UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUCA COUTO MANIQUE BARRETO

GibeMoni: Um App Android para Controle Financeiro Pessoal

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2015

AGRADECIMENTOS

Agradeço a meus pais e minhas mães, minha namorada e meus colegas de trabalho por todo o apoio e motivação que me deram. Não poderia chegar até aqui sem vocês. Agradeço também à Appus, em especial minha gestora Daniela Mota, por toda a compreensão e incentivo dados a mim para concluir o curso. Por fim, agradeço ao Professor Leandro Wives por se mostrar sempre solícito e me orientar ao longo do projeto, sendo fundamental para o desenvolvimento deste trabalho.

RESUMO

O objetivo deste estudo é desenvolver um aplicativo mobile de controle financeiro pessoal, com o intuito de facilitar a inserção de despesas, através da funcionalidade de leitura de nota fiscal eletrônica via QR-code, e a visualização de saldos. Apresentam-se as decisões tomadas no projeto do aplicativo, como modelagens criadas e arquitetura utilizada. A pesquisa traz ainda um guia de uso, com os principais fluxos de interação do sistema, complementados com *screenshots* da ferramenta. Também foi realizada uma avaliação de usabilidade do aplicativo com cinco usuários de diferentes perfis. Constatou-se que a ferramenta demonstrou possuir uma interface amigável, com avaliação bastante positiva entre os usuários.

Palavras-chave: Android. Controle Financeiro Pessoal. Código QR. Nota Fiscal Eletrônica.

GibeMoni: An Android App for Personal Financial Management

ABSTRACT

This survey aims to develop a mobile app for personal financial management in order to facilitate the control of expenses through the electronic invoice reading functionality via QR-code, including the view of the balances. Decisions made up during the elaboration of the project, as set models and architecture, are emphasized. The research also combines a tutorial with the main interaction flows of the system, supplemented by screenshots of the tool. An evaluation was realized with five users with different profiles to check the app's usability. The results pointed out a friendly interface leading to a positive evaluation among the users.

Keywords: Android. Personal Financial Management. QR Code. Electronic Invoice.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MVC	Model-View-Controller
URL	Uniform Resource Locator
ENAT	Encontro Nacional de Administradores Tributários
NFC-e	Nota Fiscal de Consumidor Eletrônica
QR	Quick Response
XML	Extensible Markup Language
CSV	Comma Separated Values
OFX	Open Financial Exchange
UX	User Experience

SUMÁRIO

1 INTRODUÇÃO

Uma das maiores dificuldades das pessoas na sociedade moderna é o controle de suas despesas. Segundo Cerbasi (2009), organizar a vida financeira é essencial para que a pessoa tenha maior controle sobre seu dinheiro, maior consciência sobre suas escolhas e maior eficiência no uso de sua renda.

De acordo com Galhardo (2008), controlar as despesas financeiras é entender e acompanhar as várias entradas e saídas em um fluxo financeiro.

Com o desenvolvimento da tecnologia e a popularização dos computadores pessoais, não tardou para que surgissem ferramentas desktop para auxiliar nesta tarefa. Alguns dos exemplos mais conhecidos e consolidados são Quicken[1], Microsoft Money[2] e GNU Cash[3] (Nagata, 2012).

A partir do surgimento de smartphones, notou-se que a mobilidade oferecida pelo dispositivo tornava a tarefa de inserir novos gastos muito mais fluida, permitindo registrar uma despesa no instante em que ela ocorresse e verificar os saldos das contas a qualquer momento.

1.1 Motivação

Entretanto, percebe-se que os aplicativos de forma geral não são capazes de reter os usuários e proporcionar educação financeira. A rotina de controle e inserção de gastos é muitas vezes trabalhosa e burocrática, e as formas de visualização dos gastos costumam ser rasas ou não amigáveis, desestimulando os usuários.

Diante deste quadro, apresenta-se como problema de pesquisa: como fornecer uma ferramenta que facilite a inserção de operações financeiras com uma forma amigável de apresentar os saldos de maneira organizada e intuitiva?

---

[1]QUICKEN. Disponível em: <http://www.quicken.com/>. Acesso em: 30 de nov. de 2015.

[2]MONEY, Microsoft. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=20738>. Acesso em: 30 de nov. de 2015.

[3]CASH, GNU. Disponível em: <http://www.gnucash.org/>. Acesso em: 30 de nov. de 2015.

1.2 Objetivo

Este estudo tem como objetivo desenvolver um aplicativo de controle financeiro pessoal focado em oferecer uma interface amigável através dos recursos de leitura via QR-code de Nota Fiscal ao Consumidor Eletrônica e visualização intuitiva de gastos e saldos.

1.3 Estrutura do Texto

Os próximos capítulos são organizados da seguinte maneira: o Capítulo 2 aborda a fundamentação teórica, trazendo trabalhos relacionados e aplicativos similares e apresentando a nota fiscal eletrônica. O Capítulo 3 aprofunda as decisões tomadas no projeto do aplicativo, como modelagens criadas e arquitetura utilizada. Na sequência, o Capítulo 4 traz um guia de uso, com os principais fluxos de interação do sistema, complementados com *screenshots* da ferramenta. O Capítulo 5 apresenta uma avaliação da aplicação através de usuários com diferentes perfis. Por fim, no Capítulo 6 é feita uma análise sobre os resultados obtidos na avaliação e como eles se relacionam com o objetivo proposto pelo estudo. Nele ainda são apresentadas melhorias e novas funcionalidades que poderão ser realizadas em trabalhos futuros.

2 ESTADO DA ARTE E FUNDAMENTAÇÃO TEÓRICA

Este capítulo traz os estudos realizados no início do desenvolvimento deste trabalho, que contribuíram para a definição do escopo das funcionalidades e dos requisitos do aplicativo proposto.

Primeiramente são apresentados os trabalhos relacionados, realizados em cima de aplicações Android e que influenciaram nas decisões de modelagem tomadas. Em seguida, são listados aplicativos similares de finanças pessoais, que auxiliaram na tarefa de elencar as funcionalidades do produto. Por fim é descrita a nota fiscal eletrônica e sua implantação no Rio Grande do Sul.

2.1 Trabalhos Relacionados

Antes de iniciar a implementação do aplicativo foram procuradas monografias da área da computação que envolvessem o desenvolvimento Android. Dentre as encontradas, algumas se destacaram por apresentarem conceitos de arquitetura Android — em especial fazendo comparações com a arquitetura MVC — ou detalharem funcionalidades como leitura de QR-code, sendo de grande utilidade para a concepção e o desenvolvimento do *app* GibeMoni. Estes trabalhos, que serviram como base desta pesquisa, são apresentados a seguir.

2.1.1. SACI - Sistema de Apoio à Coleta de Informações

Em SACI, Souza (2014) propõe um sistema para coleta de dados em auditorias de pontos de venda composto por um aplicativo Android, uma aplicação Web e um servidor RESTful. Nele, um administrador configura formulários de pesquisas — para medir, por exemplo, a qualidade de atendimento — através do *webapp* enquanto os entrevistadores utilizam dispositivos Android para visualizar informações sobre os pontos de venda e aplicar as pesquisas. O trabalho ainda apresenta uma arquitetura baseada em MVC e detalha conceitos importantes sobre o desenvolvimento em Android.

2.1.2. Check in Poa: Um aplicativo Android para turistas em Porto Alegre

O trabalho foca no desenvolvimento de um aplicativo para turismo em Porto Alegre sob a perspectiva de um jogo. Macalão (2013) apresenta uma arquitetura baseada em MVC, traçando uma comparação entre o modelo clássico e os componentes principais da plataforma Android. Este comparativo serviu de inspiração para os questionamentos levantados ao longo do desenvolvimento deste trabalho, que serão apresentados na próxima seção.

2.1.3. Desenvolvimento de um Protótipo de Solução Colaborativa para o Controle de Listas de Compras

Abbeg (2014) apresenta Batatas, um protótipo de aplicativo Android para o gerenciamento compartilhado de listas de compras. O trabalho é composto por um servidor RESTful e um cliente Android e segue boas práticas de desenvolvimento.

2.1.4. Suporte a Notas Fiscais Eletrônicas e Integração com Facebook em Aplicativo Android para Gerenciamento de Listas de Compras Colaborativas

Uma das principais referências deste estudo, o trabalho de Figueiredo (2015) estende a proposta de Abbeg (2014), inserindo novas funcionalidades no aplicativo Batatas. Além de desenvolver o controle de usuários e integração com o Facebook, o autor implementa a leitura de código de barras e de nota fiscal eletrônica para facilitar a criação de listas de compras.

Os componentes de leitura de QR-code e extração de dados do portal da Secretaria da Fazenda, reaproveitados em *GibeMoni*, foram fundamentais para atingir o objetivo deste trabalho.

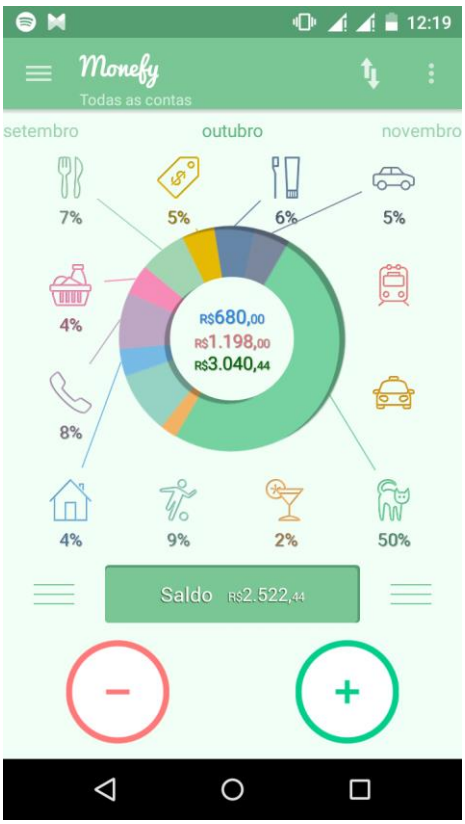
2.2. Aplicativos Similares

Após estabelecer os objetivos deste trabalho, pesquisou-se na *PlayStore* — a loja de aplicativos do Google — os apps de controle financeiro pessoal mais populares. Esta pesquisa permitiu um levantamento mais profundo das funcionalidades e requisitos do sistema a ser desenvolvido. A seguir são descritos os aplicativos que apresentaram as características mais marcantes e que tiveram o maior impacto na concepção da ferramenta proposta.

2.2.1. Monefy

O Monefy[4] é um aplicativo de finanças pessoais focado na simplicidade e em apresentar uma interface amigável. O aplicativo opera na sua versão Lite (gratuita), onde permite as ações básicas de controle financeiro — como inserção manual, categorização e visualização de saldo por períodos — e Pro — que pode ser habilitada através de pagamento ou permitindo publicidades —, com funcionalidades adicionais para gerência de categorias, senha de segurança e sincronização dos dados através do Dropbox. A Figura 2.1 apresenta a interface do app.

Figura 2.1 – Tela principal do app Monefy Lite



Fonte: Arquivo do autor.

[4] GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.monefy.app.lite>. Acesso em: 26 de ago. de 2015.

2.2.2. Minhas Economias

O aplicativo Minhas Economias[5], por sua vez, foca em oferecer um conjunto maior de funcionalidades. O app permite aos usuários lançarem operações futuras, agendar despesas e receitas cíclicas (como salário e conta de luz) e apresenta uma quantidade enorme de categorias e subcategorias. O software ainda exibe notícias relacionadas com finanças e possibilita a visualização de extratos em mais opções de gráficos para ajudar no controle financeiro. A interface do produto é apresentada na Figura 2.2.

Figura 2.2 – Tela principal do aplicativo Minhas Economias

Fonte: Arquivo do autor.

[5]GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.minhaseconomias>. Acesso em 26 de ago. de 2015.

2.2.3. Money Manager EX

Money Manager EX[6] é um software livre disponibilizado em diversas plataformas, com versões tanto para desktop quanto para Android. O aplicativo foca na customização, permitindo o gerenciamento total de contas, categorias, beneficiários e até mesmo relatórios e gráficos. Além disso, por se tratar de um projeto *open-source*, a colaboração da comunidade permite ao sistema oferecer uma vasta variedade de idiomas e moedas. Porém, por permitir um alto grau de customização, a ferramenta peca na usabilidade, tendo formulários confusos para a inserção manual e mesmo para a gerência de contas. As Figuras 2.3 e 2.4 mostram, respectivamente, a tela principal do aplicativo e o formulário de inserção manual.

Figura 2.3 – Tela principal do app Money Manager Ex

Fonte: Arquivo do autor.

Figura 2.4 – Tela de Inserção Manual do app Money Manager Ex

---

[6] GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.money.manager.ex>. Acesso em: 26 de ago. de 2015.

Fonte: Arquivo do autor.

2.2.4. GuiaBolso

Com mais de 1.000.000 de downloads, o GuiaBolso é o app mais baixado da lista de aplicativos de finanças pessoais estudada[7]. O software possui um conjunto de recursos que permite ao usuário automatizar quase todo o processo de inserção de operações financeiras, sincronizando dados de contas bancárias e cartões de crédito e inferindo categorias às transações extraídas.

O programa ainda incorpora conceitos de *Gamification*, criando uma pontuação para os usuários para estimulá-los a manter hábitos financeiros saudáveis. Entretanto, por focar tanto na automatização, os recursos manuais do aplicativo são bastante limitados. É trabalhoso criar uma conta não vinculada a banco ou mesmo inserir um gasto novo

[7] GOOGLE. **Play Store**. <https://play.google.com/store/apps/details?id=br.com.guiabolso>. Acesso em: 26 de ago. de 2015.

manualmente, além disso, ambas as tarefas exigem conexão com a internet. A seguir, as Figuras 2.5 e 2.6 apresentam a longa tela principal e o conceito de *gamification* aplicado pelo app respectivamente.

Figura 2.5 – Tela principal do Guia Bolso

Fonte: Arquivo do autor.

Figura 2.6 – Demonstração de *Gamification* no app Guia Bolso

Fonte: GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=br.com.guiabolso>. Acesso em: 26 de ago. de 2015.

2.2.5. Tabela Comparativa

Após concluir a listagem de aplicativos estudados foram elencadas características e funcionalidades relevantes para a realização do objetivo proposto. Em cima disso foi montada a Tabela 2.1, que faz uma análise comparativa entre as ferramentas selecionadas.

Tabela 2.1 – Análise comparativa dos Aplicativos Similares

	Monefy	**Minhas Economias**	**Money Manager EX**	**Guia Bolso**
Acesso	Offline (Online na versão PRO)	Offline e Online	Offline e Online	Online
Plataformas Adicionais	Nenhuma	Web	Desktop	Web
Modos de Inserção	Manual (Importação na versão PRO)	Manual	Manual e Importação (.OFX)	Manual, Sincronização de Contas
Internacionalização	Idiomas e moedas	Não	Idiomas e moedas	Moedas (real e dólar)
Foco	Simplicidade, "*Friendliness*"	Funcionalidades	Customização, Crossplatform, Open Source	Automatização, *Gamification*
Downloads	500.000 a 1.000.000	100.000 a 500.000	50.000 a 100.000	1.000.000 a 5.000.000

Fonte: Produzido pelo Autor.

2.3. Nota Fiscal Eletrônica

A implantação da Nota Fiscal Eletrônica no Brasil foi realizada a partir da assinatura do protocolo ENAT 03/2005, pelos secretários de Fazenda dos Estados e Distrito Federal, o secretário da Receita Federal e os representantes das Secretarias de Finanças dos municípios das Capitais.

O objetivo do projeto foi a implantação de um modelo nacional de documento fiscal eletrônico para substituir a sistemática de emissão do documento fiscal em papel, com validade jurídica garantida pela assinatura digital do remetente. Uma das vantagens da NFe é permitir o acompanhamento em tempo real das operações comerciais pelo Fisco (FAZENDA, 2015).

No Rio Grande do Sul, o Projeto está em operação desde setembro de 2006, com as empresas credenciadas como emissoras emitindo normalmente a Nota Fiscal Eletrônica em substituição à tradicional Nota Fiscal em papel, modelo 1 e 1A. (SEFAZ, 2015)

Já a Nota Fiscal de Consumidor Eletrônica (NFC-e), segundo o Ministério da Fazenda (2015), foi instituída pelo Ajuste Sinief I (Sistema Nacional de Informações Econômicas e Fiscais) nº 01/2013, que alterou o Ajuste Sinief n º 07/2005 (Nota Fiscal Eletrônica - NF-e).

O Código QR (Quick Response), ou QR-code, é um tipo de código de barras bidimensional desenvolvido pela Denso Wave em 1994, no Japão. Em 2000 a marca foi aprovada como padrão ISO[8]. Existem diversos tipos de códigos, com capacidade de armazenamento variando entre 35 (Micro QR) e 40.000 (iQR) numerais, com o tipo mais utilizado, o Model2, podendo armazenar cerca de 7000 dígitos numéricos[9].

---

[8] WAVE, Denso. **History of QR Code**. Disponível em: <http://www.qrcode.com/en/history/> Acesso em: três de dez. de 2015.

[9] WAVE, Denso. **Types of QR Code**. Disponível em: <http://www.qrcode.com/en/codes/> Acesso em: três de dez. de 2015.

3 PROJETO E IMPLEMENTAÇÃO DO APLICATIVO

Neste capítulo serão detalhadas as fases de projeto e implementação do trabalho, abordando a modelagem e arquitetura usadas na concepção do aplicativo e descrevendo o seu processo de desenvolvimento.

3.1 Modelagem

Esta seção apresenta as modelagens realizadas para o desenvolvimento do projeto. As funcionalidades foram modeladas com o uso de *user stories*, enquanto o modelos de dados foi especificado através de um diagrama relacional.

3.1.1. User Stories

User Stories são, segundo Cohn (2004), artefatos de desenvolvimento ágil que descrevem funcionalidades valiosas aos usuários ou clientes do sistema. Neste projeto, o uso deste artefato baseou-se no *template* "Como <papel> quero <ação/funcionalidade> para <objetivo>" (COHN, 2004). As histórias criadas são apresentadas a seguir.

3.1.1.1. Inserção Manual de Operação

Como usuário quero inserir manualmente uma Operação financeira para contabilizar gastos e receitas.

3.1.1.2. Inserção de Operações através de Leitura de QR-Code

Como usuário quero inserir Operações financeiras através da leitura de um QR-code para contabilizar gastos e receitas de uma forma mais eficiente.

3.1.1.3. Contas Default

Como usuário quero ter contas financeiras pré-criadas pelo sistema para poder focar nas operações sem perder tempo.

3.1.1.4. Categorias Default

Como usuário quero ter Categorias pré-criadas pelo sistema para poder focar nas operações sem perder tempo.

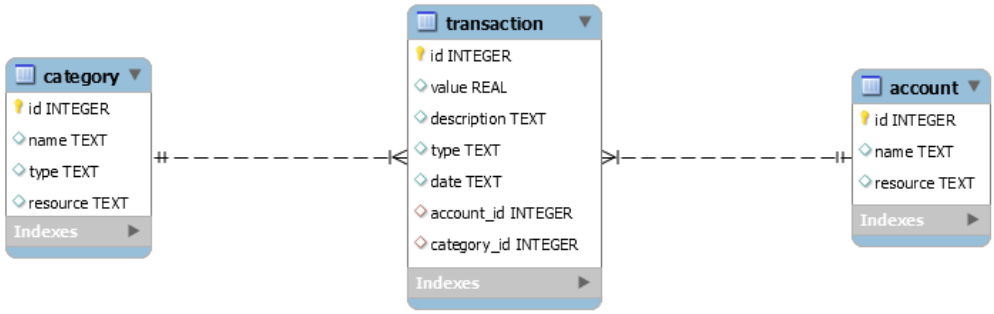
3.1.1.5. Saldo atual

Como usuário quero verificar o saldo atual de minhas contas para ter controle sobre minhas despesas.

3.1.2. Entidades Relacionais

A modelagem de dados, ou de entidades relacionais, permite demonstrar como serão construídas as estruturas de dados que darão suporte aos processos de negócio. Além disso, possibilita definir como esses dados estarão organizados e quais os relacionamentos que se pretende estabelecer entre eles (DEBASTIANI, 2015). A Figura 3.1 apresenta as entidades modeladas, que são detalhadas a seguir.

Figura 3.1 – Modelo de Entidades Relacionais



Fonte: Produzido pelo autor.

3.1.2.1. Conta Financeira (account)

Esta entidade descreve uma conta financeira, como “Conta Corrente”, “Poupança”, “Carteira”, entre outros.

id: Identificador único da conta.

name: Nome da conta.

resource: Sufixo utilizado para extrair os recursos (ícones, cores, textos internacionalizáveis) relativos à conta. A extração de recursos será detalhada na próxima seção.

3.1.2.2. Categoria (category)

Entidade que define uma categoria de operação financeira. As categorias são divididas entre os tipos *receita* e *débito*. São exemplos de categorias "Alimentação", "Transporte", "Lazer" (todas do tipo *débito*) e "Salário" (do tipo *receita*).

id: Identificador único da categoria.

name: Nome da categoria.

type: Tipo da categoria. É modelado como uma enumeração no código e convertido para texto no banco.

resource: Sufixo utilizado para extrair os recursos relativos à categoria.

3.1.2.3. Operação Financeira (transaction)

A principal entidade do projeto. Descreve uma operação financeira, como um almoço em um restaurante pago com dinheiro, a taxa de inscrição de um curso de especialização paga com cartão de crédito, ou mesmo o rendimento mensal de uma poupança. As operações também são divididas entre os tipos *receita* e *débito*, além de serem organizadas através de sua categoria e conta associada.

id: Identificador único da operação.

value: Valor monetário da operação.

description: Descrição opcional da operação.

type: Tipo da operação, também modelado como uma enumeração no código e convertido para texto no banco.

date: Data da operação. Modelada no banco como texto no padrão ISO 8601, isto é, ano-mês-dia, como 2015-09-23 e 2011-04-16.

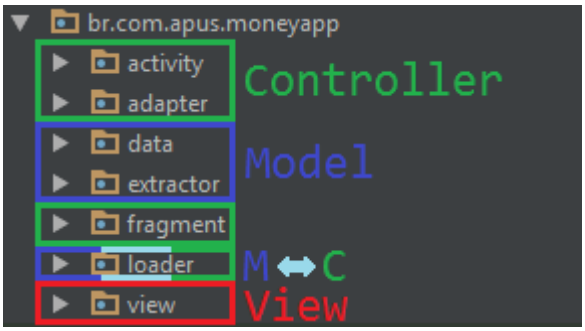
account_id: Identificador da conta atribuída à operação.

category_id: Identificador da categoria atribuída à operação.

3.2. Arquitetura

Nesta seção é apresentada a arquitetura do sistema. Desde a concepção do projeto procurou-se estudar formas de implementar o aplicativo seguindo os padrões *Model-View-Controller* (MVC) propostos por Burbeck (1992). A Figura 3.2 apresenta os pacotes do projeto, indicando seu papel dentro da Arquitetura MVC. Cada pacote será detalhado a seguir.

Figura 3.2 – Pacotes do projeto e seus respectivos papeis na Arquitetura MVC



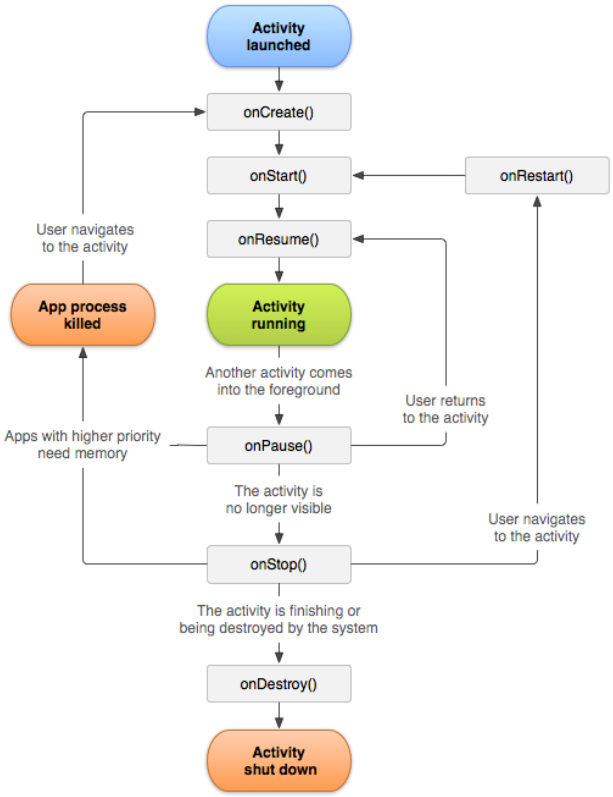
Fonte: Produzido pelo autor.

3.2.1. Activities

As *Activities* são os principais componentes da Arquitetura Android. Cada atividade é responsável por uma janela do aplicativo, controlando todas as ações de usuário nela e gerenciando os dados necessários. Além disso, estes componentes gerenciam seu ciclo de vida, apresentado na Figura 3.3, que é afetado por sua associação com outras atividades e pela execução de suas tarefas[10].

Figura 3.3 – Ciclo de Vida de uma *Activity*

[10] GOOGLE. **Activities | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/components/activities.html>. Acesso em: três de dez. de 2015.

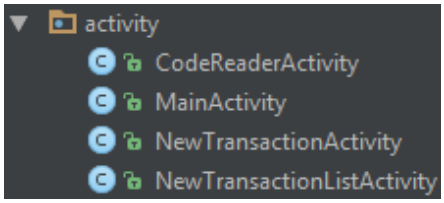


Fonte: http://developer.android.com/intl/pt-br/guide/components/activities.html

Dentro da Arquitetura MVC a *Activity* ocupa o papel de *Controller*, solicitando os dados necessários aos *models* e alterando-os através do controle das ações do usuário, feitas através de uma *view* especificada por um Arquivo de Layout — um dos próximos componentes a ser detalhado.

As atividades do app são apresentadas na Figura 3.4. *MainActivity* controla a tela de Visão Geral, *CodeReaderActivity* é o componente responsável pela janela de captura de QR-code, enquanto *NewTransactionActivity* e *NewTransactionListActivity* cuidam, respectivamente, da inserção manual e da inserção via NFC-e.

Figura 3.4 – Atividades implementadas

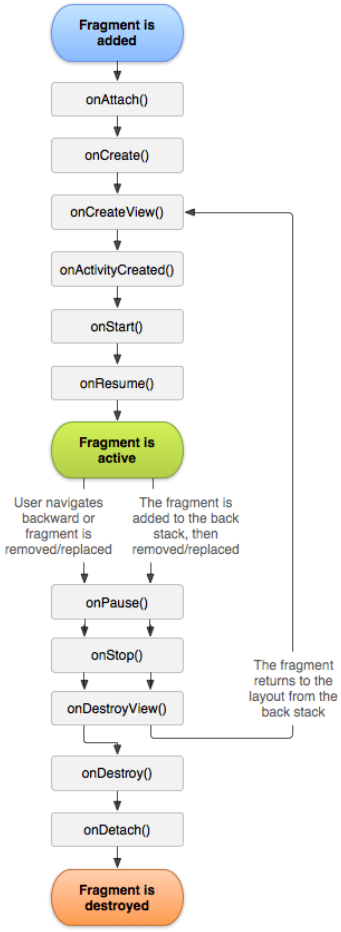


Fonte: Produzido pelo autor.

3.2.2. Fragments

Fragments, como o nome sugere, são componentes que encapsulam partes reaproveitáveis de *Activities*, definindo seu próprio layout e possuindo um ciclo de vida próprio — exibido na Figura 3.5 —, que fica diretamente atrelado ao ciclo de vida de uma atividade quando esta o anexa[11]. Assim como as atividades, estes componentes atuam na Arquitetura MVC como *Controllers*.

Figura 3.5 – Ciclo de Vida de um *Fragment*

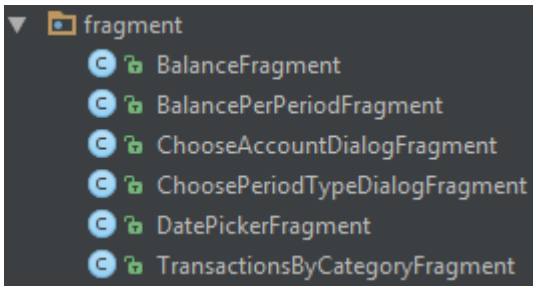


Fonte: http://developer.android.com/intl/pt-br/guide/components/fragments.html

[11] GOOGLE. **Fragments | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/components/fragments.html>. Acesso em: três de dez. de 2015.

Um fragmento pode expor seu estado a uma atividade através de uma interface que implementa o *design pattern Observer*, onde um objeto notifica automaticamente um conjunto de elementos dependentes (observadores) sempre que seu estado é alterado (GAMMA ET AL., 2004). Assim, no momento da anexação do fragmento à atividade, o primeiro define a última como sua observadora, exigindo que ela implemente os métodos da interface que tratam o envio de notificações. A Figura 3.6 apresenta os fragmentos implementados no projeto.

Figura 3.6 – Fragmentos implementados



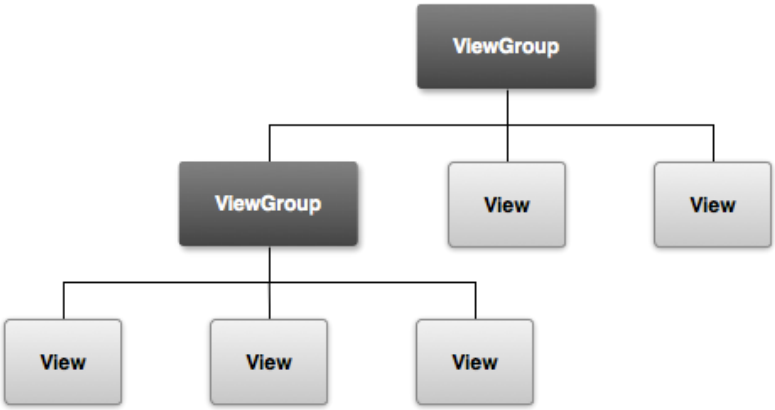
Fonte: Produzido pelo autor.

3.2.3. Views e Layouts

A Arquitetura Android simplifica bastante a criação de *views* por permitir a especificação através de arquivos XML. O layout de uma atividade, fragmento ou componente é definido sob uma estrutura hierárquica de objetos das classes *View* e *ViewGroup*, como mostrado na Figura 3.7[12].

Figura 3.7 – Estrutura hierárquica de *Views* e *ViewGroups*

[12] GOOGLE. **UI Overview | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/ui/overview.html>. Acesso em: três de dez. de 2015.



Fonte: http://developer.android.com/intl/pt-br/guide/topics/ui/overview.html.

Views definem componentes gráficos atômicos, aparecendo na árvore hierárquica do layout sempre como folhas. São exemplos de *Views* caixas de texto, imagens, botões e switches. *GroupViews* por sua vez são *containers* de Views e aparecem sempre como ramos no layout. Containers lineares, de grid e relativos são exemplos disso[13].

Um caso especial de *GroupViews* que vale ser destacado pela importância neste trabalho são as *AdapterViews*, que não contêm *Views* como elementos filhos, mas sim *Adapters*. Estes componentes lidam com dados mutáveis, gerando Views dinamicamente, e serão detalhados adiante. Exemplos de *AdapterViews* incluem listas, spinners e matrizes[14].

A Figura 3.7 apresenta todos os componentes de layout criados neste trabalho. Um recurso interessante do Android que pode ser visualizado na imagem é a possibilidade de criar layouts específicos conforme a orientação — paisagem ou retrato — do dispositivo, os quais são aplicados automaticamente pelo sistema, como é o caso de *activity_main.xml*, o layout da tela de Visão Geral.

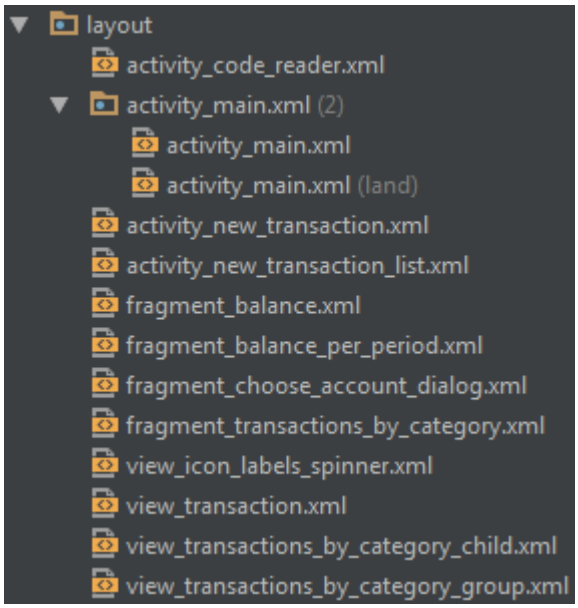
---

[13] GOOGLE. **UI Overview | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/ui/overview.html>. Acesso em: três de dez. de 2015.

[14] GOOGLE. **Layouts | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/ui/declaring-layout.html#AdapterViews> . Acesso em três de dez. de 2015.

GOOGLE. **AdapterView | Android Developers**. Disponível em http://developer.android.com/intl/pt-br/reference/android/widget/AdapterView.html>. Acesso em: três de dez. de 2015.

Figura 3.7 – Layouts implementados



Fonte: Produzido pelo autor.

3.2.4. Adapters

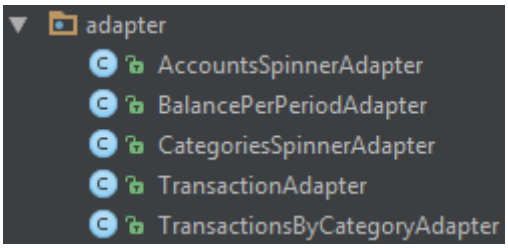
Adapters funcionam como uma ponte entre uma *AdapterView* e o conjunto de dados subjacentes a ela. Estes componentes atuam como "minicontroladores", monitorando o conjunto e criando *Views* — e gerenciando os seus eventos — para cada um de seus itens sempre que solicitado através de alguma ação do usuário[15].

A Figura 3.8 apresenta os *Adapters* desenvolvidos no projeto. *TransactionByCategoryAdapter* é um adaptador de listas expansíveis, responsável pela listagem de categorias na tela de Visão Geral, onde cada item pode ser expandido para exibir suas operações registradas. Outro componente importante deste pacote — também utilizado na tela principal do aplicativo — é o *BalancePerPeriodAdapter*, que se encarrega de, a partir da periodicidade escolhida pelo usuário, agregar as operações nos períodos existentes e calcular a receita total e o saldo das contas selecionadas.

Figura 3.8 – Adapters implementados

---

[15] GOOGLE. **Adapters | Android Developers**. Disponível em < http://developer.android.com/intl/pt-br/reference/android/widget/Adapter.html>. Acesso em três de nov. de 2015.

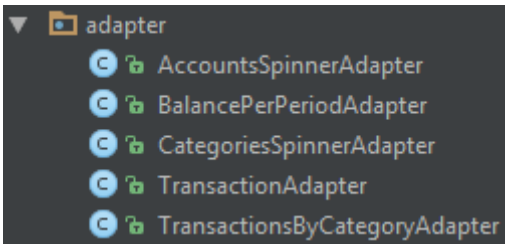


Fonte: Produzido pelo autor.

3.2.5. Data

O pacote *Data* define as classes apresentadas na Figura 3.9, que lidam com o banco de dados e com a lógica dos modelos de dados, exercendo o papel da camada de *Model* no MVC. O Android oferece suporte ao SQLite através de classes como *SQLiteOpenHelper* — sobrecarregada aqui por *MoneyDbHelper* —, que definem métodos para executar as principais instruções de bancos de dados e deve ser sobrecarregada para implementar os comandos de criação e atualização do banco[16]. Também faz parte do padrão de desenvolvimento Android a criação de uma classe *Contract* (neste caso, *MoneyContract*) que especifica o *schema* do projeto inteiro[17].

Figura 3.9 – Datas implementados



Fonte: Produzido pelo autor.

---

[16] GOOGLE. **Storage Options | Android Developers**. Disponível em : <http://developer.android.com/intl/pt-br/guide/topics/data/data-storage.html#db>. Acesso em: três de dez. de 2015.

[17] GOOGLE. **Salvando dados em bancos de dados do SQL | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/training/basics/data-storage/databases.html>.

Ainda neste pacote foi criada a classe abstrata *DataObject*, que define um comportamento baseado no *pattern Active Record*, descrito por Fowler (2002), onde um objeto encapsula uma linha de uma tabela, os comportamentos de acesso ao banco e a lógica de negócio do dado. Dessa forma, cada objeto das classes concretas *Account*, *Category* e *Transaction* detém o conhecimento sobre como se salvar no banco, seja criando uma nova entrada ou atualizando uma já existente, ou se excluir.

A classe *DataObject* define ainda uma interface chamada *Querier*, que apresenta o método *query* para executar uma busca de dados no banco. As implementações desta interface podem ser customizadas com atributos que alteram o comportamento da carga — por exemplo, solicitando apenas operações que ocorreram durante o mês de outubro —, criando uma maneira única mas bastante flexível de se realizar buscas. A Figura 3.10 demonstra um exemplo de uso da implementação *CategoryQuerier*.

Figura 3.10 – Exemplos de uso de um Querier

```
Category.CategoryQuerier querier = new Category.CategoryQuerier(context);

List<Category> allCategories = querier.query();
List<Category> debtCategories = querier.setType(Transaction.Type.DEBT).query();
List<Category> incomeCategories = querier.setType(Transaction.Type.INCOME).query();
```

Fonte: Produzido pelo autor.

3.2.6. Loaders

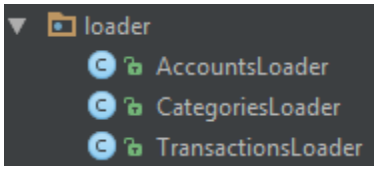
Os Loaders são componentes do Android introduzidos na versão 3.0. Sua função é a de facilitar o gerenciamento da carga de dados nas atividades e fragmentos, permitindo carga assíncrona e o monitoramento da fonte dos dados — retornando valores atualizados em caso de alterações no conteúdo — através de uma interface simplificada[18].

Neste projeto foi implementado um Loader para cada entidade do modelo de dados, como pode ser visto na Figura 3.11. Após implementar uma classe loader, para utilizá-la em uma atividade ou fragmento é necessário que esta última implemente a interface

[18] GOOGLE. **Loaders | Android Developers**. Disponível em: < http://developer.android.com/intl/pt-br/guide/components/loaders.html>. Acesso em: três de dez. de 2015.

LoaderManager.LoaderCallbacks, que define métodos para lidar com a criação e reinicialização de loaders e o comportamento desejado ao concluir um carregamento.

Figura 3.11 – Carregadores implementados



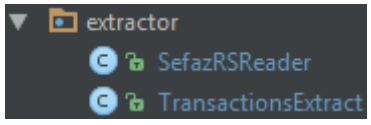
Fonte: Produzido pelo autor.

Dentro de uma arquitetura MVC, os loaders desempenham um papel de comunicação entre a camada de Modelos e Controllers, buscando dados sempre que solicitado pelas Activities e Fragments e atualizando-os conforme ocorrem mudanças na base.

3.2.7. Extractors

As classes desse pacote, vistas na Figura 3.12, foram criadas especificamente para realizar a tarefa de extrair as operações financeiras de uma nota fiscal eletrônica. *TransactionsExtract*, apresentada abaixo na Figura 3.13, representa as informações que devem ser extraídas de uma NFC-e. Os objetos desta classe são construídos recebendo um extrator — uma implementação da interface *Extractor*, que define métodos para se obter cada um dos atributos de TransactionsExtract — e uma fonte de informação (por exemplo, a URL da nota).

Figura 3.12 – Extratores implementados



Fonte: Produzido pelo autor.

A interface *Extractor* foi construída baseando-se no design pattern *Strategy*, onde, segundo Gamma et al (1994), uma classe pode apresentar diversos comportamentos e a escolha de qual deles utilizar precisa ser feita dinamicamente em tempo de execução. Dessa forma, é possível realizar a implementação de diversas estratégias de extração conforme as especificações de cada órgão regulador. A escolha da abordagem, por sua vez, pode ser feita

com base na fonte da informação, analisando, por exemplo, o domínio da URL obtida no QR-code. Para o escopo deste trabalho, implementou-se apenas um extrator, *SefazRSReader*, para as NFC-e regulamentadas pela Secretaria da Fazenda do Rio Grande do Sul.

Figura 3.13 – Trecho de código da classe *TransactionsExtractor*

```
13  public class TransactionsExtract {
14      private final String issuer;
15      private final Date issuingDate;
16      private final String accessKey;
17      private final List<Product> products;
18      private final List<PaymentMethod> paymentMethods;
19
20      public TransactionsExtract(Extractor callbacks, String source) throws IOException, ParseException {
21          callbacks.init(source);
22          issuer = callbacks.extractIssuer();
23          issuingDate = callbacks.extractIssuingDate();
24          accessKey = callbacks.extractAccessKey();
25          products = callbacks.extractProducts();
26          paymentMethods = callbacks.extractPaymentMethods();
27      }
28
29      public List<Transaction> createTransactionsByProducts() {...}
41
42      public List<Transaction> createTransactionsByPaymentMethods() {...}
54
55      protected static class Product {...}
66
67      protected static class PaymentMethod {...}
76
77      public interface Extractor {
78          void init(String source) throws IOException;
79          String extractIssuer();
80          Date extractIssuingDate() throws ParseException;
81          String extractAccessKey();
82          List<Product> extractProducts() throws ParseException;
83          List<PaymentMethod> extractPaymentMethods() throws ParseException;
84      }
85  }
```

Fonte: Produzido pelo autor.

3.3 Processo de Desenvolvimento

O desenvolvimento do aplicativo foi dividido em ciclos de 15 dias, chamados *sprints*, baseando-se em metodologias ágeis como o *Scrum*[19].

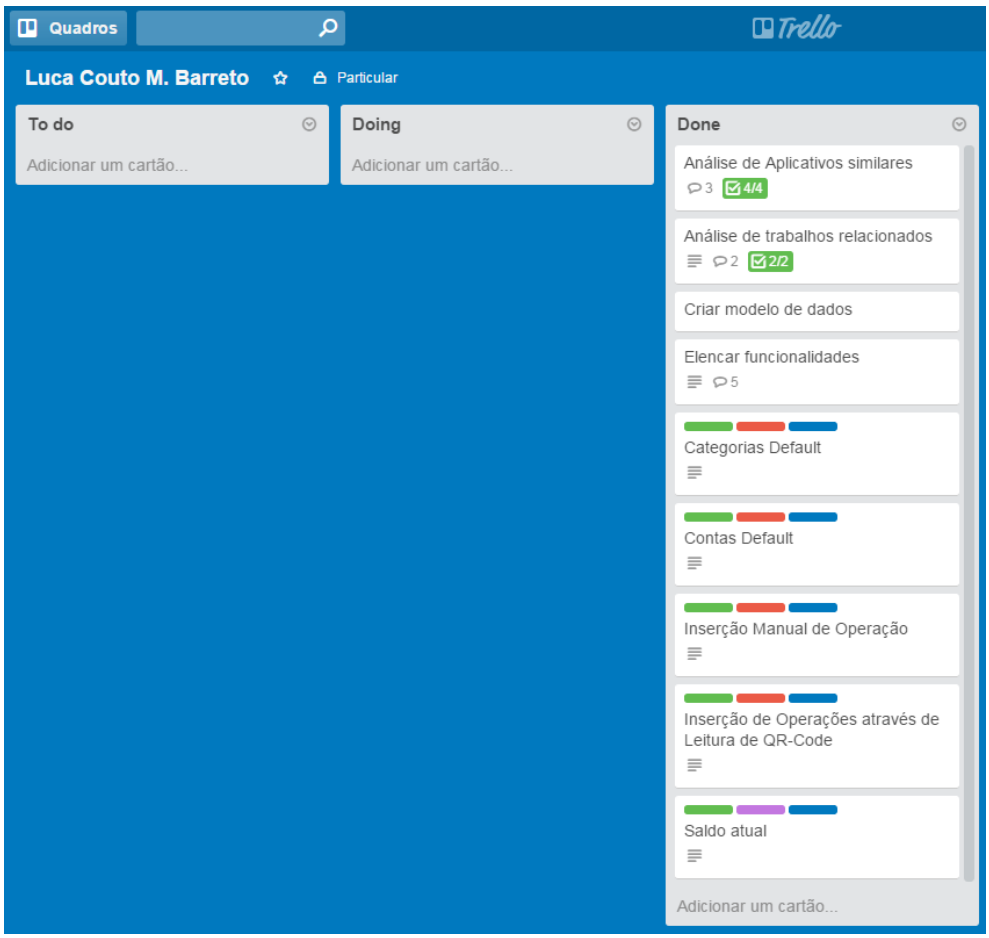
O primeiro ciclo, único a não envolver codificação, serviu para analisar aplicativos similares e trabalhos relacionados – descritos no capítulo anterior –, elencar as funcionalidades e fazer a modelagem de dados – cujos resultados foram apresentados na seção 3.1.

[19] SCRUM Alliance. Disponível em: <https://www.scrumalliance.org/>. Acesso em 10 de dez. de 2015.

O segundo ciclo consistiu em implementar a camada de dados (subseção 3.2.5) e realizar as tarefas de criação de contas e categorias default (respectivamente itens 3.1.1.3 e 3.1.1.4).

No terceiro ciclo foi concluída a tarefa de inserção manual (3.1.1.1) e realizada a primeira parte da tela de visão geral (3.1.1.5), com a navegação de saldos. O quarto e último ciclo fechou a etapa de desenvolvimento com a implementação da inserção via NFC-e (3.1.1.2) e conclusão da tela de visão geral com as configurações de navegação (periodicidade e seleção de contas). A Figura 3.14 apresenta o *kanban* utilizado para permitir o acompanhamento das tarefas após a finalização do processo.

Figura 3.14 – Kanban com as tarefas de desenvolvimento



Fonte: Produzido pelo autor.

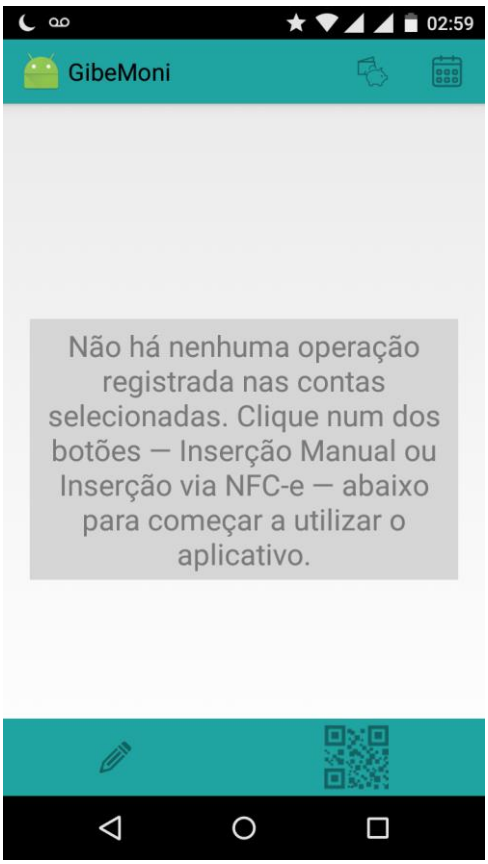
4 GUIA DE USO

Esta seção demonstra o funcionamento do aplicativo através de suas telas. São apresentados os principais fluxos de uso da ferramenta, desde o primeiro acesso até a visualização dos gastos discriminados por categorias e contas financeiras ao longo do tempo, incluindo o registro manual de operações e a extração automática via *QR-code*.

4.1. Primeiros passos

Ao acessar o aplicativo pela primeira vez, o usuário ainda não possui nenhum dado registrado, portanto a tela inicial, mostrada na Figura 4.1, indica a ele onde ir para inserir suas primeiras operações.

Figura 4.1 – Tela Inicial



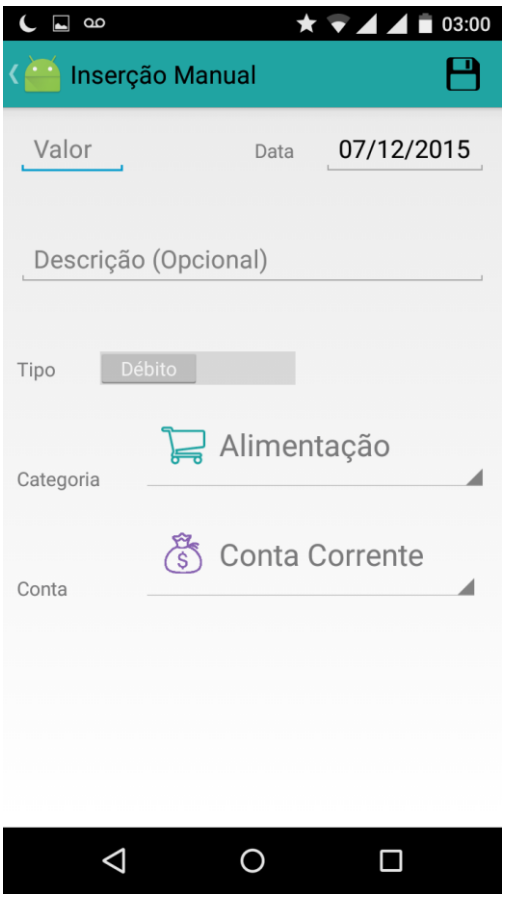
Fonte: Produzido pelo autor.

4.2. Inserção Manual

Caso opte por inserir uma operação manualmente, o usuário é apresentado a um formulário, onde define o tipo da transação (*despesa* ou *receita*), o seu valor e sua data — inicialmente apresentada com a data atual —, tendo a possibilidade de adicionar uma descrição. Por fim, deve-se atribuir uma conta financeira e escolher a qual categoria pertence a operação. O formulário de inserção manual é apresentado na Figura 4.2.

O aplicativo não permite, no momento, a criação ou edição das contas e categorias, mas apresenta ao usuário os principais tipos de contas (*Conta Corrente*, *Carteira*, *Poupança* e *Cartão de Crédito*) e categorias bastante abrangentes (*Alimentação*, *Casa*, *Educação*, *Lazer*, *Gastos Pessoais*, *Saúde e Bem-estar*, *Transporte*, *Presentes* e *Outros* para despesas, *Salário* e *Rendimento* para receitas).

Figura 4.2 – Formulário de Inserção Manual



Fonte: Produzido pelo autor.

4.3. Inserção por QR-Code

Quando se deseja inserir um conjunto de operações via leitura de nota fiscal eletrônica, o aplicativo ativa a câmera do smartphone e pede ao usuário para apontá-la para o QR-code da nota, como visto na Figura 4.3. O leitor de QR-code é implementado pelo próprio aplicativo, exigindo apenas que o dispositivo do usuário disponha de uma câmera, mas dispensando-lhe a necessidade de possuir ou instalar outro software para esta tarefa.

Figura 4.3 – Tela de Captura de QR-code

Fonte: Produzido pelo autor.

Tendo capturado o código, o programa acessa a versão online da nota pelo portal da Secretaria da Fazenda e extrai os itens registrados. Após a extração, o usuário visualiza a listagem das operações obtidas, organizadas por itens de compra separados. A Figura 4.4 apresenta a tela com a listagem. Por fim, o usuário deve confirmar o salvamento da lista, retornando à tela de Visão Geral.

Figura 4.4 – Listagem dos itens extraídos da NFC-e

Fonte: Produzido pelo autor.

4.4. Visualização das operações e saldo das contas

A partir do momento em que as primeiras operações são salvas, a tela principal do sistema passa a exibir informações sobre os saldos das contas do usuário.

A parte superior da tela apresenta o saldo das contas selecionadas no período atual, permitindo ao usuário deslizar horizontalmente para navegar por outros períodos. O usuário também pode, através da barra do topo, selecionar a separação de períodos por mês ou ano. A barra do topo permite também selecionar quais as contas devem ser exibidas, podendo, por exemplo, ocultar a *Poupança* ou visualizar apenas o *Cartão de Crédito*.

Na segunda metade são apresentadas as operações do período atual. A lista apresenta as categorias com seus gastos totais, ordenadas de forma decrescente (Figura 4.5). Ao selecionar uma categoria suas operações são apresentadas em ordem temporal, exibindo suas descrições, valores, datas e contas atribuídas (Figura 4.6). A Figura 4.7 mostra ainda a visualização da tela na orientação *paisagem* (horizontal).

Figura 4.5 – Visão geral e a listagem decrescente de Categorias

Fonte: Produzido pelo autor.

Figura 4.6 – Categoria expandida exibindo as operações atribuídas a ela

Fonte: Produzido pelo autor.

Figura 4.7 – Layout da Visão Geral na orientação paisagem (horizontal)

Fonte: Produzido pelo autor.

5 AVALIAÇÃO

A avaliação de usabilidade do aplicativo foi realizada com cinco usuários, denominados Usuário 1 (U1), Usuário 2 (U2), Usuário (U3), Usuário (U4) e Usuário 5 (U5), de diferentes perfis que são detalhados abaixo.

1. U1: Desenvolvedor, 25 anos, possui bastante experiência com aplicativos e já utilizou apps de controle financeiro.
2. U2: UX designer, 27 anos, também possui bastante experiência com aplicativos e já utilizou apps de controle financeiro.
3. U3: Estudante, 21 anos, possui moderada experiência com aplicativos, mas nunca utilizou nenhuma ferramenta de controle financeiro.
4. U4: Comerciante, 45 anos, utiliza smartphone apenas para acessar redes sociais e nunca utilizou nenhuma ferramenta digital para controle financeiro.
5. U5: Executivo de contas, 46 anos, utiliza smartphone para o trabalho, além de acessar redes sociais. Nunca utilizou uma ferramenta digital para controle financeiro.

A seguir é apresentada a metodologia utilizada e os resultados obtidos.

5.1. Metodologia

A metodologia utilizada consistiu na realização de tarefas simples baseadas nas *user stories* descritas na Seção 3.1.1. Cada usuário informou previamente a sua experiência com aplicativos mobile e classificou cada tarefa com uma nota de 1 a 5 de acordo com a facilidade para realizá-la. As tarefas realizadas são descritas abaixo.

5.1.1 Inserção Manual

Nesta tarefa foi solicitado aos usuários para cadastrar manualmente algumas operações no aplicativo, como depósitos de salário e pagamento de contas.

5.1.2 Inserção via NFC-e

Dando continuidade à tarefa anterior, os usuários tiveram de cadastrar despesas de supermercado através do leitor de nota fiscal eletrônica, conferindo os dados extraídos e categorizando as operações.

5.1.3 Visualização de saldos – Navegação

Após alimentar o sistema com dados, foi pedido aos usuários que explorassem a tela de visão geral, visualizando seu saldo, os gastos por categorias, expandindo-os para visualizar suas operações e navegando entre os meses registrados.

5.1.4 Visualização de saldos – Configuração

Por fim, os usuários puderam explorar as funções de configuração da visualização, selecionando quais contas apresentar e qual a periodicidade utilizada para acumular as operações.

5.2. Resultados

Com base nas respostas, foi elaborada a Tabela 5.1, apresentada a seguir, com as notas de cada usuário para cada tarefa realizada.

Tabela 5.1 – Resultados do Teste de Usabilidade

		U1	**U2**	**U3**	**U4**	**U5**
Perfil	**Profissão**	Desenvolvedor	UX Designer	Estudante	Comerciante	Executivo de Contas
	Idade	25	27	21	45	46
	Gênero	Masc.	Fem.	Fem.	Fem.	Masc.
	Experiência Prévia	5/5	5/5	3/5	1/5	2/5
Inserção Manual		4/5	5/5	4/5	4/5	5/5
Inserção via		5/5	4/5	5/5	5/5	4/5

NFC-e					
Saldo – Navegação	5/5	5/5	5/5	4/5	4/5
Saldo – Configuração	5/5	5/5	4/5	4/5	5/5

Fonte: Produzido pelo Autor

6 CONCLUSÕES

Este estudo teve como objetivo desenvolver um aplicativo de controle financeiro pessoal focado em oferecer uma interface amigável através dos recursos de leitura de NFC-e e visualização de saldos intuitiva.

Os resultados obtidos apresentam médias variando entre 4,4 e 4,6, com o desempenho maior nas tarefas 2 e 3, respectivamente Inserção via NFC-e e Navegação na Visualização de Saldos, justamente os itens mais relacionados ao objetivo geral da pesquisa. Isto demonstra que o aplicativo desenvolvido, apesar do escopo reduzido, cumpriu satisfatoriamente o seu propósito.

No escopo do trabalho foram apresentados outros aplicativos similares. Após a conclusão do teste de usabilidade é possível realizar uma análise comparativa entre as ferramentas selecionadas e o aplicativo desenvolvido, como destacado na Tabela 6.1.

Tabela 6.1 – Análise comparativa dos Aplicativos Similares com GibeMoni

	Monefy	**Minhas Economias**	**Money Manager EX**	**Guia Bolso**	**GibeMoni**
Acesso	Offline (Online na versão PRO)	Offline e Online	Offline e Online	Online	Offline
Plataformas Adicionais	Nenhuma	Web	Desktop	Web	Nenhuma
Modos de Inserção	Manual (Importação na versão PRO)	Manual	Manual e Importação (.OFX)	Manual, Sincronização de Contas	Manual, via NFC-e
Internacio-nalização	Idiomas e moedas	Não	Idiomas e moedas	Moedas (real e dólar)	Permite internaciona-lizar idiomas
Foco	Simplicidade, "*Friendliness*"	Funcionalidades	Customização, Crossplatform, Open Source	Automatização, *Gamification*	Facilidade de Inserção e Visualização

Downloads	500.000 a 1.000.000	100.000 a 500.000	50.000 a 100.000	1.000.000 a 5.000.000	

Fonte: Produzido pelo Autor.

Antes do desenvolvimento do aplicativo considerava-se que todas estas características seriam relevantes. Contudo, verificou-se que funcionalidades como acesso online (sincronização com servidor web), plataformas adicionais — em especial web — e internacionalização eram secundárias ao cumprimento do objetivo proposto. Essas funcionalidades não foram desenvolvidas e podem ser consideradas as maiores limitações do trabalho.

Estas três funcionalidades podem ser abordadas em trabalhos futuros. Além destas, propõe-se outros tópicos, detalhados abaixo.

- Gerenciamento — criação e edição — de contas e categorias
- Subcategorias ou tags, para especificar melhor os gastos (“ônibus” e “combustível” dentro da categoria “transporte”, por exemplo)
- Diferenciação de contas — cartões de crédito, por exemplo, possuem limite e não saldo
- Transferências entre contas, para permitir o registro de saques (transferência de conta corrente para carteira) e pagamentos de cartão de crédito, por exemplo
- Importação de arquivos, como o formato .OFX, de transações financeiras, ou o .CSV, para dados tabulares como um extrato bancário
- Recomendação de categorias na leitura de notas fiscais — inferir, por exemplo, que uma compra realizada numa padaria pertença à categoria “Alimentação”.

Por fim, além das funcionalidades descritas acima, um novo teste de usabilidade — contendo questões com resposta aberta, para permitir uma análise qualitativa, além de entrevistar mais usuários, garantindo uma análise quantitativa mais refinada — é também proposto como trabalho futuro.

REFERÊNCIAS

ABEGG, M. **Desenvolvimento de um Protótipo de Solução Colaborativa para o Controle de Listas de Compras**. 2014. Monografia (Graduação em Ciência da Computação) — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/handle/10183/110331>

BURBECK, Steve. **Applications Programming in Smalltalk-80:** How to use Model-View-Controller (MVC). [S.l.:s.n], 1992.

CASH, GNU. Disponível em: <http://www.gnucash.org/>. Acesso em: 30 de nov. de 2015.

CERBASI, Gustavo. **Como organizar sua vida financeira:** inteligência financeira pessoal na prática. Rio de Janeiro: Elsevier, 2009.

COHN, Mike. **User Story Applied:** for Agile Software Development. Boston: Addison-Wesley, 2004.

DEBASTIANI, Carlos Alberto. **Definindo Escopo em Projetos de Software**. São Paulo. Novatec, 2015.

FAZENDA, M. da. **Nota Fiscal Eletrônica**. Disponível em: <http://www.nfe.fazenda.gov.br>. Acesso em: 30 de nov. 2015.

FIGUEIREDO, A. T. **Suporte a Notas Fiscais Eletrônicas e Integração com Facebook em Aplicativo Android para Gerenciamento de Listas de Compras Colaborativas**. 2015. Monografia (Graduação em Ciência da Computação) — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/handle/10183/126076>

FOWLER, Martin. **Patterns of Enterprise Application Architecture**. Boston: Addison-Wesley, 2002.

GALHARDO, Maurício. **Finanças pessoais:** uma questão de qualidade de vida. São Paulo: Totalidade, 2008.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns:** Elements of Reusable Object-Oriented Software. [S.l.]: Addison-Wesley, 1994.

GOOGLE. **Activities | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/components/activities.html>. Acesso em: três de dez. de 2015.

GOOGLE. **Adapters | Android Developers**. Disponível em < http://developer.android.com/intl/pt-br/reference/android/widget/Adapter.html>. Acesso em três de nov. de 2015.

GOOGLE. **AdapterView | Android Developers**. Disponível em http://developer.android.com/intl/pt-br/reference/android/widget/AdapterView.html>. Acesso em: três de dez. de 2015.

GOOGLE. **Fragments | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/components/fragments.html>. Acesso em: três de dez. de 2015.

GOOGLE. **Layouts | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/ui/declaring-layout.html#AdapterViews> . Acesso em três de dez. de 2015.

GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.monefy.app.lite>. Acesso em: 26 de ago. de 2015.

GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.minhaseconomias>. Acesso em 26 de ago. de 2015.

GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store/apps/details?id=com.money.manager.ex>. Acesso em: 26 de ago. de 2015.

GOOGLE. **Play Store**. <https://play.google.com/store/apps/details?id=br.com.guiabolso>. Acesso em: 26 de ago. de 2015.

GOOGLE. **Salvando dados em bancos de dados do SQL | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/training/basics/data-storage/databases.html>.

GOOGLE. **Storage Options | Android Developers**. Disponível em : <http://developer.android.com/intl/pt-br/guide/topics/data/data-storage.html#db>. Acesso em: três de dez. de 2015.

GOOGLE. **UI Overview | Android Developers**. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/ui/overview.html>. Acesso em: três de dez. de 2015.

MACALÃO, P. R. **Check in Poa:** um Aplicativo Android para turistas em Porto Alegre. 2013. Monografia (Graduação em Ciência da Computação) — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/handle/10183/86434>

MONEY, Microsoft. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=20738>. Acesso em: 30 de nov. de 2015.

QUICKEN. Disponível em: <http://www.quicken.com/>. Acesso em: 30 de nov. de 2015.

SCRUM Alliance. Disponível em: <https://www.scrumalliance.org/>. Acesso em 10 de dez. de 2015.

SEFAZ-RS. **Nota Fiscal Eletrônica no RS**. Disponível em: <www.sefaz.rs.gov.br>. Acesso em: 30 de nov 2015

SOUZA, M. V. **SACI:** Sistema de Apoio à Coleta de Informações. 2014. Monografia (Graduação em Ciência da Computação) — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/handle/10183/100293>

WAVE, Denso. **History of QR Code**. Disponível em: <http://www.qrcode.com/en/history/> Acesso em: 3 de dez. de 2015.

WAVE, Denso. **Types of QR Code**. Disponível em: <http://www.qrcode.com/en/codes/> Acesso em: 3 de dez. de 2015.