebimento de pendência

```
@POST
@Transactional
@Consumes("application/json")
public Response create(FilaTO de) {
	filaBO.recebeFilaTO(de);

	TipoIntegracao tipo = de.getTipo();
	switch (tipo) {
	case CONTA:
		return contaBO.recebe(de);
	case LANCAMENTO:
		return lancamentoBO.recebe(de);
	case COMPARTILHAR_CONTA:
		return compartilhamentoBO.recebe(de);
	default:
		return Response.status(Status.BAD_REQUEST).build();
	}
}
```

A `annotation @POST` significa que o *webservice* vai executar o comando POST do HTTP, ou seja, incluir um novo objeto. A `annotation @Transactional` significa que o comando pertence a uma única transação de banco de dados, e qualquer erro que ocorra o *rollback* será total. A `annotation @Consumes("application/json")` indica que o retorno deste método será um arquivo json e em caso de não encontrar um tipo de pendência válido o código 400 é retornado.

3.3.1.3 Persistência

Quando o software é instalado pela primeira vez no *smarthphone* um código de criação de tabela é executado. Cada tabela precisa ser explicitamente criada no código fonte. O Quadro 6 mostra o código fonte utilizado na classe `DataBaseHelper` que é responsável pelos *scripts* de criação e atualização de base no software *mobile*.

Quadro 6 - Classe `DataBaseHelper` responsável pela criação e atualização da base.

```
public class DatabaseHelper extends OrmLiteSqliteOpenHelper {

   private static final String DATABASE_NAME = "coddi.sqlite";
   private static final int DATABASE_VERSION = 137;

   public DatabaseHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase database, ConnectionSource connectionSource) {
      try {
         TableUtils.createTable(connectionSource, Usuario.class);
         TableUtils.createTable(connectionSource, Categoria.class);
         TableUtils.createTable(connectionSource, Conta.class);
         TableUtils.createTable(connectionSource, Lancamento.class);
         TableUtils.createTable(connectionSource, Fila.class);
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }

   @Override
   public void onUpgrade(SQLiteDatabase database, ConnectionSource connectionSource, int oldVersion, int
newVersion) {
      try {
         TableUtils.dropTable(connectionSource, Usuario.class, true);
         TableUtils.dropTable(connectionSource, Categoria.class, true);
         TableUtils.dropTable(connectionSource, Conta.class, true);
         TableUtils.dropTable(connectionSource, Lancamento.class, true);
         TableUtils.dropTable(connectionSource, Fila.class, true);

         onCreate(database, connectionSource);
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }
}
```

O método `onCreate()` é executado sempre que o aplicativo é instalado pela primeira vez. O método `onUpgrade()` é executado sempre que a versão atual é diferente da versão futura, inclusive, na sua assinatura há explicitamente a versão atual e a versão futura da base, possibilitando assim uma conversão segura.

No caso do Quadro 6, em todas as atualizações de base as tabelas são deletadas e criadas novamente. O modelo de entidade-relacionamento da base de dados fica idêntico ao diagrama de classes da Figura 9, onde cada classe se torna uma tabela e cada atributo se torna uma coluna no banco de dados.

No *webservice* a responsabilidade de indicar como a criação de base deve ocorrer é feita em um arquivo `persistence.xml` conforme o Quadro 7.

Quadro 7 - Persistence.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.1"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="Coddi-persistence-unit" transaction-type="JTA">
    <description>Forge Persistence Unit</description>
    <jta-data-source>java:jboss/datasources/postgresql</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action"
            value="none" />
      <property name="javax.persistence.schema-generation.scripts.action" value="drop-and-create"/>
      <property name="javax.persistence.schema-generation.scripts.create-target" value="CoddiCreate.ddl"/>
      <property name="javax.persistence.schema-generation.scripts.drop-target" value="CoddiDrop.ddl"/>
    </properties>
  </persistence-unit>
</persistence>
```

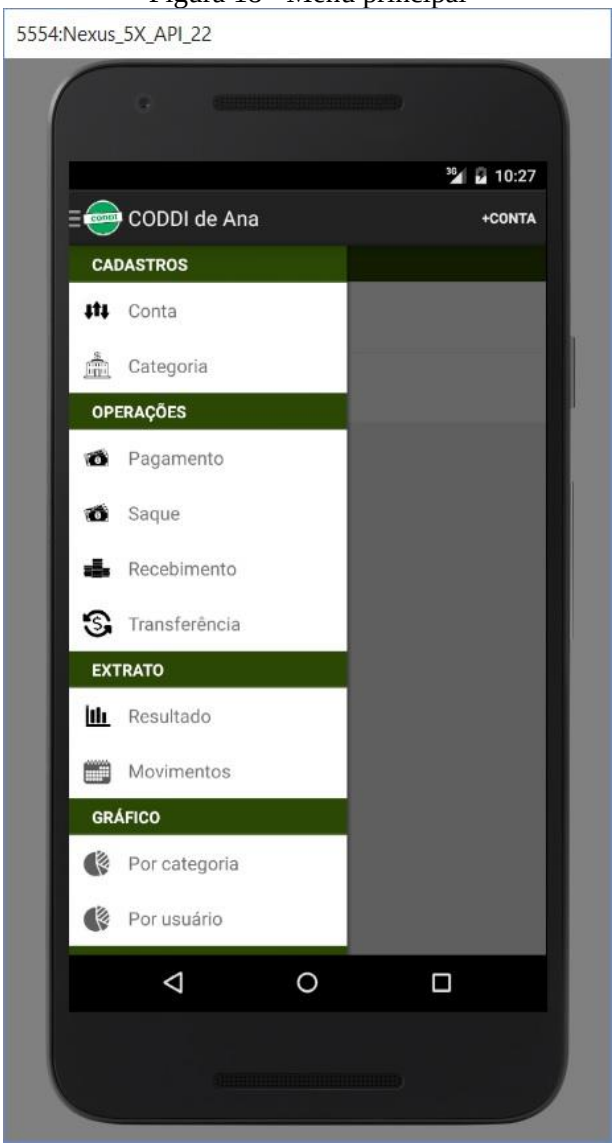
A *tag* `<property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />` indica o que acontece durante a geração do *schema* da base. Neste trabalho a opção utilizada foi a *drop-and-create* indicando que em todas as inicializações do WildFly o mecanismo de persistência recriará totalmente a base de dados. As configurações de conexão com o banco também estão disponíveis nessa classe.

As classes Java com a *annotation* `@Entity` são traduzidas em tabelas do banco via reflexão e, diferentemente do software *mobile* não precisam ser criadas explicitamente.

3.3.2 Operacionalidade da implementação

O aplicativo é iniciado com a tela de login. Após cadastrado o usuário é apresentado ao menu principal do sistema como na Figura 18.

Figura 18 - Menu principal



As opções de menu são:

a) conta: criação de compartilhamento de conta;
b) categoria: visualização de categorias;
c) pagamento: adiciona e visualiza novos pagamentos;
d) saque: adiciona e visualiza saques;
e) recebimento: adiciona e visualiza recebimentos;
f) transferência: adiciona e visualiza transferências entre contas;
g) resultados: mostra o resultados por mês de todas as despesas e receitas;
h) movimentos: exibe todos os movimentos do mês;
i) relatório por categoria: exibe um relatório de pizza com o somatório dos gastos por categoria;
j) relatório por usuário: exibe um relatório de pizza com o somatório dos gastos por

usuário.

3.3.2.1 Operações

As operações são as maneiras com que o usuário identificará suas transações diárias. As operações disponíveis no aplicativo são:

a) pagamento;
b) saque;
c) recebimento;
d) transferência.

Um pagamento é sempre um gasto do tipo despesa. Possui uma data, uma descrição, uma categoria do tipo despesa e uma conta de saída. Suas formas de pagamento são dinheiro, cartão de débito ou cartão de crédito.

Um saque é sempre uma operação de dois lançamentos, um do tipo despesa e outro do tipo receita. Possui uma data, uma descrição, uma conta de origem e uma conta de destino. Saques não são identificados por categoria. A conta de origem é sempre uma conta do tipo conta corrente ou conta poupança e a conta de destino é sempre uma conta do tipo carteira.

Um recebimento é sempre do tipo receita. É utilizado quando o usuário recebe valor em algumas das contas. Possui uma data, uma descrição, uma conta de destino. A conta de destino pode ser uma carteira (em dinheiro), uma conta corrente (depósito) ou conta poupança (depósito).

Uma transferência não possui categoria. Ocorre na forma de dois lançamentos, um de despesa e outro de receita. Possui uma data, uma descrição, uma conta de origem do tipo carteira ou conta corrente e uma conta de destino do tipo carteira, conta corrente ou conta poupança.

3.3.2.2 Formas de pagamento

As formas de pagamento são uma mescla entre as contas e seu tipo. As contas do tipo carteira por exemplo, se transformam na forma de pagamento dinheiro. Abaixo cada tipo de conta e sua respectiva forma de pagamento:

a) carteira: em dinheiro;
b) cartão de crédito: cartão de crédito;
c) cartão de débito: cartão de débito;
d) conta corrente: saque e depósito;
e) conta poupança: saque e depósito.

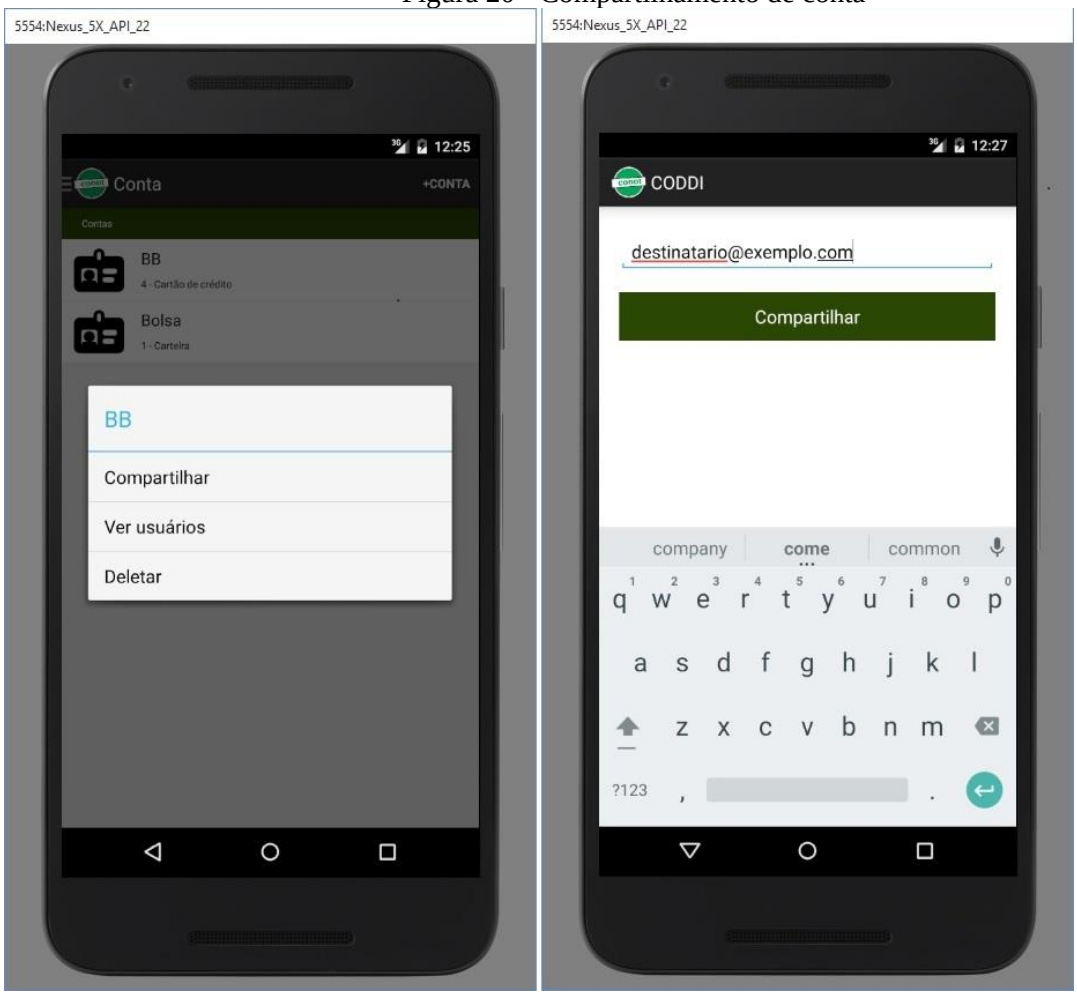
A Figura 19 ilustra as formas de pagamento do tipo dinheiro e cartão.

Figura 19 - Pagamento em dinheiro

3.3.2.3 Compartilhamento de conta

O compartilhamento de uma conta possibilita o aplicativo trabalhar de forma multiusuário. Para compartilhar uma conta é necessário conhecer o e-mail do destinatário com quem se deseja compartilhar a conta. A Figura 20 ilustra o menu onde é possível visualizar a opção de compartilhar a conta.

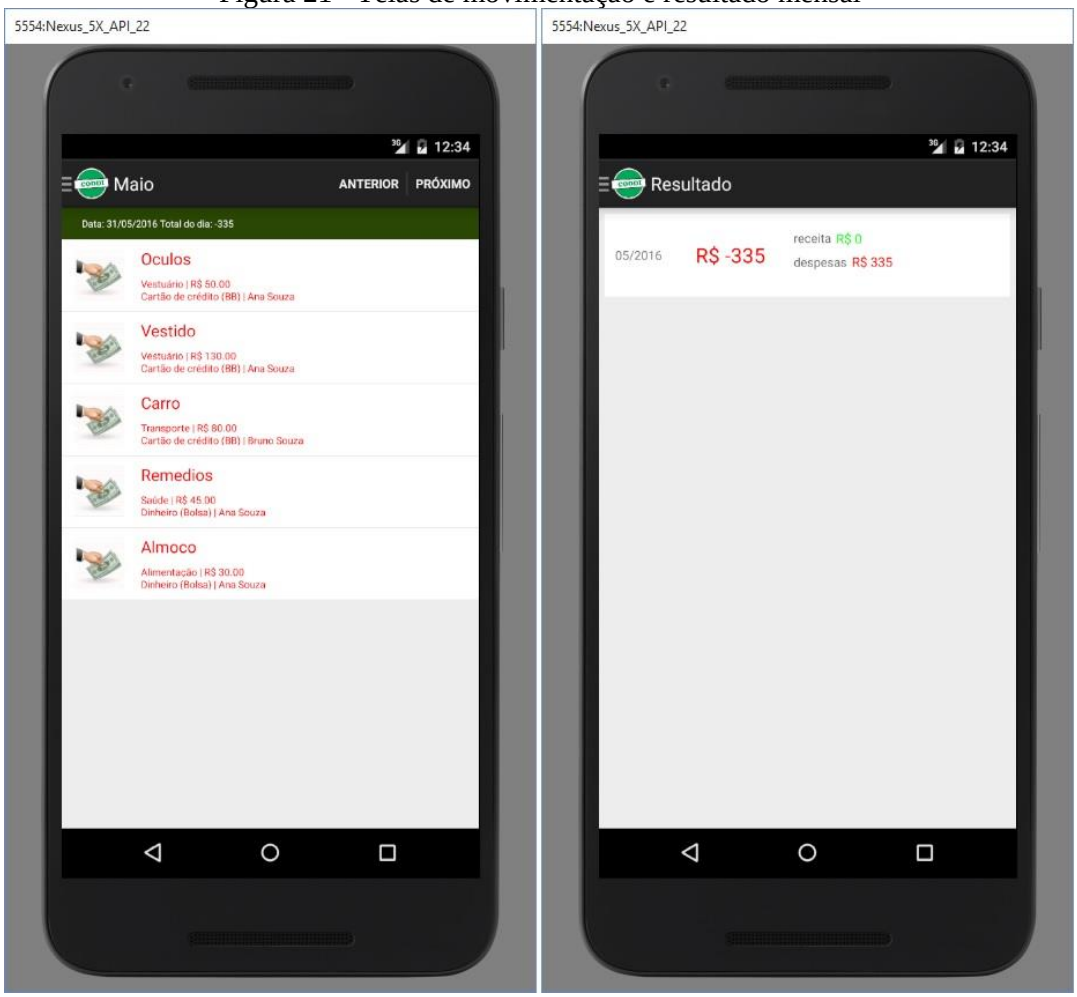
Figura 20 - Compartilhamento de conta



3.3.2.4 Relatórios e movimentações

As movimentações podem ser visualizadas através do menu “Movimentos”. Os movimentos podem ser do tipo despesa ou receita. Na tela de movimentos os registros estão agrupados por data em ordem decrescente. Nesta tela é possível identificar o usuário que efetuou a movimentação pelo seu respectivo nome. A Figura 21 ilustra as telas de movimentação e resultado mensal.

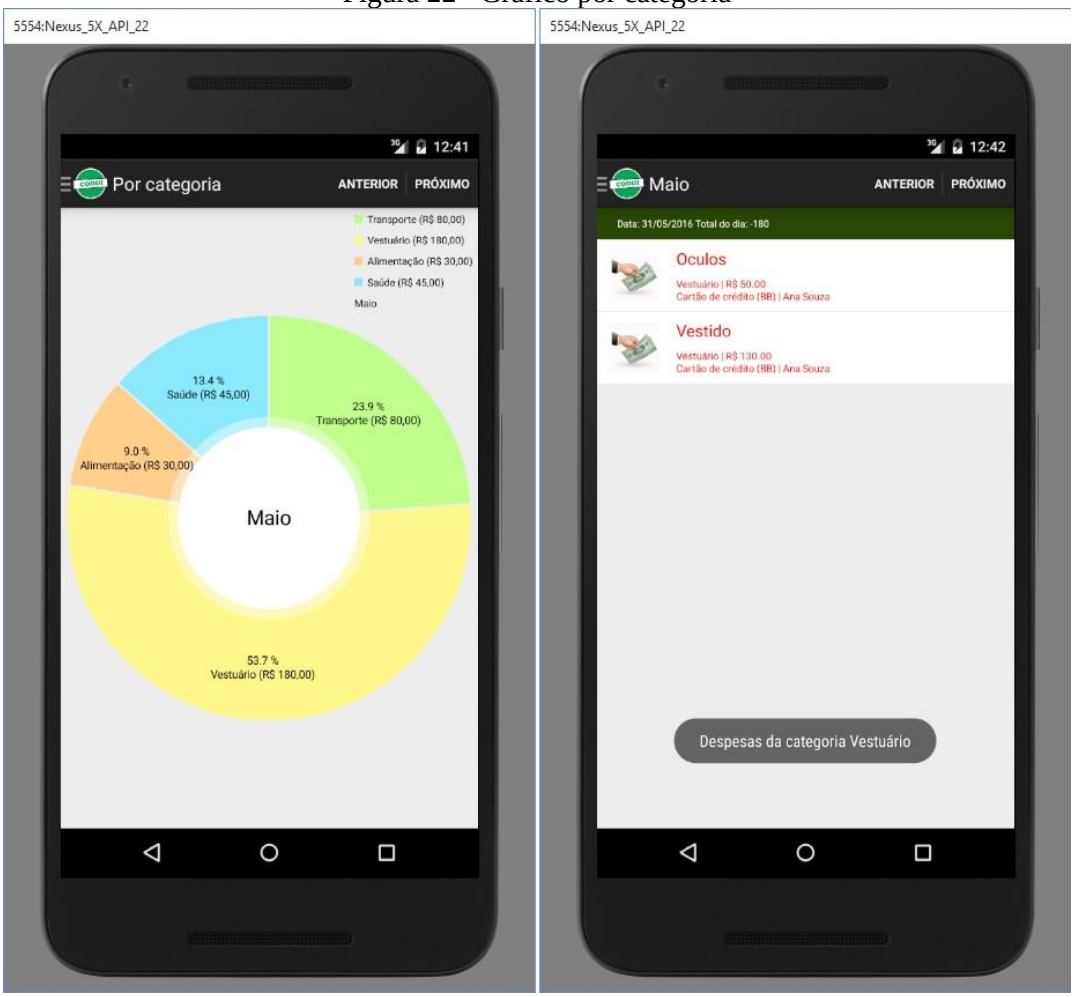
Figura 21 - Telas de movimentação e resultado mensal



A tela de resultado mensal informa ao usuário o saldo geral de suas despesas e receitas agrupado pelo mês em que ocorreu. As despesas e receitas de outros usuários também são somadas para o resultado.

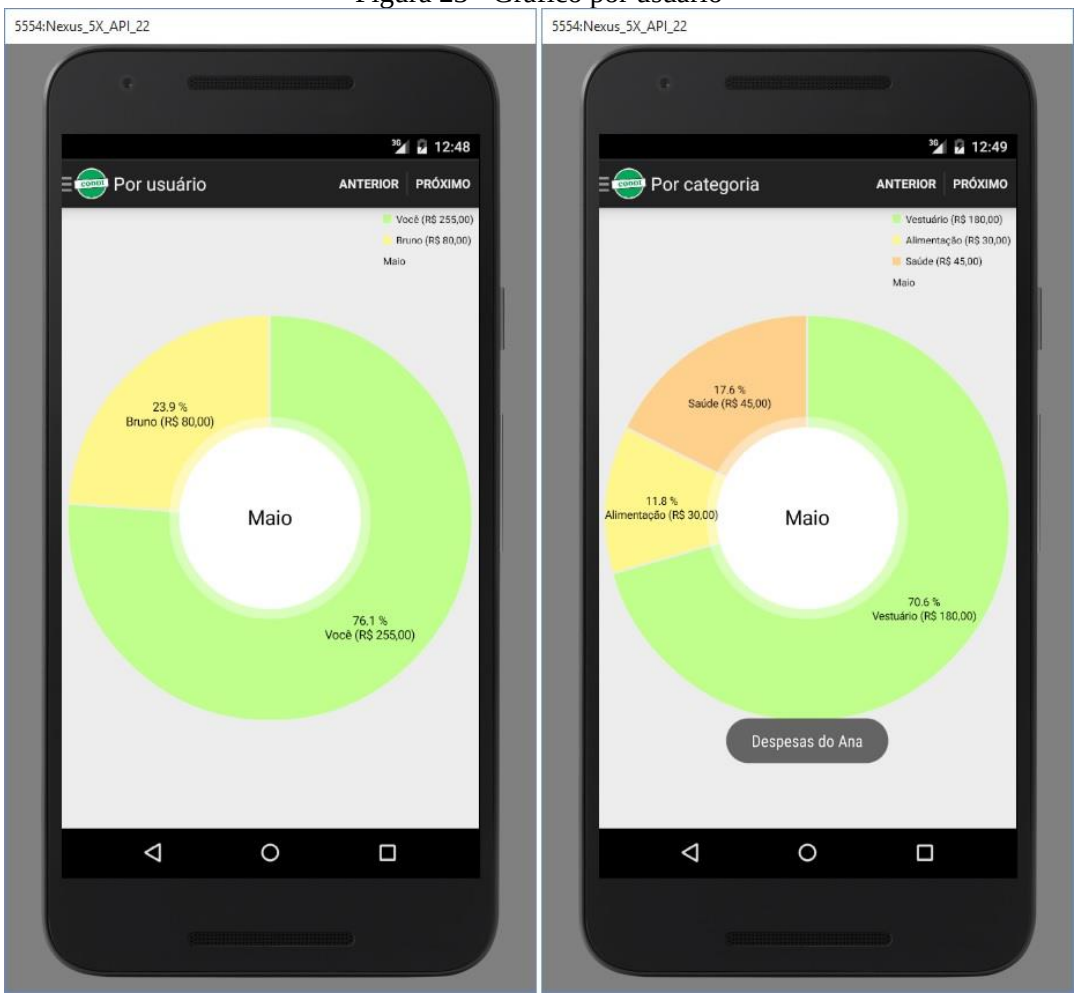
Para um maior controle das despesas o usuário poderá visualizar dois tipos de relatório agrupados por usuário ou por categoria. Em ambos os relatórios é possível mostrar mais informações sobre cada usuário ou categoria. A imagem ilustra o gráfico agrupado por categorias e os detalhes exibidos ao especificar uma categoria do relatório.

Figura 22 - Gráfico por categoria



O gráfico agrupado por usuário possui um nível a mais que o gráfico por categoria. Nele é possível especificar apenas as categorias e despesas de um único usuário. A Figura 23 ilustra o gráfico agrupado por usuário e também o detalhe de apenas um usuário.

Figura 23 - Gráfico por usuário



3.4 RESULTADOS E DISCUSSÕES

Este trabalho obteve sucesso no que diz respeito ao atendimento dos requisitos e objetivos propostos. A implementação como um todo contemplou a sincronização dos dados, replicação e inclusão de dados em outro dispositivo. Mostrou-se funcional em ambientes de testes, no entanto, precisaria de alguns ajustes para fazer parte da plataforma de download do Android, o Google Play.

A comparação com os trabalhos correlatos é apresentada no Quadro 8. As funcionalidades básicas de um aplicativo financeiro são comuns em ambos os *softwares*. Alguns dos softwares correlatos apresentam funcionalidades utilizadas como extensões deste trabalho. A principal funcionalidade que se destaca do software desenvolvido e relação aos correlatos é a característica multiusuário.

O software desenvolvido é capaz de identificar lançamentos realizados por outros usuários compartilhando uma mesma conta. Porém tanto o Mobills, quanto o Minhas

Economias conseguem trabalhar de forma multidispositivo o que possibilita duas pessoas movimentarem uma mesma conta, mas sem controle e rastreabilidade.

Quadro 8 - Comparação com trabalhos correlatos

Funcionalidade	**CODDI**	**Mobills**	**Minhas Economias**
Lançamentos de despesas e receitas	X	X	X
Categorias de despesa e receita	X	X	X
Gerenciamento de contas	X	X	X
Extrato de movimentos	X	X	X
Multiusuário	X		
Off-line	X	X	
Exportação para Excel		X	
Multidispositivo		X	X

Por fim é possível é possível diferenciar a ferramenta desenvolvida pelas suas características diferenciadas em relação aos correlatos.

CONCLUSÕES

Os objetivos propostos foram alcançados. O aplicativo realiza operações off-line, as integra assim que encontra conexão. É capaz de compartilhar conta entre usuários e mostrar gráficos de despesas.

O principal desafio da implementação de um dispositivo sincronizado com o servidor é o desenvolvimento assíncrono e multiusuário. As validações, pendências e processos devem ser pensadas com muita atenção e cautela. É preciso levar em consideração que o dispositivo local do usuário nem sempre está com as últimas atualizações do *webservice* e algumas validações devem prever este comportamento.

As operações são sincronizadas em formas de pendência prevendo o uso do aplicativo em modo off-line. As pendências integram de acordo com sua prioridade, evitando problemas após o envio.

O grande desafio da implementação deste trabalho foi referenciar o registro de uma maneira que se tornasse comum entre o dispositivo local e o *webservice*, sem a necessidade da transferência de código de referência, por exemplo. Códigos de referência são comuns em sistemas integrados e demandam um controle de estratégia da integração, pois podem haver registros prontos para serem integrados e que ainda não sabem seu devido código de referência no servidor. O aplicativo desenvolvido não possui nenhuma coluna no banco de dados para referenciar o registro no servidor, pois seu desenvolvimento foi pensado para isso.

Como ferramenta de sistema de controle de versão, o Git foi importante na validação de novas iterações de desenvolvimento. A cada nova funcionalidade um *branch* era criado e suas alterações eram repassadas para uma versão estável. No caso de instabilidade a ação de reverter o código é razoavelmente fácil.

Por fim o resultado foi como esperado, ou melhor, como planejado: um controle de despesas para a plataforma Android que pode operar *off-line* e com função multiusuário.

3.5 EXTENSÕES

Sugerem-se as seguintes extensões:

a) integração das compras feitas com cartão via SMS;
b) importar valor através da câmera do *smarthphone* de comprovantes fiscais e não fiscais;
c) adaptação do aplicativo para Google Play.

REFERÊNCIAS

ALMEIDA, T. **Mobills Gerenciador Financeiro**. 2014. Disponível em: <http://www.showmetech.com.br/review-mobills-gerenciador-financeiro/ >. Acesso em: 21 mar. 2016.

ÁVILA, R. **Como Fazer uma Planilha de Gastos no Excel**. 2015. Disponível em: <http://blog.luz.vc/como-fazer/ como-fazer-uma-planilha-de-gastos-excel/>. Acesso em: 19 mar. 2016.

BASTOS, R. **Gastos fantasmas: que são e como evitá-los**. 2011. Disponível em: <http://www.tribunadabahia.com.br/2011/11/21/ gastos-fantasmas--que-sao-e-como-evita-los>. Acesso em: 10 mar. 2016.

BRAGA, S. **Gastos fantasmas podem estourar o orçamento**. 2013. Disponível em: <http://www.calilecalil.com.br/calil/materias-publicas/materias. asp?ID=1315>. Acesso em: 10 mar. 2016.

CARVALHO, S. **Android Studio:** vantagens e desvantagens com relação ao Eclipse. 2013. Disponível em: <http://imasters.com.br/mobile/ android/android-studio-vantagens-e-desvantagens-com-relacao-ao-eclipse/?trace= 1519021197&source=single>. Acesso em: 25 mar. 2016.

CIDRAL, B. **Afinal, o que é Android?** 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2011/01/afinal-o-que-e-android.html>. Acesso em: 20 mar. 2016.

HOJI, M. **FINANÇAS DA FAMÍLIA**: O caminho para a Independência Financeira. Primeira. São Paulo: Pontifbooks, 2007.

LEITÃO, V. **Porque fazer controle de gastos pessoal?** 2015. Disponível em: <https://blog.mobills.com.br/2016/02/ por-que-fazer-controle-de-gastos-pessoal>. Acesso em: 18 mar. 2016.

MOBILLS. **Sobre a equipe Mobills.** [S.l.], 2014. Disponível em: <https://www.mobills.com.br/equipe>. Acesso em 23 mar. 2016.

MONTEIRO, J. B. **Google Android**. Primeira. [S.l.]: Casa do Código, 2013.

NAVARRO, C**. Orçamento Familiar além das Planilhas de Controle Financeiro**. 2014. Disponível em: <http://dinheirama.com/blog/2014/09/04/ orcamento-familiar-alem-planilhas-controle-financeiro>. Acesso em 19 mar. 2016.

NAVARRO, R. **Como realizar um Controle de Gastos Prático!** 2015. Disponível em: <http://www.coachfinanceiro.com/ planilha-para-controle-de-gastos/>. Acesso em: 27 mar. 2016.

PADILHA, J. **Gastos fantasmas:** um perigo para seu orçamento. 2013. Disponível em: <http://www.produzindo.net/ gastos-fantasmas-um-perito-paga-o-seu-orcamento/>. Acesso em: 19 mar. 2016.

PERETTI, L. C. **Educação Financeira:** aprenda a cuidar do seu dinheiro. Terceira. [S.l.]: Dois Vizinhos, 2008.

RABELLO, R. **Iniciando com Android:** introdução ao Android Studio. 2015.. Disponível em: <https://tasafo.wordpress.com/2015/01/19/ iniciando-com-android-introducao-ao-android-studio/>. Acesso em: 19 mar. 2016

RABELLO, R. R. **Android:** um novo paradigma de desenvolvimento móvel. 2013. Disponível em: < http://www.devmedia.com.br/artigo-webmobile-18-android-um-novo-paradigma-de-desenvolvimento-movel/9350 >. Acesso em: 19 mar. 2016.

SANTOS, A. C. **Web Services em aplicações Android e iOS.** 2015. Disponível em: <http://www.devmedia.com.br/ web-services-em-aplicacoes-android-e-ios/28901>. Acesso em: 23 mar. 2016.

SAUDATE, A. **REST:** construa API's Inteligentes de maneira simples. Primeira. [S.l.]: Casa do Código, 2014. ISBN 9788566250985.

TODOCELULAR. **Minhas Economias**. 2015. Disponível em: <http://www.tudocelular.com/ios/apps/n88/minhas-economias.html>. Acesso em: 29 mar. 2016.

TORRALVO, C. F. et al. **Planejamento financeiro pessoal e gestão do patrimônio**. São Paulo: Atlas, 2012.

VAZ, M. **Use o Mobills e tenha em mãos um gerenciador financeiro completo**. 2014. Disponível em: <http://www.techtudo.com.br/tudo-sobre/ mobills.html>. Acesso em: 19 mar. 2016.

APÊNDICE A – Descrição dos Casos de Uso

Quadro 9 - Caso de uso 001 - Logar no sistema

UC001 - Logar no sistema

Caminho básico
Caminho Básico:
[1] O usuário informa o login e a senha. [2] O usuário clica no botão de login. [3] O sistema exibe as informações do usuário. [4] Fim.

Exceções
Usuário ou senha incorretos.:
(at [3])
O sistema informa a mensagem "Senha não confere,".

Pré-condição *Situação:* Aprovado
Estar cadastrado no sistema.

Quadro 10 - Caso de uso 002 - Cadastrar novo usuário

UC002 - Cadastrar novo usuário

Caminho básico
Caminho Básico:
[1] O usuário clica no botão "cadastre-se". [2] O usuário informa o nome, sobrenome, e-mail, senha e repete a senha. [3] O usuário clica no botão "Salvar". [4] O sistema exibe a mensagem "Usuário incluído com sucesso,".

Exceções
Login já cadastrado.:
(at [4])
O sistema exibe a mensagem de "E-mail já cadastrado.".

Senha deve ter no mínimo 6 caracteres.:
(at [4])
O sistema exibe a mensagem "A senha deve ter no mínimo 6 caracteres"..

Quadro 11 - Caso de uso 004 - Cadastrar Carteira

UC003 - Cadastrar Carteira

Caminho Básico

Caminho Básico:

[1] O usuário clica na opção "+Conta". [2] O usuário informa o nome da conta e seleciona o tipo "Carteira". [3] O sistema grava as informações e exibe a mensagem "Registro incluido com sucesso.".

Pré-condição *Situação:* Aprovado

Estar logado no sistema.

Quadro 12 - Caso de uso 005 - Cadastrar cartão de crédito

UC004 - Cadastrar Cartão de Crédito

Caminho Básico

Caminho Básico:

[1] O usuário clica na opção "+Conta". [2] O usuário informa o nome da conta e o tipo "Cartão de crédito". [3] O sistema exibe a mensagem "Registro incluído com sucesso.".

Pré-condição *Situação:* Aprovado

Estar logado no sistema.

Quadro 13 - Caso de uso 006 - Cadastrar cartão de débito

UC005 - Cadastrar Cartão de Débito

Caminho Básico

Caminho básico:

[1] O usuário clica na opção "+Conta". [2] O usuário informa o nome da conta e o tipo "Cartão de débito". [3] O sistema exibe a mensagem "Registro incluído com sucesso".

Quadro 14 - Caso de uso 009 - Exibir gráfico de pizza por usuário

UC006 - Exibir gráfico de pizza por usuário

Caminho Básico

Caminho básico:

[1] O usuário seleciona o menu de "Gráfico por usuário". [2] O sistema exibe o gráfico de pizza dos lançamentos agrupados por usuário.

Caminhos Alternativos

Alternate 01:

(at [1])

O usuário clica em um dos usuários do gráfico. O sistema exibe os lançamentos daquele usuário, agrupados por categorias em forma de gráfico de pizza.

Quadro 15 - Caso de uso 010 - Compartilhar conta

UC007 - Compartilhar Conta

Caminho básico
Caminho básico:
[1] O usuário seleciona a conta. [2] O usuário clica na opção "Compartilhar". [3] O usuário informa o e-mail do destinatário. [4] O sistema exibe a mensagem de "Conta compartilhada com sucesso,".

Caminhos Alternativos
Usuário não encontrado:
(at [4])
O sistema exibe a mensagem "E-mail não encontrado".

Quadro 16 - Caso de uso 011 - Exibir movimentos

UC008 - Exibir movimentos

Caminho básico
Caminho básico:
[1] O usuário clica no menu "Movimentos". [2] O sistema exibe os movimentos do mês em order decrescente de data.

Pré-condição *Situação:* Aprovado
Está logado no sistema

Quadro 17 - Caso de uso 012 - Exibir gráfico de pizza por categoria

UC009 - Exibir gráfico de pizza por categoria

Caminho básico
Caminho básico:
[1] O usuário seleciona o menu "Gráfico por categoria". [2] O sistema exibe os lançamentos agrupados por categoria através de um gráfico de pizza.

Pré-condição *Situação:* Aprovado
Estar logado no sistema

Quadro 18 - Caso de uso 013 - Efetuar recebimento

UC010 - Efetuar Recebimento

Caminho básico
Caminho básico:
[1] O usuário seleciona a opção "+Rec". [2] O usuário informa o valor, a data, a categoria, a conta de destino e a descrição. [3] O sistema exibe a mensagem "Registro incluído com sucesso!".

Pré-condição *Situação:* Aprovado
Estar logado no sistema

Quadro 19 - Caso de uso 014 - Efetuar pagamentos

UC011 - Efetuar Pagamentos

Caminho básico
Caminho básico:
[1] O usuário seleciona a opção "+Pag". [2] O usuário informa o valor, a data, a categoria, a forma de pagamento e descrição do pagamento. [3] O sistema exibe a mensagem de "Registro incluído com sucesso!".

Pré-condição *Situação:* Aprovado
Estar logado no sistema

Quadro 20 - Caso de uso 015 - Efetuar transferências

UC012 - Efetuar transferências

Caminho básico
Caminho básico:
[1] O usuário seleciona a opção "+Transf". [2] O usuário informa o valor, a data, a conta de origem, a conta de destino e a descrição da transferência. [3] O sistema exibe a mensagem de "Registro incluído com sucesso!".

Pré-condição *Situação:* Aprovado
Estar logado no sistema

Quadro 21 - Caso de uso 016 - Efetuar saques

UC013 - Efetuar saques

Caminho básico
Caminho básico:
[1] O usuário seleciona a opção "+SAQUE". [2] O usuário informa o valor, a data, a conta de origem, a conta de destino e a descrição. [3] O sistema exibe a mensagem "Registro incluído com sucesso.".

Pré-condição *Situação:* Aprovado
Estar logado no sistema

Quadro 22 - Caso de uso 017 - Exibir resultado mensal

UC014 - Exibir resultado mensal

Caminho básico
Caminho básico:
[1] O usuário seleciona o menu "Resultado". [2] O sistema lista o resultado mensal que significa a soma de todas as receitas menos a soma de todas as receitas do mês.

Pré-condição *Situação:* Aprovado
Estar logado no sistema