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

ANDRE TRINDADE FIGUEIREDO

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

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

Prof. Dr. Leandro Krug Wives
Orientador

Porto Alegre, junho de 2015

AGRADECIMENTOS

Agradeço à minha família, à minha namorada, Rayane, e aos meus amigos pelo apoio e incentivo durante esse curso. Agradeço ao meu orientador, professor Leandro Krug Wives, que tornou esse trabalho possível.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
REST	Representational State Transfer
SOAP	Simple Object Access Protocol

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Com a popularização das mídias sociais e dispositivos móveis e a criação das notas fiscais eletrônicas, é necessário que a tecnologia esteja em constante avanço para ajudar as pessoas a gerenciar suas listas de compras de maneira colaborativa. Com a finalidade de obter um aplicativo que acompanhe esse cenário, este trabalho descreve os passos para a implementação do suporte a notas fiscais eletrônicas, códigos de barras e integração com a rede social Facebook em um aplicativo cliente para o sistema operacional Android e um servidor Web RESTful.

Palavras-chave: Código QR, código de barras, nota fiscal eletrônica, Facebook, Android, REST.

Electronic Bills Support and Facebook Integration in Android Application for Collaborative Shopping Lists Management

ABSTRACT

Along with the popularization of social networks e mobile devices and the creation of electronic bills, it is necessary that the technology is constantly advancing to help people collaboratively manage their shopping lists. To obtain a mobile application that follows this scenery, this work presents the steps to implement support to electronic bills, barcodes and integration to Facebook social network in a client application to Android operating system and a RESTful Web server.

Keywords: QR code, barcode, electronic bills, facebook, android, REST.

1 INTRODUÇÃO

Os dispositivos móveis têm conquistado cada vez mais o seu espaço na vida cotidiana das pessoas. O seu sucesso comercial atual tem atraído a atenção do setor de tecnologia, promovendo um rápido avanço das tecnologias relacionadas. Segundo um estudo da Cisco (2015), há uma previsão de crescimento de 800% do tráfego de dados móveis de 2014 a 2019, três vezes maior que a crescimento da internet cabeada.

Esse cenário combinado com outro sucesso atual, as mídias sociais, torna as pessoas cada vez mais interconectadas. O que antigamente era feito somente pessoalmente ou via telefonemas, agora tem tido o apoio das mídias sociais e aplicativos de conversação via Internet. Em pesquisa, "68% dos usuários de smartphones afirmam usá-lo para acessar redes sociais, 60% para ler notícias [..] e 20% usam para comprar."(TELEFONICA, 2014). A mesma pesquisa afirma que houve um aumento no uso de smartphones por jovens de 18 a 30 anos, passando de 63%, em 2013, para 78%, em 2014.

No caso de uso descrito neste trabalho, pode-se estabelecer uma trajetória ao longo dos últimos anos. Por exemplo, uma pessoa de uma família vai ao mercado, criando uma lista de compras e outra fica em casa. Nem sempre quem foi ao mercado lembra de todos os produtos. Não raramente, quem fica em casa lembra de mais produtos a inserir na lista quando a outra pessoa já está no mercado. Também, é muito comum que listas de compras sejam repetidas, como listas com itens essenciais de alimentação e higiene.

Antigamente, para lembrar dos produtos a comprar no mercado, as pessoas recorriam à memória ou ao papel. Às vezes, contavam até com a companhia de outra pessoa no mercado para ajudar a lembrar da lista de compras. Mais recentemente, com a popularização do telefone móvel, foi possível efetuar ligações para a comunicação sobre a lista de compras. Atualmente, a gama de opções foi ampliada com o surgimento dos aplicativos de mensagens via Internet, como o WhatsApp e Facebook Messenger, dado o menor custo.

Ainda que seja possível usufruir das vantagens do uso de mensagens via Internet, é uma solução que não é especializada em listas de comprar ou com recursos específicos para tal. Mesmo assim, ainda é preciso recorrer à memória ou a uma pesquisa no histórico de mensagens dos aplicativos para repetir uma lista de compras.

Com o crescente avanço dos dispositivos móveis e da tecnologia dos dados móveis,

recorre-se cada vez mais ao seu auxílio para realizar as tarefas cotidianos mais simples. A facilidade de ter uma lista de compras compartilhada acessível de maneira independente da localização, com recursos especificadamente destinados a tal, pode garantir uma simplicidade maior na realização das compras no mercado.

Há soluções alternativas no mercado de aplicativos móveis especializados em listas de compras. Algumas delas serão apresentadas na seção 2.6. Apesar disso, não pode ser verificado em nenhum desses aplicativos no presente momento, compartilhamento de listas de compras com amigos de uma mídia social e recursos como leitura de código de barras e de notas fiscais eletrônicas, mas grande parte tem autenticação integrada com o Google e Facebook.

O trabalho proposto por Abegg (2014) prevê um aplicativo de listas de compras colaborativas, porém não contém controle de usuários, como controle de contas, autenticação e listas por usuário, e recursos de leituras de códigos de barras e notas fiscais eletrônicas. Apesar de armazenar produtos em sua base de dados, não há o recurso de auto completar na interface do aplicativo.

1.1 Objetivo

Este trabalho estende o trabalho de Abegg e tem o objetivo de descrever o processo de desenvolvimentos de diversas funcionalidades novas ausentes no trabalho dele. Essas funcionalidades são: controle de usuários, integração com rede social Facebook, compartilhamento de listas com amigos do Facebook que usam o aplicativo e leituras de códigos de barras de produtos e de notas fiscais eletrônicas.

Com a autenticação integrada das mídias sociais, tem uma inegável facilidade para a administração de contas de usuário e o compartilhamento de listas com amigos. A introdução das controle de contas de usuário permite listas de compras por usuário. Com a criação das notas fiscais eletrônicas (NF-e) tornou-se possível replicar uma lista de compras instantaneamente a partir da leitura de uma nota fiscal com a impressão do código QR, para acesso à NF-e.

Além disso, funcionalidades secundárias serão introduzidas, como recurso de auto completar com nome de produtos, e *cache* local da base de produtos para maior rapidez e independência da rede móvel.

1.2 Estrutura do texto

Este trabalho está organizado em cinco capítulos. O capítulo 1 contextualiza e apresenta a causa da motivação do trabalho. Após, são apresentados o objetivo e a organização do texto. O capítulo 2 introduz o desenvolvimento de aplicativos móveis para a plataforma Android usando a biblioteca Facebook SDK para a integração com a rede social e outros

componentes usados, além do estilo arquitetural REST, trabalhos relacionados e soluções existentes. O capítulo 3 apresenta a modelagem e a arquitetura do sistema. O capítulo 4 descreve o processo de implementação da solução proposta, detalhando sobre o desenvolvimento do servidor web e do cliente Android. O capítulo 5 apresenta o funcionamento do aplicativo implementado e suas funcionalidades. O capítulo 6 descreve a avaliação por usuários reais do aplicativo implementado. Finalmente, o capítulo 7 apresenta as conclusões, retrospectiva sobre o trabalho e resultados obtidos, e relaciona funcionalidades que podem ser adicionadas à solução.

2 FUNDAMENTAÇÃO TEÓRICA E TECNOLOGIAS USADAS

Este capítulo apresenta uma introdução ao desenvolvimento de aplicativos móveis para a plataforma Android. A abordagem descreve a integração com os componentes usados para a integração com a rede social Facebook e a leitura de produtos por códigos de barras e de notas fiscais pelo código QR. Também é apresentado o estilo arquitetural REST, implementado no servidor web, além de trabalhos relacionados e soluções já existentes no mercado.

2.1 Códigos de barras

Os códigos de barras são representações gráficas de dados alfanuméricos administrados pela GS1, organização internacional, neutra e sem fins lucrativos. A GS1 trabalha em conjunto com governos e setores da indústria e comércio para desenvolver e manter diversos padrões com foco no varejo, logística e assistência médica. Outros padrões envolvidos são relativos a processos de negócios, como uma rede global destinada ao compartilhamento de dados de produtos (GSDN), transportes e logística, como de rastreabilidade de produtos, além de chaves de identificação universais para produtos, serviços, documentos e até componentes automotivos. Por sua vez, a GS1 surgiu da associação de duas associações organizações americanas e europeias com fins muito semelhantes, criadas a partir da criação do primeiro código de barras, na década de 70 (GS1, 2015a).

Por terem padrões de representação bem estabelecidos e simples, os códigos têm o intuito de ser universalmente legíveis, facilitando o reconhecimento dos dados através da leitura ótica por infravermelho independentemente de localidade ou língua. Segundo a GS1 (2015b), os códigos são classificados em dois tipos: **unidimensional**, ou linear, e **bidimensional**, ou matriz.

Os códigos de barras unidimensionais comumente representam identificadores únicos de produtos. Um código de barra linear consiste em uma série de barras com larguras variáveis. De acordo com a empresa Simplesmente Código de Barras (2015) - comercializadora de códigos de barras, o formato mais conhecido é o EAN/UPC, mais especifi-

camente no Brasil e para nosso estudo, o formato EAN-13. Esse formato é o mais usado pelos produtos disponíveis em um supermercado. Esse código de barras representam um número de treze dígitos, onde os três primeiros identificam o país, outros nove identificam empresa e produto, e o último é um dígito verificador.

Segundo Prass (2011), os códigos de barra bidimensionais comumente representam dados alfanuméricos, tipicamente endereços de páginas web. Um código de barra bidimensional consiste em diversos blocos quadrados ou até círculos. O formato mais famoso é o código QR, presente em algumas notas fiscais de mercado quando há uma versão eletrônica desta – a Nota Fiscal Eletrônica, apresentada na próxima seção.

2.2 Nota fiscal eletrônica

Segundo o Ministério da Fazenda (2015), a Nota Fiscal Eletrônica (NF-e) é um documento com validade jurídica garantida pela assinatura digital, e emitido e armazenado eletronicamente equivalente a nota fiscal física. O projeto iniciou-se em 2005 com a presença de grande empresas e secretarias seis estados: Rio Grande do Sul, Santa Catarina, São Paulo, Goiás, Bahia e Maranhão. A Receita Federal coordena o projeto, atendendo a uma Emenda Constitucional de 2003, a qual determina uma integração das administrações tributárias da União e de suas Unidades Federativas para o compartilhamento de informações fiscais.

Entre os diversos benefícios entre os variados setores, estão (FAZENDA, 2015):

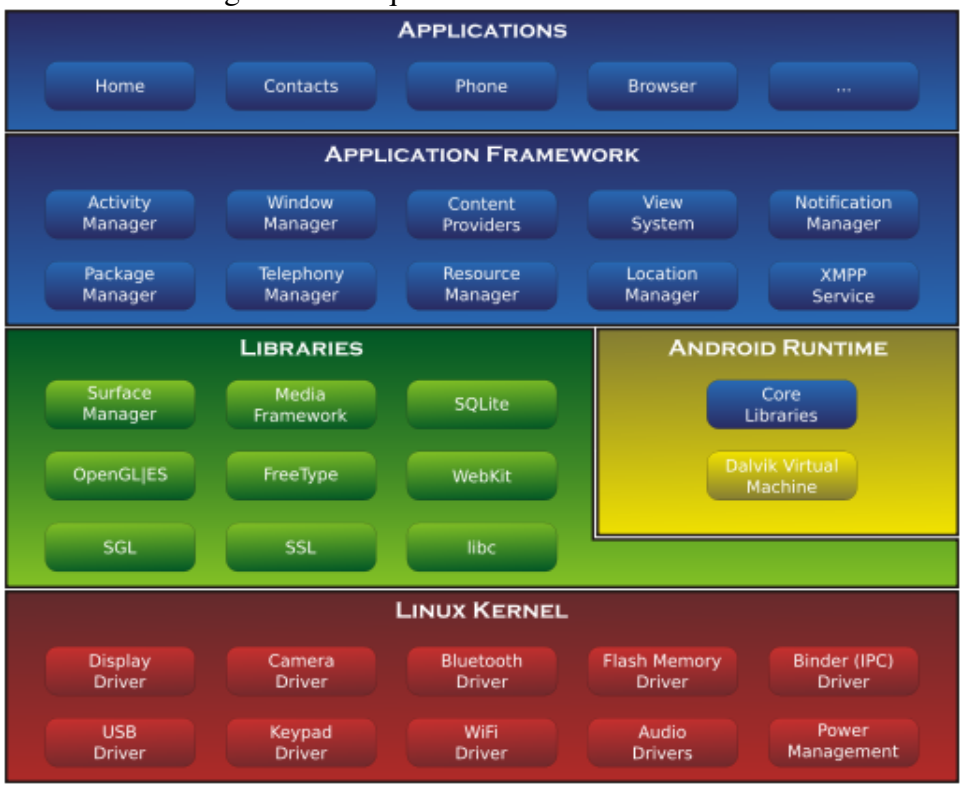
- Diminuição da sonegação e aumento da arrecadação;
- Impacto positivo no meio ambiente;
- Incentivo ao comércio eletrônico e ao uso de novas tecnologias;
- Redução de custos para o emissor.

As secretarias da Fazenda de cada estado mantêm uma página web, onde é possível verificar uma NF-e, apresentada do mesmo modo que uma nota fiscal física. Quando um mercado emite uma NF-e no ato da compra, geralmente é impresso também a nota fiscal física com informações para acesso a NF-e, como endereço web e chave de acesso, além de um código QR representando um endereço web para acesso direto à página com a NF-e por um dispositivo móvel.

2.3 Desenvolvimento de aplicativos móveis e a plataforma Android

Esta seção descreve a plataforma Android e as bibliotecas usadas para a implementação do trabalho. O desenvolvimento para aplicativos móveis difere em alguns aspectos do desenvolvimento para aplicativos web. Algumas diferenças a serem notadas são (CONSULTANTS, 2011):

Figura 2.1: Arquitetura do Sistema Android.



Fonte: (TEAM, 2009)

- Modelo e fluxo de interação do usuário com o dispositivo móvel;

- Restrições causadas pela latência, modo trabalhar com a *cache* e carregamento do aplicativo;

- Taxas de transferências de dados mais baixas do que por cabo, e disponibilidade de tráfego sujeita a maior instabilidade.

O sistema operacional Android foi inicialmente desenvolvido pela empresa Android, Inc., a qual recebia suporte financeiro da Google. Em 2005, ela foi adquirida pela Google. A Google continuou o seu desenvolvimento em conjunto com a Open Handset Alliance, revelando-o em 2007 e lançando-o em seu primeiro dispositivo móvel em 2008. A Open Handset Alliance é um consórcio de empresas de hardware, software e telecomunicações, cujo objetivo é o desenvolvimento de tecnologias móveis abertas. O código do sistema operacional é disponibilizado pelo Google sob licença de código aberto.

2.3.1 Arquitetura Android

A arquitetura Android, ilustrada na Figura 2.1, é descrita em cinco camadas distribuídas em quatro níveis. Tais camadas e níveis são descritos a seguir.

2.3.1.1 Sistema Operacional

Chamada de Kernel Linux no Android, uma vez que usa o núcleo do sistema operacional Linux. Ela é responsável por serviços de mais baixo nível da plataforma, como gerenciamento de memória e processos, segurança, etc.

2.3.1.2 Bibliotecas

A camada de bibliotecas fornece suporte através de APIs para fontes de texto, mídia, segurança dos dados, rede, internet, gerência de superfícies e biblioteca nativas (libc).

2.3.1.3 Android Runtime

No mesmo nível da camada das Bibliotecas, a camada de execução (*Android Runtime*) oferece suporte para execução de aplicativos desenvolvidos para a plataforma Android, e vem passando por uma transformação ao longo das últimas versões do Android. Nas versões anteriores ao Android 5.0, a camada continha a Máquina Virtual Dalvik (DVM), derivada da Máquina Virtual Java e otimizada para o ambiente móvel, usando pouca memória e permitindo múltiplas instâncias. A DVM foi substituída pelo Android Runtime a partir do Android 5.0, estando essa última já presente como transição desde a versão 4.4 do Android. A Android Runtime, transforma os *bytecodes* compilados pela máquina virtual, em instruções nativas. A eliminação de código interpretado adiciona maior velocidade de execução e reduz o consumo de memória e energia.

2.3.1.4 Framework de Aplicação

Esta camada contém APIs usadas pelos aplicativos, como por exemplo controle de localização, da câmera e de notificações. Elas são disponibilizadas como classes Java no Framework de desenvolvimento Android para os desenvolvedores.

2.3.1.5 Aplicação

Camada onde residem as aplicações disponíveis ao usuário final, como as providas nativamente pelo sistema, como por exemplo Agenda e Calendário, por meio dos fabricantes ou instaladas pelo usuário pela loja de aplicativos Google Play, ou outros meios.

2.3.2 Conceitos básicos

O framework de desenvolvimento do Android fornece componentes para a construção de uma aplicação. Os principais são os Serviços (*Services*) e as Atividades (*Activities*). Um Serviço é um componente que executa tarefas de longa duração em segundo plano sem usar a interagir com o usuário. Isto significa que o usuário pode abrir outro aplicativo no dispositivo e o serviço continuará a execução. Eles podem se comunicar com outras aplicações, tocas música, efetuar operações de leitura e escrita de arquivos, entre outros. Uma Atividade é um componente que provê uma interface para a interação do usuário com

a aplicação. Como este trabalho fez uso de Atividades, elas serão detalhadas a seguir.

2.3.2.1 Atividades

Como introduzido anteriormente, Atividades são componentes que fornecem interfaces para aplicação. Atividade são iniciadas a partir de outras Atividades, sobrepondo as antigas formando o que pode ser chamado de fluxo da aplicação aos olhos do usuário. Em caso específico, a atividade carregada inicialmente ao executar o programa, costuma ser chamada de Atividade Principal ("Main Activity"). Como analogia, as Atividades e suas interações podem ser comparadas às páginas web de uma aba de um navegador, onde cada Atividade corresponde a uma página web, de onde podem ser abertas outras páginas, mas havendo somente e só uma página aberta ativa nesta aba. Desse mesmo modo, a Atividade Principal poderia ser comparada à página inicial de um site web.

Segundo a Google (2014a), o ciclo de vida de uma atividade contém estados para indicar o seu estágio, entre eles, os principais são:

***Created*:** Indica que a atividade foi criada. A partir desse momento, as suas variáveis e atributos são mantidas na memória até o encerramento da atividade, mesmo que outra atividade tome o primeiro plano.

***Resumed*:** Indica que a atividade está em primeiro plano e o usuário pode interagir com ela.

***Paused*:** A atividade está somente parcialmente coberta por outra atividade, mais comumente porque esta última não está usando todo o espaço disponível deixando parte da atividade pai visível.

***Stopped*:** A atividade está totalmente coberta. Geralmente, a atividade entra nesse estado após chamar uma nova atividade que a cubra, até o fim desta última, retornando para a atividade pai.

***Destroyed*:** Indica que a atividade foi encerrada. Toda a informação é apagada da memória.

A cada transição entre esses estados, o sistema invoca a execução de métodos como quando uma Atividade está sendo criada (*onCreate*) ou lançada para o segundo plano (*onPause*). Os principais métodos são:

***onCreate*:** Chamado imediatamente ao criação da atividade, esse método deve sempre ser implementado. É onde os componentes são inicializados, como atributos e o mais importante, a definição do layout a ser usado.

***onResume*:** Esse método é invocado quando inicia ou reinicia a interação da atividade com o usuário, isto é, a partir dos estados *Created*, *Resumed* ou *Paused*.

***onPause*:** Esse método é invocado quando o usuário está deixando a atividade. Apesar de que seu retorno possa ser possível, é recomendável que sejam alterações sejam salvas para a camada de persistência.

O diagrama de um ciclo de vida de uma Atividade é representado na Figura 2.2.

2.4 Desenvolvimento Web RESTful

O desenvolvimento web RESTful baseia-se no estilo arquitetural REST, acrônimo para *Representational State Transfer*, ou Transferência de Estado Representacional em português. REST consiste em uma série de restrições aplicadas a componentes de um sistema de hipermídia e foi proposto inicialmente por Fiedling (2000), em sua tese de doutorado. Como afeta um nível alto de abstração, lidando com o *design* de arquitetura, é independente de implementação. As principais vantagens do REST são performance, escalabilidade, simplicidade e portabilidade.

Valendo-se das melhores práticas para a construção de serviços web escalonáveis, é usualmente usado como alternativa ao uso do protocolo *SOAP* devido principalmente a maior simplicidade. Serviços web que seguem esse estilo arquitetural, são chamados de serviços RESTful.

Um serviço web RESTful tem as seguintes características (SILVA, 2011):

- Uma URI básica para o serviço, como por exemplo http://www.exemplo.com/recursos;
- Um tipo de mídia (MIME) de Internet para os dados suportados para o serviço, tipicamente formatos JSON e XML;
- Um conjunto de operações análogas a CRUD e definidas pelos principais verbos de métodos HTTP: PUT, GET, POST e DELETE, usadas para criar, visualizar, atualizar e deletar.

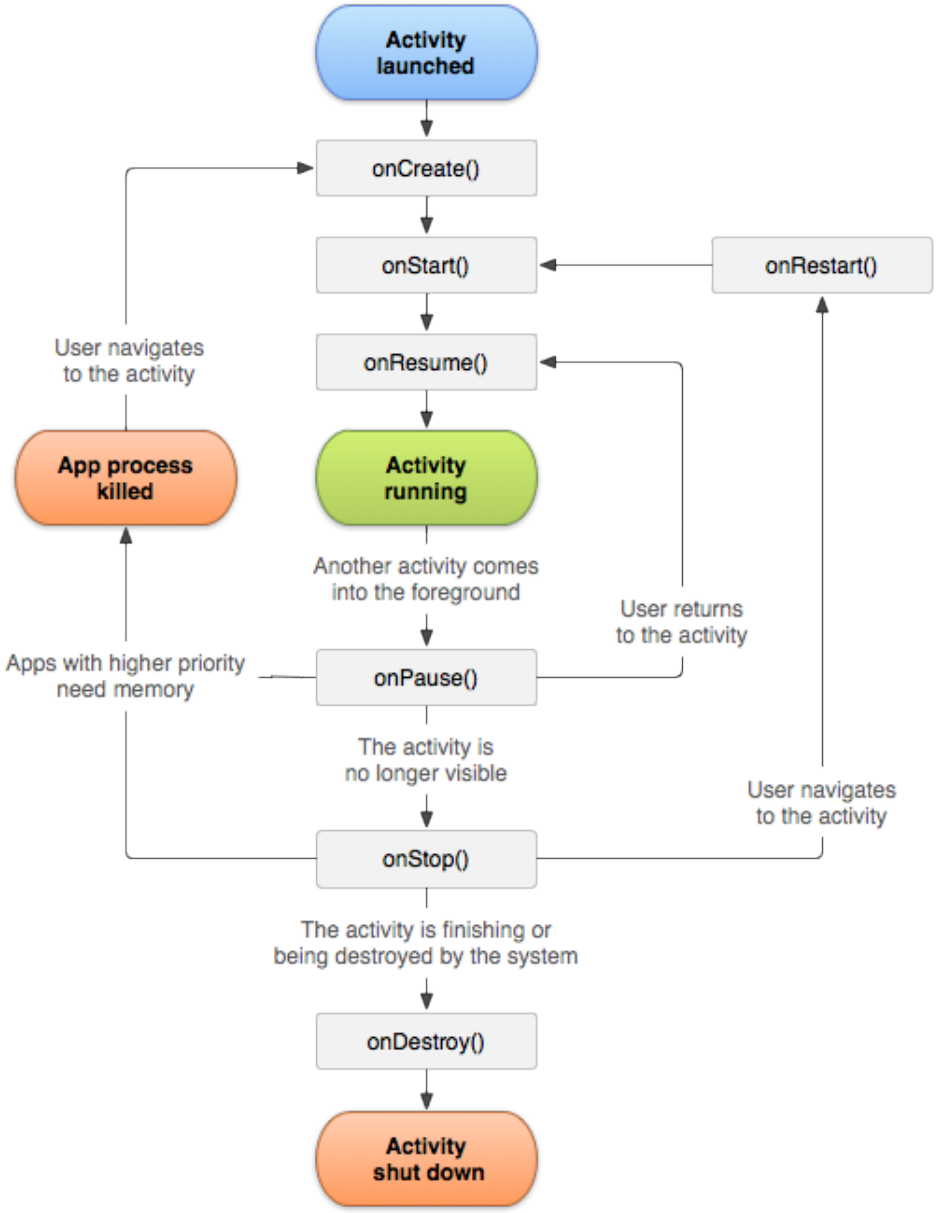
De acordo com Amundsen (AMUNDSEN, 2008), a série de restrições descritas anteriormente para caracterizar um aplicativo REST são:

Cliente-Servidor - Separação entre cliente e servidor interligando-os através de uma interface.

Protocolo *Stateless* - Comunicação entre cliente e servidor não deve manter estado entre requisições, isto é, todos os dados necessários devem ser enviados por requisição. Dados relativos à sessão devem ser mantidos no cliente.

Cache - Visando a performance e escalabilidade através da eliminação de respostas do servidor duplicadas, esta restrição estabelece que uma resposta do servidor pode ser definida como *cacheable* ou não. Deste modo, *cache* bem aplicados eliminam

Figura 2.2: Ciclo de vida de uma Atividade.



Fonte: (GOOGLE, 2014a)

parcialmente ou completamente algumas interações desnecessárias entre cliente e servidor.

Sistema de camadas - Sistemas (servidores) ou componentes de um servidor em camadas. Ganha-se em performance, simplicidades e portabilidade. Um fator importante para o ganho de performance é a possibilidade de uso de balanceamento de servidores com a independência ganha com esse sistema.

Interface uniforme - Aplicando generalizações nos componentes, a implementação é desacoplada dos serviços. Tem como benefícios a simplicidade da arquitetura e a visibilidade das interações, e como desvantagem, a perda de eficiência pela troca da especialização pela generalização. Garante-se esse requisito através de identificação única de recursos pelo endereço, manipulação de recursos através de suas representações, mensagem de resposta auto descritivas através do uso adequado de métodos HTTP, e hipermídia como a máquina de estados da aplicação com ações sendo identificadas pelo cliente apenas da representação recebida previamente do servidor.

Código sob-demanda Restrição opcional, define que o servidor pode estender ou personalizar uma funcionalidade pela transferência de código executável, dos formatos Java *applets* ou scripts clientes, como Javascript.

2.5 Trabalhos relacionados

Nesta seção serão apresentados trabalhos relacionados ao objetivo deste trabalho, i.e., que envolvam desenvolvimento Android e desenvolvimento Web RESTful. O estudo dos trabalhos relacionados construiu a base para a definição das funcionalidades aqui propostas.

2.5.1 Desenvolvimento de um Protótipo de Solução Colaborativa para o Controle de Listas de Compras

Trabalho do qual este trabalho proposto estende, Abegg (2014) propõe um protótipo para o gerenciamento compartilhado de listas de compras. Tal sistema é composto de um aplicativo cliente em Android e um servidor web RESTful que se comunicam através do protocolo HTTP, esse protótipo segue boas práticas de desenvolvimento, do qual foi possível adaptar para a introdução das novas funcionalidades propostas por este presente trabalho.

Por se tratar de um protótipo, ele não tem controle de usuários, que permitiria listas de compras por usuário. Além disso, este presente trabalho acrescenta funcionalidades de leitura de código de barras e notas fiscais eletrônicas.

2.5.2 Check in Poa

Em "Check In Poa: um aplicativo Android para turistas em Porto Alegre", Macalão (2013) apresenta o processo de implementação de um jogo inspirado no Monopólio para ajudar turistas a conhecer a cidade de Porto Alegre.

O trabalho apresenta com detalhes a plataforma Android e IDEs para o desenvolvimento de aplicativos para a plataforma no Android. Além disso, o trabalho descreve a integração do aplicativo Android com a API do Facebook.

2.5.3 PyRester

Em "PyRester: Uma abordagem baseada em modelos U2TP para geração de código de teste unitário para RESTful Web Services", Rosa (2011) propõe um método baseado em geração automática de código de testes a partir de modelos para a implementação de casos de teste unitário para serviços web RESTful. São apresentados conceitos importantes sobre o estilo arquitetural REST, sua utilização na implementação de serviços Web RESTful e boas práticas que devem ser seguidas.

2.5.4 SACI

Em "SACI: Sistema de Apoio a Coleta de Informações", Souza (2014) propõe um sistema para gerenciamento e coleta de dados diretamente no campo. O trabalho contém dois componentes: um cliente Android e um servidor RESTful. Conceitos básicos importantes sobre REST, Android e sua integração são apresentados.

2.6 Aplicativos semelhantes

Os aplicativos de gerenciamento de listas de compras disponíveis na Play Store, loja de aplicativos para Android da Google, com maior destaque, serão apresentados e terão algumas de suas funcionalidades comparadas.

Os aplicativos analisados são:

- Meu Carrinho: provavelmente o aplicativo mais usado no Brasil para gerenciamento de listas de compras, oferece uma série de funcionalidades, além de interface web.

- Batatas: protótipo no qual esse trabalho é baseado, propõe uma solução de gerenciamento colaborativo de listas de compras de código aberto, porém não prevê um controle de usuários.

- Out of Milk: aplicativo mais usado no mundo (GOOGLE, 2014b), tem autenticação integrada com o Google e o Facebook, porém com compartilhamento de listas somente por endereços de e-mail. Tem foco na simplicidade, além de interface web. Também tem recurso de leitura de código de barras.

Tabela 2.1: Comparação das funcionalidades presentes nos aplicativos semelhantes

Funcionalidades/Aplicativo	Shopping List	Meu Carrinho	Out of Milk	Batatas (ABEGG, 2014)
Múltiplas Listas de compras	**Sim**	**Sim**	**Sim**	**Sim**
Controle de Usuários	**Sim**	**Sim**	**Sim**	Não
Edição colaborativa de listas	Não	**Sim**	Não	**Sim**
Compartilhamento de listas	**Sim**	**Sim**	**Sim**	**Sim**
Categorização de Itens	**Sim**	Não	Não	Não
Suporte a múltipla plataformas	Não	**Sim**	Não	Não
Registro de Preço	Não	**Sim**	**Sim**	Não
Leitura de Código de Barras	Não	**Sim**	**Sim**	Não
Leitura de Notas fiscais (Nf-e)	Não	Não	Não	Não
Integração com Facebook	Não	**Sim**	**Sim**	Não
Código Aberto	Não	Não	Não	**Sim**

Fonte: adaptação de Abegg (2014, p. 44).

- Shopping List: aplicativo semelhante ao Out of Milk, porém não tem interface web e as listas são compartilhadas por números de telefone.

A comparação entre os produtos é apresentada na Tabela 2.1.

3 MODELAGEM E PROJETO DO APLICATIVO

Neste capítulo serão apresentados a identificação dos requisitos funcionais através de *user stories* e o modelos de entidades. Os requisitos não-funcionais serão omitidos, para manter o foco no desenvolvimento do protótipo que atenda ao propósito deste trabalho.

Este trabalho propõe uma extensão a um trabalho pré-existente realizado por Abegg (2015). Alguns aspectos do projeto não precisaram ser definidos, como algumas entidades e arquiteturas da implementação dos aplicativos servidor e cliente. No entanto, eles serão descritos integralmente.

3.1 Modelagem do sistema

Nesta seção serão enumeradas as *User Stories* do projeto como meio de apresentar os requisitos do sistema. Depois, serão os modelos de Entidades e seus tipos de dados serão introduzidos.

3.1.1 User Stories

Nesta seção serão apresentados os requisitos funcionais deste trabalho em adição ao protótipo pré-existente na forma de *user stories*. *User stories*, histórias de usuário em português, são descrições breves de funcionalidades visando a simplicidade e eficiência na identificação de requisitos. Geralmente, elas se limitam a uma frase no formato "Como um <tipo de usuário>, quero <ação/objetivo> para que <finalidade/benefício>.", onde a terceira oração é facultativa, segundo Cohen (2008).

3.1.1.1 Visualizar minhas listas de compras

"Como um usuário, quero visualizar as listas de compras criadas por mim ou compartilhadas comigo para que eu possa visualizar, adicionar, remover e marcar como adquiridos produtos a comprar."

3.1.1.2 Criar listas de compras a partir de outras já realizadas

"Como um usuário, quero criar uma lista de compras a partir de outra já realizada para facilitar a criação de listas recorrentes com algumas diferenças."

3.1.1.3 Marcar itens adquiridos a partir de nota fiscal

"Como um usuário, quero poder marcar múltiplos produtos de uma lista como adquiridos a partir de uma nota fiscal para que eu sabia quais produtos foram comprados."

3.1.1.4 Marcar item adquirido a partir de código de barra

"Como um usuário, quero marcar um produto como adquirido a partir do seu código de barra para que eu não precise procurá-lo na lista de compras."

3.1.1.5 Compartilhar lista com amigo

"Como um usuário, quero compartilhar uma lista de compras com um amigo para que possamos nos comunicar de maneira efetiva e direta em tempo real a distância sobre atualizações nessa lista."

3.1.2 Entidades

Naturalmente, este trabalho propõe também a ampliação de entidades especificações do protótipo de listas de compras.

Uma de suas propostas é um controle de usuário integrado com a rede social Facebook. Cada usuário poderá criar ter compartilhado consigo, várias listas de compras. Para tal, foram criadas as entidades User, que representa um usuário, e List_User, que representa a relação *n para n* entre listas e usuários.

Outras três entidades já existiam no trabalho pré-existente, e sofreram algumas modificações. São elas: List, que representa uma lista de compras, Product, que representa um produto, e Item, que representa a relação *n para n* entre listas e produtos.

O modelo Entidade Relacionamento é ilustrado na Figura 3.1. Chaves amarelas indicam chaves primárias, chaves vermelhas indicam chaves estrangeiras, campos preenchidos são campos não-nulos e campos vazados são campos nulos.

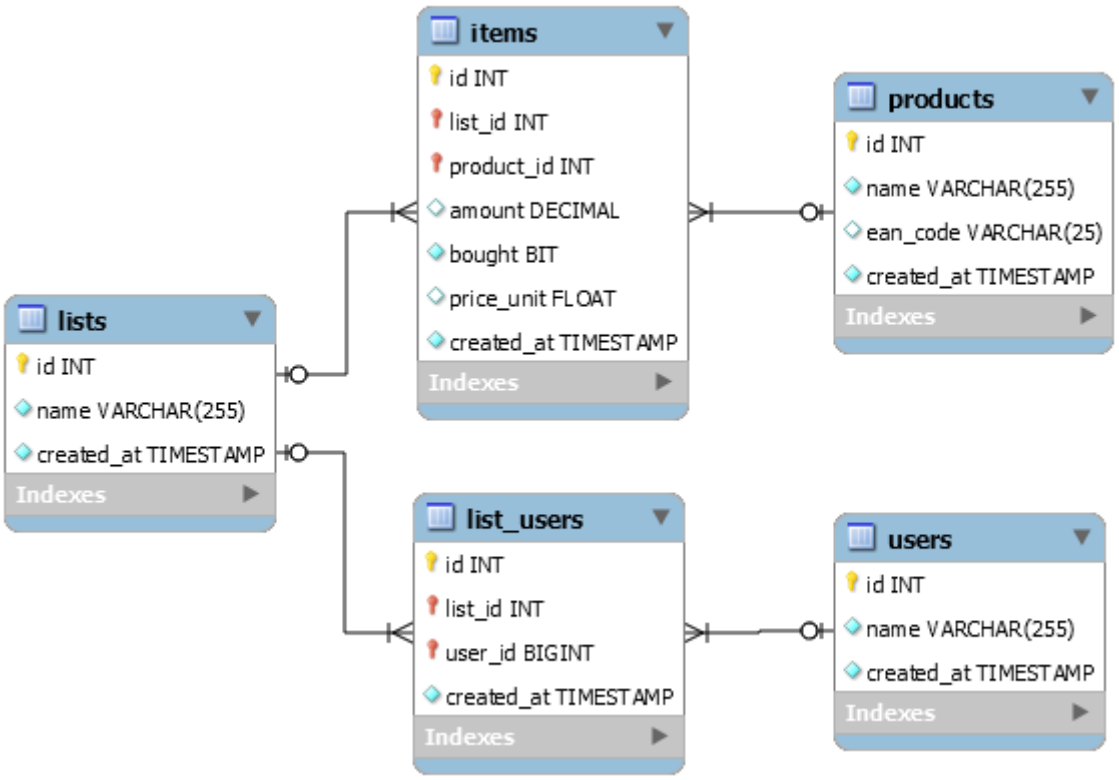
As entidades estão detalhadas a seguir.

*3.1.2.1 Lista de compras (*List*)*

Esta entidade representa uma lista de compras criada e compartilhada por usuários e contém uma coleção de produtos a adquirir. Foi modificada a partir da existente do trabalho de Abegg e suas alterações são apresentadas na subseção 4.1.1.1.

id: Identificador único da lista, é inteiro e auto incrementado.

Figura 3.1: Modelo Entidade Relacionamento.



name: Nome da lista de compras, e.g. "Lista para churrasco". Preenchimento obrigatório.

created_at: Data de criação da lista de compras. Preenchimento obrigatório com padrão data e hora atual do sistema, se não preenchido.

*3.1.2.2 Produto (*Product*)*

Esta entidade representa um produto que poderá constar em várias listas de compras. Esse produto tem duas origens: fornecido manualmente pelo usuário ou criado a partir de uma leitura de nota fiscal. Foi modificada a partir da existente do trabalho de Abegg e suas alterações são apresentadas na subseção 4.1.1.1.

id: Identificador único do produto, é inteiro e auto incrementado.

name: Nome do produto, e.g. "Sal grosso". Preenchimento obrigatório.

ean_code: Código numérico do produto, tipicamente uma sequência de 13 dígitos.

created_at: Data de criação do produto. Preenchimento obrigatório com padrão data e hora atual do sistema, se não preenchido.

*3.1.2.3 Item (*Item*)*

Esta entidade representa o relacionamento entre uma lista de compras e um produto, identificando que produtos constam numa determinada lista de compras. Foi modificada a partir da existente do trabalho de Abegg e suas alterações são apresentadas na subseção 4.1.1.1.

id: Identificador único da relação lista-produto, é inteiro e auto incrementado.

list_id: Identificador da lista de compras, relacionado a entidade List.

product_id: Identificador do produto, relacionado a entidade Product.

amount: Quantidade de produtos. Permite valores fracionados para representar gramas ou mililitros, por exemplo.

bought: Estado de aquisição, que pode ser somente Verdadeiro ou Falso. Preenchimento obrigatório com padrão Falso, se não preenchido.

price_unit: Preço unitário.

created_at: Data de adição do produto à lista de compras. Preenchimento obrigatório com padrão data e hora atual do sistema, se não preenchido.

*3.1.2.4 Usuário (*User*)*

Esta entidade criada, representa um usuário do sistema. Neste trabalho, contém os dados do usuário na rede social do Facebook.

id: Código numérico do usuário na rede social Facebook, e.g. "1439509100", nesta implementação, porém podendo ser um identificador único universal se houver mais de uma rede social usada para autenticação integrada.

name: Nome do usuário. Preenchimento obrigatório.

points: Número inteiro que representa os pontos do usuário. Preenchimento obrigatório com padrão "0", se não preenchido.

created_at: Data de criação do usuário. Preenchimento obrigatório com padrão data e hora atual do sistema, se não preenchido.

*3.1.2.5 Lista_Usuário (*List_User*)*

Esta entidade criada, representa o relacionamento entre uma lista de compras e um usuário, identificando que o usuário tem acesso a lista de compras. Portanto, essa entidade representa um compartilhamento, de fato.

id: Identificador único do produto, é inteiro e auto incrementado.

list_id: Identificador da lista de compras, relacionado a entidade List.

user_id: Identificador da lista de compras, relacionado a entidade User.

created_at: Data de criação da lista de compras pelo usuário ou data compartilhamento da lista de compras dado a ele. Preenchimento obrigatório com padrão data e hora atual do sistema, se não preenchido.

4 PROCESSO DE IMPLEMENTAÇÃO

Neste capítulo será apresentado o processo de implementação das funcionalidades e modelos descritos no capítulo anterior.

4.1 Desenvolvimento do Servidor

Nessa seção serão apresentadas as alterações realizadas no servidor web para atender as novas funcionalidades. O servidor foi desenvolvido na linguagem Ruby usando framework para aplicações web Sinatra, provendo um ambiente favorável a uma simples e rápida implementação. A biblioteca de acesso aos dados usada é a Sequel. A aplicação está hospedada no servidor Heroku.

A estrutura da implementação do servidor está representada na Figura 4.1.

4.1.1 Implementação dos Modelos

O protótipo preexistente contém três modelos de entidades implementadas. Nesta subseção, serão descritas alterações nos modelos preexistentes e adições de novos modelos no servidor, e figuras mostrarão os códigos implementados. Essas duas categorias de modelos são apresentadas a seguir.

4.1.1.1 Modificações em modelos preexistentes

Para atender aos novos requisitos, modelos preexistentes sofreram modificações, como adições de relacionamentos com novas entidades para controle de usuários e de campos para identificação de dados de códigos de barras. As modificações dos modelos preexistentes e as imagens com as alterações em destaque estão descritas a seguir.

List: Abstração da lista de compras. Tem como atributos o nome e um conjunto de *Items*. Anteriormente, havia alguns métodos que auxiliavam na adição e recuperação de itens, um método para verificar se há itens na lista e um método para representar a lista no formato JSON.

Foi adicionada nesse modelo, a informação da relação de entidade com a nova entidade *List_User*. Além disso, foi modificado um método auxiliar ao método

Figura 4.1: Estrutura de pastas e arquivos do servidor.

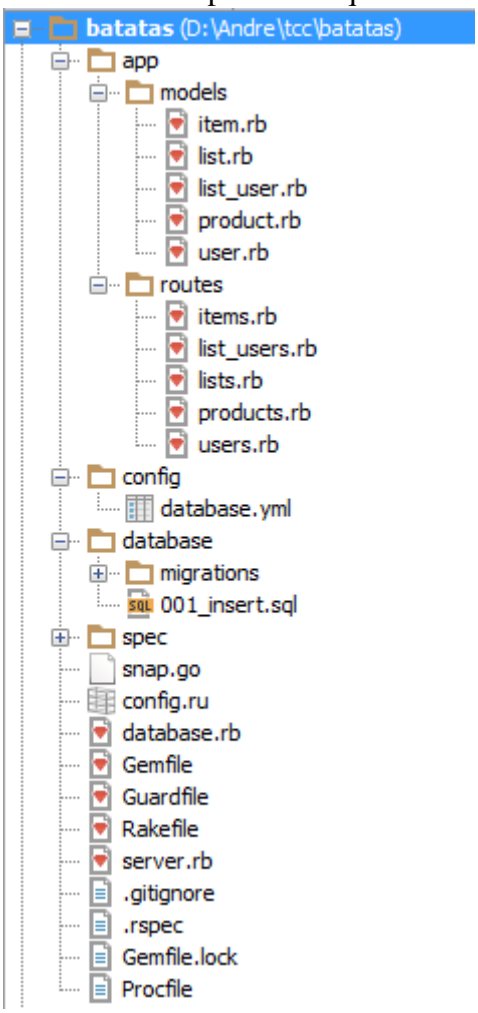


Figura 4.2: Arquivo list.rb. Alterações realizadas no modelo Lista.

```
class List < Sequel::Model
  one_to_many :items
  one_to_many :list_users

  def add(items)
    items = [items].flatten
    items.each { |item| add_item(from_json(item)) }
    save
  end

  def to_json
    {
        id: id,
        name: name,
        items: items.map(&:to_json)
    }
  end

  def item(id)
    items.find { |i| i.id == id}
  end

  def empty?
    items.empty?
  end

  private
  def from_json(item)
    record = Product.get_by_params(item['name'], item['ean_code'])
    options = {product_id: record[:id], amount: item['amount']}
    options[:bought] = item['bought'] if item['bought']
    options
  end
end
```

de adição de itens à lista. A implementação desse modelo com suas alterações é ilustrada na Figura 4.2.

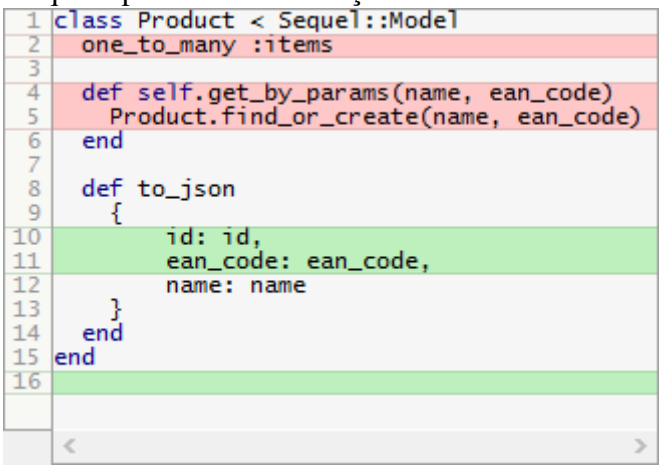
Product: Abstração de produtos criados pelo usuário para compor uma lista de compras. O modelo implementado no servidor contém um método para encontrar um produto pelo seu nome criando-o, se inexistente. Também há outro método para retornar a representação da instância de um produto no formato JSON.

O método para encontrar um produto pelo nome foi alterado para permitir o atributo de código de barras e refatorado com a substituição de uma função equivalente pertencente à biblioteca *Sequel*. Também, foram adicionados o identificador e o código de barras relativos ao produto na representação em JSON. A implementação desse modelo com suas alterações é ilustrada na Figura 4.3.

Item: Modelo que que relaciona uma lista de compras aos produtos que ela contém. Intermedeia a relação "*n para n*"entre listas de compras e produtos.

No modelo preexistente contém métodos para identificar se um produto da lista foi

Figura 4.3: Arquivo product.rb. Alterações realizadas no modelo Produto.



```
1  class Product < Sequel::Model
2    one_to_many :items
3
4    def self.get_by_params(name, ean_code)
5      Product.find_or_create(name, ean_code)
6    end
7
8    def to_json
9      {
10         id: id,
11         ean_code: ean_code,
12         name: name
13     }
14   end
15 end
16
```

adquirido ou não, e um método para representar no formato JSON uma instância de um item de uma lista identificando sua quantidade e seu estado de aquisição.

Para esse trabalho, foi adicionado à representação no formato JSON, o identificador da lista de compra correspondente e a informação do código de barras do produto. A implementação desse modelo com suas alterações é ilustrada na Figura 4.4.

4.1.1.2 Novos modelos

Além das alterações realizadas nas estruturas existentes, foram criados dois novos modelos necessários para a implementação do controle de usuários. Eles são descritos a seguir.

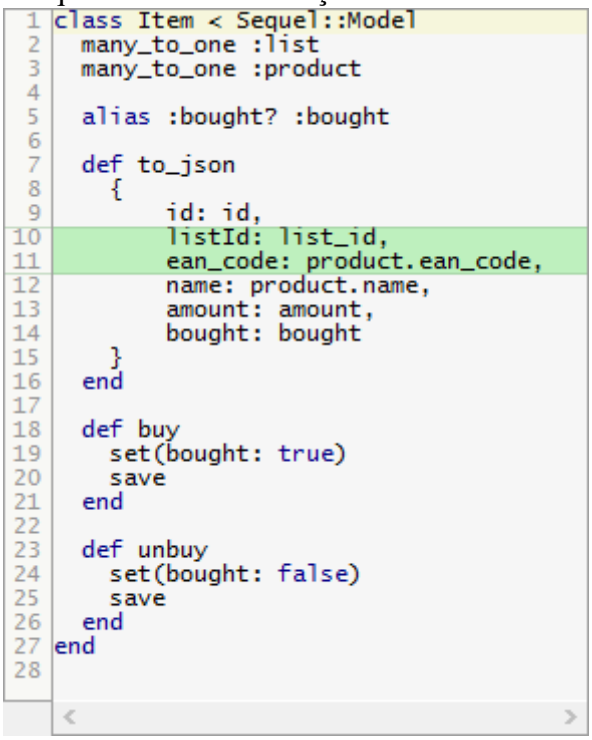
User: Modelo que representa um usuário cadastrado no sistema. Contém um método para adicionar um usuário pelo identificador numérico do Facebook e nome. Se existente mas o nome difere, o nome é atualizado. Contém também um método para representar um usuário no formato JSON, informando o identificador e nome. A implementação desse modelo é ilustrada na Figura 4.5.

List_User: Modelo que relaciona listas de compras a usuários, condensando a informação da lista de usuários que uma lista de compras possui. Possui um método que cria um relação entre uma lista e um usuário. É o compartilhamento de uma lista com um usuário. Além disso, possui métodos auxiliares para representação em formato JSON, necessários para listagem das listas de compras e usuário compartilhados. A implementação desse modelo é ilustrada na Figura 4.6.

4.2 Desenvolvimento do Cliente em Android

Nessa seção são apresentadas as alterações realizadas no aplicativo móvel para atender as novas funcionalidades.

Figura 4.4: Arquivo item.rb. Alterações realizadas no modelo Item.



```
class Item < Sequel::Model
  many_to_one :list
  many_to_one :product

  alias :bought? :bought

  def to_json
    {
        id: id,
        listId: list_id,
        ean_code: product.ean_code,
        name: product.name,
        amount: amount,
        bought: bought
    }
  end

  def buy
    set(bought: true)
    save
  end

  def unbuy
    set(bought: false)
    save
  end
end
```

Figura 4.5: Arquivo user.rb. Implementação do modelo Usuário.

```
class User < Sequel::Model
  one_to_many :list_users
  unrestrict_primary_key

  def self.create_or_update(params)
    user = User[params['id']]

    if user.nil?
      user = User.create(:id => params['id'], :name => params['name'])
    elsif user['name'] != params['name']
      user = User.update(:name => params['name'])
    end

    user
  end

  def to_json
    {
        id: id,
        name: name
    }
  end
end
```

Figura 4.6: Arquivo list_user.rb. Implementação do modelo Lista de Usuário.

```ruby
class List_User < Sequel::Model
  many_to_one :list
  many_to_one :user

  def self.add(list_id, user_id)
    list_user = List_User.where(:list_id => list_id, :user_id => user_id)
    if list_user.nil?
      return List_User.create(:list_id => list_id, :user_id => user_id)
    end
    list_user
  end

  def to_json
    {
      list_id: list_id,
      user_id: user_id,
      list: list.to_json
    }
  end

  def user_id_to_json
    user_id.to_s
  end

  def list_to_json
    list.to_json unless !list
  end
end

```

O aplicativo móvel foi desenvolvido para a plataforma Android usando a ferramenta de automação Gradle. Foi usada a IDE Android Studio, para a implementação das funcionalidades para este trabalho.

4.2.1 Biblioteca Facebook SDK

O trabalho prevê o compartilhamento das listas de compras com outras pessoas. Será usado neste trabalho, o compartilhamento via autenticação de usuário integrado com a rede social Facebook. O Facebook foi escolhido devido a sua popularidade. Isso facilita o uso do aplicativo objetivo deste trabalho, uma vez que não só é necessário que o usuário tenha um perfil numa rede social, mas que a pessoa ou pessoas com quem ele compartilha uma lista também tenha. Outra vantagem, é que como o seu uso é bem disseminado, a implementação do aplicativo é facilitada pela grande comunidade de desenvolvedores. O compartilhamento via rede social Google Plus também foi cogitado, porém houve dificuldades técnicas quanto a sua implementação.

O Facebook provê uma biblioteca para facilitar a integração com a rede social. Ela é chamada de Facebook SDK. A versão usada para o trabalho proposta é a "Facebook SDK for Android"versão 4.0. As funcionalidades usadas dessa bilbioteca são:

Facebook Login: Habilita autenticação com as credenciais do Facebook.

Graph API: API que disponibiliza os dados de usuário como nome, endereço de e-mail e lista de amigos. Os dados são fornecidos em forma de grafo, onde cada usuário é um nodo. O fornecimento segue os critérios da política de privacidade: somente serão fornecidos os dados que sejam permitidos para tal pelo usuário. A versão 4.0 do Facebook SDK contém a versão 2.3 desta API.

Segundo Cross (2014), a partir da versão 2.0 da Graph API, a políticas de privacidade do Facebook sofreu alteração de tal modo que somente é possível poder compartilhar uma lista de compras com um usuário que já tenha instalado o aplicativo de listas de compras e tenha aprovado o acesso ao Facebook.

A abordagem para a autenticação foi através do uso do controle de botão de login do Facebook. Este possui *callbacks*, chamadas de função, que facilitam a implementação da autenticação pelo desenvolvedor. Nesse momento, o desenvolvedor também deve definir quais permissões deseja pedir que usuário conceda ao aplicativo, como a necessária Lista de Amigos para o nosso trabalho.

4.2.2 Biblioteca ZBar

A leitura ótica está presente nesse trabalho proposto em duas funcionalidades: identificação do código de barras de um produto em mãos, e a leitura do código QR impresso, quando presente, em uma nota fiscal de mercado.

Para facilitar a implementação dessas leituras, há algumas bibliotecas específicas para tal presentes no mercado. A biblioteca ZBar foi escolhida pela facilidade de implementação, boa precisão, e transparência para o desenvolvedor ao lidar com a leitura de diferentes tipos de códigos de barras.

A biblioteca ZBar é um projeto de código abert (*open source*), e interpreta os mais populares tipos de códigos de barras através de leitura ótica, entre eles, os formatos EAN-13 e o código QR, necessários para este trabalho.

4.2.3 Estruturas de Classes e Pacotes

Foram adicionados alguns elementos ao protótipo do aplicativo Android já existente. A Figura 4.7 ilustra a estrutura de classes do projeto do aplicativo em Android. As classes destacadas foram adicionadas nesse trabalho. A maioria das outras classes já existentes sofreram modificações.

O código se divide em seis categorias de classes:

Atividades - As classes de Atividades estão na pasta raiz. Foram criadas as classes CameraPreview e QRBarcodeReadActivity para capturas de códigos QR e de barras com a câmera do dispositivo, a classe ShareFriendsActivity para implementar a interface da tela de compartilhamento, e a classe MainActivity para implementar a interface da tela inicial do sistema para autenticação pelo Facebook.

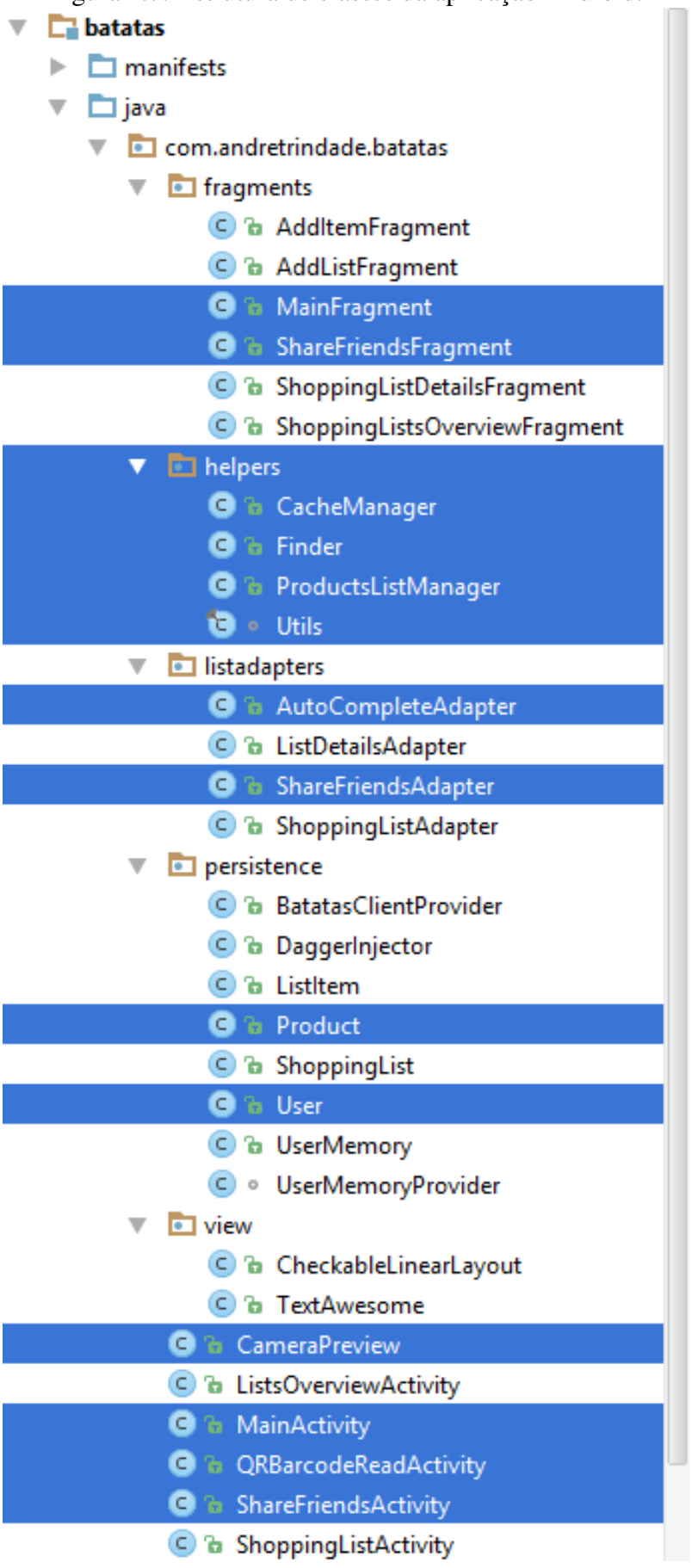
As classes ListsOverviewActivity e ShoppingListActivity, para implementação das telas de Listas de Compras e Lista de Itens, respectivamente, foram mantidas com modificações.

Fragmentos - As classes de Fragmentos estão dentro do diretório *fragments*. Foram adicionados os Fragmentos para compartilhamento (ShareFriendsFragment) e tela inicial (MainFragment). Foram modificadas as classes para listagem e operações de criação, leitura, atualização e deleção (CRUD) de listas de compras e produtos.

Persistência - As classes responsáveis pela lógica de dados e comunicação com o servidor Web RESTful estão no diretório *persistence*. Além de modificações nas classes já existentes, como a BatatasClientProvider - responsável pela interação com o servidor. Foram criadas as classes User, para implementação do controle de usuários, e Product, para implementação do reconhecimento de produtos com recursos de auto completar.

Personalização de Interface - Dois pacotes de personalização de interface foram mantidos sem nenhuma modificação. Um é responsável pelos ícones nas telas do aplicativo, e outro é responsável por permitir seleções múltiplas de listas (ABEGG, 2014).

Figura 4.7: Estrutura de classes da aplicação Android.



Adaptadores de listas - Segundo Abegg (2014, p.39), essas classes "funcionam como pontes entre as listas apresentadas ao usuário e os dados apresentados por estas listas". Foram adicionadas novas classes para as listas de compartilhamento e o autocomplemento na adição de produtos.

Auxiliares - Além dos tipos de classes anteriores, já existentes no trabalho de Abegg, foi criada uma nova categoria de classes para auxiliar a realização de tarefas e requisitos não-funcionais.

As classes CacheManager, Finder e Utils implementam uma *cache* de produtos armazenada no formato *CSV* e gerenciada na memória do aplicativo em Tabela Hash. A classe CacheManager é responsável pela comunicação com o resto do aplicativo. A classe Utils é responsável por leituras e escritas na *cache* em arquivo *CSV* e na Tabela Hash. A classe Finder é responsável por encontrar produtos na *cache* em memória ou até mesmo na base de dados do servidor.

Por fim, a classe ProductsListManager é responsável pela leitura dos produtos de uma NF-e, disponibilizada pela Secretaria da Fazenda em seu site, através de da biblioteca JSoup, que acessa páginas na internet, analisa o conteúdo HTML e retorna-o em uma estrutura de dados para a aplicação. Além disso, essa classe é responsável também por melhorias na *cache* e na base de conhecimento dos produtos no servidor.

4.2.4 Leitura de Códigos de Barras e Notas Fiscais Eletrônicas

As leituras dos códigos de barras e notas fiscais eletrônicas são uma das principais funcionalidades do projeto e objetivo deste trabalho. Para obter resultados satisfatórios, foi usado a biblioteca ZBar para facilitar a leitura ótica.

Um dos diferenciais dessa biblioteca, é o fato de ela oferecer total transparência nas leituras, trazendo para si, a tarefa de identificar qual o tipo e formato do código a ser lido. Isto fornece maior flexibilidade para a implementação e simplicidade para o código. Prova disso, é que foi necessário criar somente uma Atividade para leituras dos códigos de barras, nem precisando informar parâmetros para sua chamada.

4.2.5 Integração do aplicativo cliente com o Servidor

A comunicação entre cliente e servidor é realizada através do protocolo HTTP. Segundo Abegg, "optou-se por utilizar uma biblioteca que torna a implementação de um cliente para serviços REST bastante simples. O uso de Retrofit permite que uma API REST seja representada através de uma simples Interface JAVA."(2014, p. 39).

Como observado na Figura 4.8, essa Interface pertence a classe BatatasClientProvider, que define também o endereço da API REST. Na Interface, são usadas anotações Java nos

Figura 4.8: Arquivo BatatasClientProvider.java.

```java
public class BatatasClientProvider {
    private static final String BATATAS_STATING_ENDPOINT = "http://batatas-alfa.herokuapp.com";

    public interface BatatasClient {
        @GET("/lists/user/{user_id}")
        void getUserLists(@Path("user_id") String userId, Callback<List<ShoppingList>> callback);

        @POST("/lists/user/{user_id}")
        void addUserList(@Path("user_id") String userId, @Body ShoppingList list,
                         Callback<ShoppingList> callback);

        @GET("/lists/{list_id}")
        void getList(@Path("list_id") Long listId, Callback<ShoppingList> callback);

        @DELETE("/lists/{list_id}")
        void deleteList(@Path("list_id") Long listId, Callback<Void> callback);

        @GET("/lists/{list_id}/users")
        void getSharedFriendsIdList(@Path("list_id") Long listId, Callback<List<String>> callback);

        @POST("/lists/{list_id}/user")
        void shareListFriend(@Path("list_id") Long listId, @Body User user, Callback<String> callback);

        @DELETE("/lists/{list_id}/user/{user_id}")
        void unshareListFriend(@Path("list_id") Long listId, @Path("user_id") String userId,
                               Callback<Void> callback);

        @POST("/lists/{list_id}/items")
        void addItem(@Path("list_id") Long listId, @Body ListItem items, Callback<ListItem> callback);

        @DELETE("/lists/{list_id}/items/{item_id}")
        void deleteItem(@Path("list_id") Long listId, @Path("item_id") Long itemId, Callback<Void> callback);

        @POST("/lists/{list_id}/items/{item_id}/bought")
        void buyItem(@Path("list_id") Long listId, @Path("item_id") Long itemId, Callback<ListItem> callback);

        @DELETE("/lists/{list_id}/items/{item_id}/bought")
        void unbuyItem(@Path("list_id") Long listId, @Path("item_id") Long itemId,
                       Callback<ListItem> callback);

        @POST("/users/create")
        User createUser(@Body User user);

        @POST("/products")
        void updateProductsDb(@Body Collection<ListItem> items, Callback<Void> callback);

        @GET("/products/eancodes/{eancodes}")
        List<Product> getProducts_byEanCodes(@Path("eancodes") String eanCodes);
    }
```

métodos a serem invocados pela aplicação Android para identificar o caminho relativo de um recurso REST e o método HTTP correspondentes.

A definição do método HTTP e caminho a ser usado para uma operação no servidor é realizada por anotações Java. Foram adicionados métodos correspondentes aos criados no servidor Web RESTful, como os relativos ao controle de usuários, compartilhamento de listas e operações com a base de produtos. Além disso, foram modificados métodos já existentes, como a obtenção de listas de compras por usuários.

5 FUNCIONAMENTO DO APLICATIVO

Esta seção apresenta as telas e formulários do aplicativo desenvolvido.

5.1 Tela Login

Por decisão de implementação, somente é possível usar o aplicativo após autenticar-se na rede social do Facebook para usufruir das funcionalidades de compartilhamento de listas. Deste modo, a primeira tela do sistema é esta tela de autenticação, a qual contém dois passos: apresentação do sistema com um botão para o usuário explicitamente autenticar-se e a autenticação em si no Facebook, concedendo permissões ao aplicativo.

A autenticação gera uma chave de acesso, ou *token*, para o aplicativo e gerenciada pelo biblioteca usada do Facebook. Após a autenticação, o usuário é redirecionado para a próxima tela, a tela de Listas de Compras. Este comportamento, só acontece da primeira vez que o usuário executar o aplicativo. A partir da próxima vez que o aplicativo ser inicializado, o *token* irá garantir o acesso diretamente a tela de Listas de Compras. A exceção acontece quando o usuário encerra a sessão do Facebook pelo aplicativo ou revoga a autenticação integrada do aplicativo pela rede social Facebook.

A tela de Login é apresentada na Figura 5.1.

5.2 Tela de Listas de Compras

A principal tela do sistema enumera as listas de compras criadas pelo usuário ou compartilhadas com ele. Foi adicionada uma opção para criar listas de compras a partir da leitura de nota fiscal eletrônica. Outras funcionalidades da tela são a criação de uma nova lista e atualização das listas do usuário, acessíveis por botões na barra superior. Esta tela é apresentada na Figura 5.2.

5.3 Tela de Leituras Óticas

O aplicativo oferece a mesma tela para leitura de códigos de barras e para leitura de notas fiscais. A leitura é realizada de forma automática e não é preciso que o usuário

Figura 5.1: Tela de login do aplicativo.

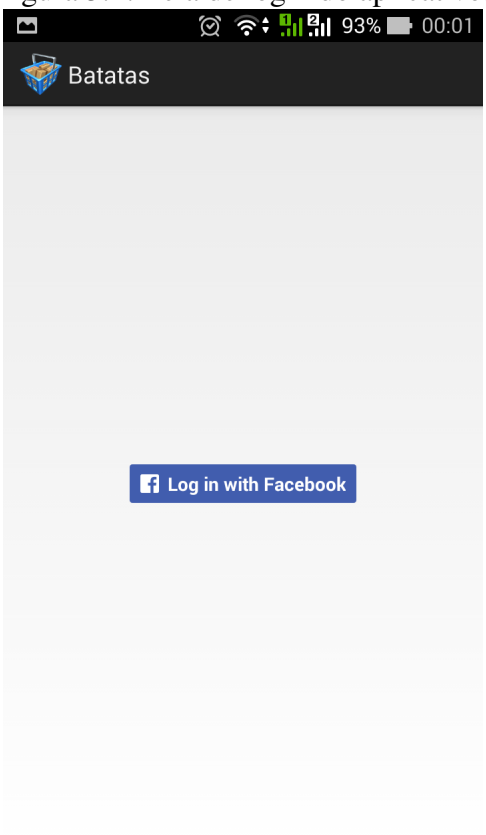


Figura 5.2: Tela de Listas de compras do aplicativo.

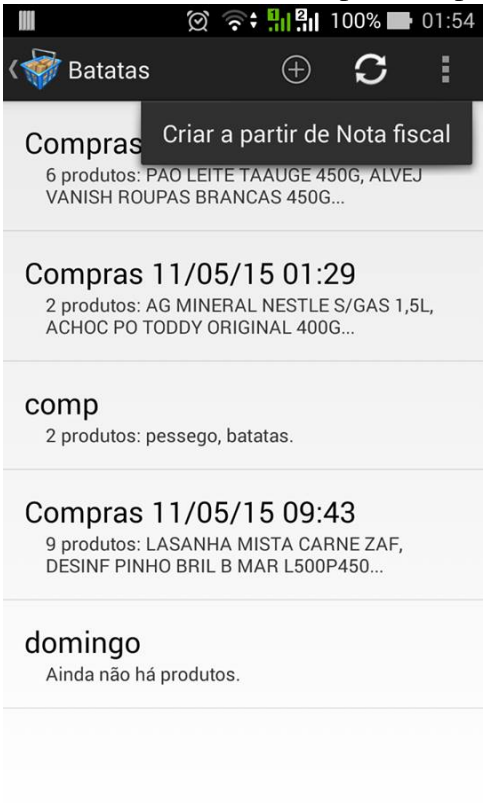


Figura 5.3: Tela de Leituras Óticas.

realize qualquer ação. No momento em que o aplicativo lê com sucesso o código, a tela é encerrada e retorna para a tela anterior.

Essa tela é mostrada ao usuário para as seguintes funcionalidades:

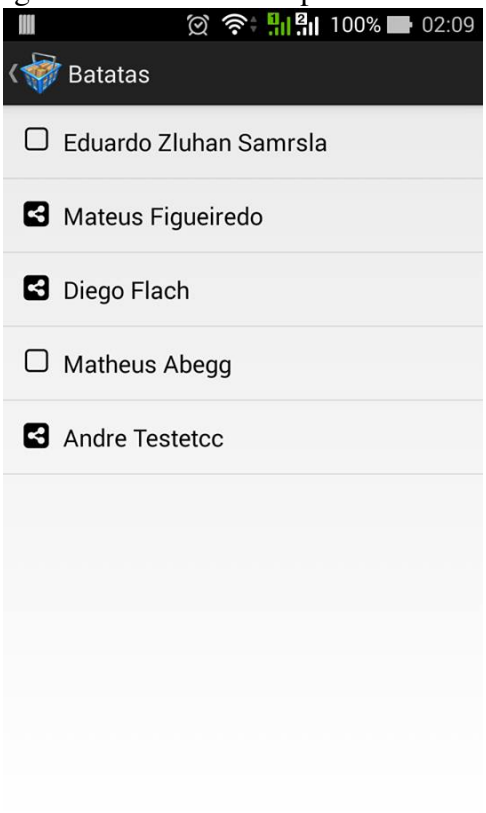
- Criar uma lista de compras a partir da leitura de uma nota fiscal, na tela de listas de compras;
- Selecionar um item adquirido pela leitura do código de barras, na tela de uma lista de compras;
- Selecionar um ou mais itens adquiridos pela leitura de nota fiscal, na tela de uma lista de compras;
- Preencher o nome de um item pela leitura do código de barras, na tela de uma lista de compras, ao adicionar um item.

A tela de Leituras Óticas é apresentada na Figura 5.3.

5.4 Tela de Compartilhamentos

A tela de Compartilhamentos é apresentada na Figura 5.4. Ela lista todos os amigos da rede social que usam o aplicativo, isto é, já o instalaram e autenticaram-se pela rede.

Figura 5.4: Tela de Compartilhamentos.



Os usuários com quem a lista foi compartilhada são marcados com o ícone de compartilhamento, os restantes têm o ícone vazado.

5.5 Requisitos do sistema

É requerido os requisitos mínimos apresentados a seguir para o funcionamento do aplicativo desenvolvido para o trabalho proposto.

Dispositivo móvel: O aplicativo somente pode ser executado em dispositivos móveis, atendendo ao foco na mobilidade ao realizar compras.

Android 4.1 "Jelly Beans": O aplicativo foi implementado para a plataforma Android tendo como requisito mínimo a API versão 16 do *framework* Android SDK. Isso implica que o dispositivo móvel esteja rodando o sistema Android com versão 4.1 ou maior.

Câmera: Para as funcionalidades de leitura ótica de notas fiscais e códigos de barra dos produtos, é preciso que o dispositivo móvel esteja equipado com uma câmera fotográfica. De modo geral, a leitura ótica tem melhor resultado com câmera com maiores resoluções, boa qualidade de captura, como boa densidade de pixels por

polegada (dpi), conferindo maior nitidez à imagem. Além disso, a versão 5.0 introduziu sensíveis melhorias no sistema tendo, entre outras, conferindo melhor desempenho da câmera.

Facebook: Para a funcionalidade de compartilhamento de listas de compras com outras pessoas, é essencial que o usuário tenha um perfil na rede social Facebook para a realização da autenticação integrada. O aplicativo só permitirá acesso às suas funcionalidades após o usuário realizar a autenticação, assim como prover todas as permissões pedidas pelo aplicativo à rede social.

Notas fiscais eletrônicas (NF-e): A leitura de notas fiscais de mercado somente será possível caso a NF-e também tenha sido emitida. Caso afirmativo, haverá o código QR impresso na parte inferior na nota fiscal física. A existência das NF-es é relativamente recente e está ainda sendo introduzida pelos mercados no atual momento. Um ponto desfavorável a expansão do seu uso é, o devido ao maior aperto fiscal decorrente do uso da NF-e, justamente sendo esse uma das razões da implementação dessa tecnologia pela Receita Federal.

6 ANÁLISE DOS RESULTADOS OBTIDOS

Como observado na Tabela 6.1, este trabalho introduz ao protótipo pré-existente o controle de usuários através da integração com o Facebook. Essas duas funcionalidades tornam de fato o protótipo em um aplicativo usável como gerenciador de listas de compras colaborativas. A integração do Facebook, apesar de presente nos aplicativos semelhantes, tem um nível de profundidade maior. Como o controle de usuários é vinculado ao Facebook, o compartilhamento de listas é possível com um amigo no Facebook do usuário autenticado a partir do momento em que esse amigo se autentica pelo aplicativo. Tal funcionalidade não é verificada em nenhum dos aplicativos semelhantes, onde a integração com o Facebook é somente realizada para autenticação no aplicativo.

Outra funcionalidade introduzida neste trabalho foi a leitura de código de barras presente dois dos outros quatro aplicativos semelhantes. Além disso, a funcionalidade que realmente diferencia esse trabalho de outros aplicativos é a integração com as Notas Fiscais Eletrônicas (NF-e). Em nenhum outro aplicativo estudado foi encontrada essa funcionalidade. Apesar de ser implementada somente visando notas fiscais do estado do Rio Grande do Sul, é oferecida uma contribuição importante para a expansão do uso em nível nacional. Além disso, é possível extrair outras informações de uma NF-e, como data da compra, identificação do mercado, endereço, e até o valor de tributos estaduais e federais.

6.1 Análise das avaliações dos usuários

O aplicativo implementado foi distribuído entre alguns usuários para experimentar suas funcionalidades. Assim é possível traçar a receptividade do público através de uma amostra, além de identificar *bugs* e possíveis melhorias de funcionalidades.

Devido a restrições de tecnologia, como incompatibilidade sistemas operacionais e versões do Android, somente quatro usuários testaram o aplicativo com sucesso. Os usuários têm perfis parecidos: jovens da faixa etária dos 25 aos 30 anos e aficionados por tecnologia. Deste modo, a amostra tem um certo vício e pode ser melhorada com a inclusão de testadores de faixa etárias e culturais mais amplas, para representar mais fielmente os usuários finais do aplicativo.

Tabela 6.1: Comparação das funcionalidades presentes nos aplicativos semelhantes com este trabalho

Funcionalidades/Aplicativo	Shopping List	Meu Carrinho	Out of Milk	Batatas (ABEGG, 2014)	Este trabalho
Múltiplas Listas de compras	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Controle de Usuários	**Sim**	**Sim**	**Sim**	Não	**Sim**
Edição colaborativa de listas	Não	**Sim**	Não	**Sim**	**Sim**
Compartilhamento de listas	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Categorização de Itens	**Sim**	Não	Não	Não	Não
Suporte a múltipla plataformas	Não	**Sim**	Não	Não	Não
Registro de Preço	Não	**Sim**	**Sim**	Não	**Sim**
Leitura de Código de Barras	Não	**Sim**	**Sim**	Não	**Sim**
Leitura de Notas fiscais (Nf-e)	Não	Não	Não	Não	**Sim**
Integração com Facebook	Não	**Sim**	**Sim**	Não	**Sim**
Código Aberto	Não	Não	Não	**Sim**	**Sim**

Após, isso foi realizado um questionário anônimo via internet para que esses usuários informassem suas experiências de uso do aplicativo e suas funcionalidades. Os resultados serão demonstrados e analisados aqui.

A Figura 6.1 apresenta os resultados da avaliação das funcionalidades que envolvem as notas fiscais. A recepção foi boa, mas como uma leve preferência pela criação de listas de compras a partir de notas fiscais do que pela seleção de produtos adquiridos através da leitura de nota fiscal.

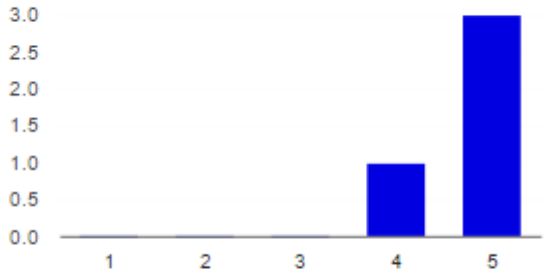
Uma possível causa desse resultado é que a tarefa de seleção de produtos envolve maior complexidade de código, e depende da existência dos produtos na base de dados com a informação do código de barras ou de heurística de reconhecimento dos produtos. Outra causa pode ser a menor aderência devido ao fato de a seleção ser feita após uma compra, e não durante. Além disso, com listas não muito longas, a tarefa de selecionar os produtos adquiridos é trivial.

Com menor receptividade, o resultado da avaliação da funcionalidade de seleção de produtos por código de barras está mostrado na Figura 6.2. Com avaliação pendendo para o meio termo entre uma experiência Ruim e Excelente, provavelmente devido ao fato de que a seleção de produto por código de barras precise de mais cliques e o tempo a mais para reconhecimento ótico do código de barras. Além disso, um produto não necessariamente terá informação de código de barras.

Como pode ser observado na Figura 6.3, as funcionalidades mais bem recebidas pelos avaliadores foram com relação ao Facebook e compartilhamento de listas. A integração com o Facebook foi recebida muito bem por todos os usuários por ser simples e eficaz, o foco da aplicação não é perdido com autenticação e cuidados em lidar com usuários. Do mesmo modo, o compartilhamento de lista de compras com os amigos do Facebook que

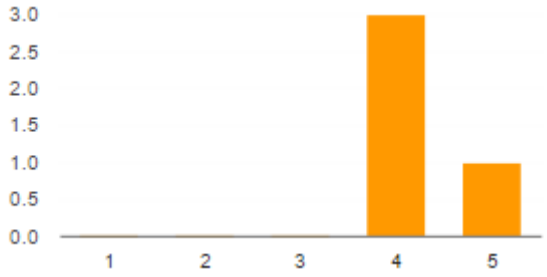
Figura 6.1: Resultado das avaliações das funcionalidades relativas às Notas Fiscais.

Criação de listas de compras a partir de notas fiscais:



Ruim: 1	0	0%
2	0	0%
3	0	0%
4	1	25%
Excelente: 5	3	75%

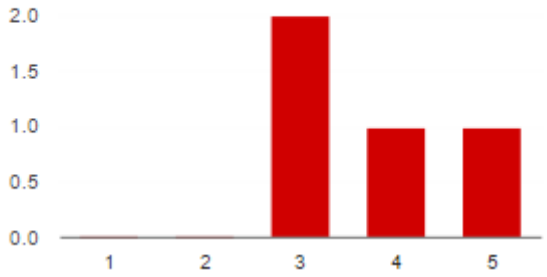
Seleção de produtos adquiridos através leitura de nota fiscal:



Ruim: 1	0	0%
2	0	0%
3	0	0%
4	3	75%
Excelente: 5	1	25%

Figura 6.2: Resultado das avaliações das funcionalidades relativas aos Códigos de Barras.

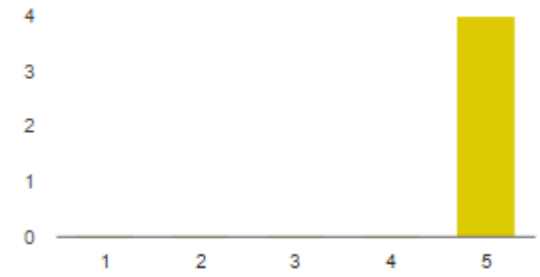
Seleção de produtos adquiridos através leitura do código de barras de um produto:



Ruim: 1	0	0%
2	0	0%
3	2	50%
4	1	25%
Excelente: 5	1	25%

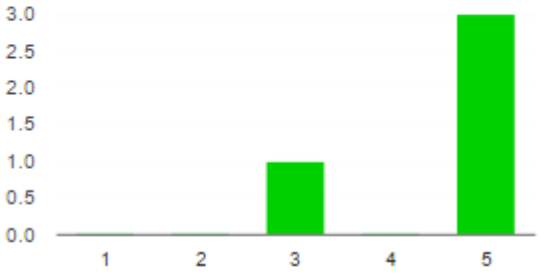
Figura 6.3: Resultado das avaliações das funcionalidades de integração com o Facebook e Compartilhamento de listas.

Integração com o Facebook:



Ruim: 1	**0**	0%
2	**0**	0%
3	**0**	0%
4	**0**	0%
Excelente: 5	**4**	100%

Compartilhamento da lista com amigos do Facebook:



Ruim: 1	**0**	0%
2	**0**	0%
3	**1**	25%
4	**0**	0%
Excelente: 5	**3**	75%

usam o aplicativo também foi bem recebido.

7 CONCLUSÃO

Este trabalho apresentou novas funcionalidades para um protótipo de gerenciamento compartilhado de listas de compras visando novas tecnologias, como as Notas fiscais eletrônicas a fim de facilitar a vida cotidiana das pessoas. Além disso, foi implementado um controle de usuário com a rede social Facebook com o compartilhamento de listas aos amigos usuário do mesmo aplicativo.

O servidor possui código aberto, sendo possível implementar clientes para outras plataformas usando a API REST em comum. O código está disponível no endereço https://github.com/andretf/batatas. Também é possível adicionar novas funcionalidades, aprofundar as existentes e até mesmo corrigir *bugs*.

É importante citar que não foi possível implementar a principal funcionalidade originalmente pensada para este trabalho. A proposta era de ler qualquer nota fiscal, independentemente de ser NF-e, para criação de listas de compras. A leitura seria pelo Reconhecimento Ótico de Caracteres (OCR). Para isso, um protótipo foi implementado usando a biblioteca OCR Tess-Two, derivada da biblioteca Tesseract, criada por engenheiros da Google, mas abandonada desde 2012.

O motivo pela impossibilidade da implementação dessa funcionalidade foi a baixa taxa de precisão do reconhecimento de produtos na leitura de notas fiscais, mesmo com operações de melhorias de imagem para OCR integradas na biblioteca, provavelmente devido a falta de contraste das notas fiscais estudadas, qualidade da impressão e fonte de texto não facilmente reconhecível. Segundo Booth (2006), o desempenho do reconhecimento ótico é proporcional definição de imagem, i.e., pontos por polegadas, precisando de idealmente de 300ppp . Assim, seria possível obter melhores resultado com redimensionamento de imagens, porém envolve custo de memória e processamento.

Algumas outras ideias de funcionalidades também foram pensadas durante as fases iniciais de desenvolvimento deste trabalho. Porém elas foram preteridas em razão do autor não ter tido familiaridade com o desenvolvimento para a plataforma Android até a implementação deste trabalho. Estas funcionalidades serão apresentadas a seguir.

7.1 Trabalhos Futuros

O trabalho oferece margem para a implementação de novas funcionalidades. Focando em treinamento de dados para leitura OCR e melhoria de imagens, será possível implementar uma leitura universal de notas fiscais. Podem ser usadas outras bibliotecas OCR ou mesmo aplicativos externos para leitura de códigos de barras.

Além disso, é possível extrair mais informações de uma NF-e do que somente a lista de produtos. Tendo o nome do mercado, endereço e local da compra é possível que tenha um mapa marcado com mercados e, ao selecionar um mercado, é listado os produtos com preços e o usuário que informou o preço mais atual. Também é possível deixar o aplicativo em formato de jogo, a chamada *gameficação*, através de um sistema de pontos.

Outra funcionalidade a ser pensada é a autenticação integrada pelo Google. Notificações de atualizações de preços para produtos preferidos ou inclusos em listas de compras do usuário podem ser implementadas.

Pode haver uma melhoria com relação aos produtos. Os itens podem ser categorizados em vários níveis. Por exemplo, um item arroz branco da uma certa marca pode ser reconhecido simplesmente como "arroz"e produto alimentício.

Finalmente, um cliente web pode conter, além das mesmas funcionalidades dos aplicativos móveis, funcionalidades extras, como um gerenciador de controle de gastos.

REFERÊNCIAS

ABEGG, M. **Desenvolvimento de um Protótipo de Solução Colaborativa para o Controle de Listas de Compras**. 2014. Trabalho Individual — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

AMUNDSEN, M. **REST - The short version**. Disponível em: <http://exyus.com/articles/rest-the-short-version/>. Acesso em: junho 2015.

BARRAS, S. C. de. **Guia de Códigos de Barras para Iniciantes**. Disponível em: <http://codigodebarrasean.com/>. Acesso em: junho 2015.

BOOTH, J. M. **Optimizing OCR Accuracy on Older Documents**: a study of scan mode, file enhancement, and software products. Disponível em: <http://www.gpo.gov/pdfs/fdsys-info/documents/WhitePaper-OptimizingOCRAccuracy.pdf>. Acesso em: junho 2015.

CONSULTANTS, B. **Mobile Technology** : software development for the mobile market. Disponível em: <http://www.bbconsult.co.uk/Expertise/Mobile-Applications/Mobile-Technology>. Acesso em: maio 2015.

CROSS, S. **Facebook Graph Api v2.0+ - /me/friends returns empty, or only friends who also use my app**. Disponível em: <http://stackoverflow.com/a/23417628/986862>. Acesso em: junho 2015.

FAZENDA, M. da. **Nota Fiscal Eletrônica**. Disponível em: <http://www.nfe.fazenda.gov.br/portal/sobreNFe.aspx?tipoConteudo=HaV+iXy7HdM=>. Acesso em: junho 2015.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado) — University of California, Irvine.

GOOGLE. **Managing the Activity Lifecycle**. Disponível em: <https://developer.android.com/training/basics/activity-lifecycle/index.html>. Acesso em: outubro 2014.

GOOGLE. **Play Store**. Disponível em: <https://play.google.com/store>. Acesso em: outubro 2014.

GS1. **About GS1**. Disponível em:
<http://www.gs1.org/about>. Acesso em: junho 2015.

GS1. **GS1 Standards**. Disponível em:
<http://www.gs1.org/standards>. Acesso em: junho 2015.

MACALÃO, P. R. **Check in Poa: um aplicativo android para turistas em porto alegre**. 2013. Trabalho Individual — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

PRASS, R. **Entenda o que são os 'QR Codes', códigos lidos pelos celulares**. Disponível em:
<http://g1.globo.com/tecnologia/noticia/2011/05/entenda-o-que-sao-os-qr-codes-codigos-lidos-pelos-celulares.html>. Acesso em: junho 2015.

SILVA, T. R. D. **PyRester**: uma abordagem baseada em modelos u2tp para geração de código de teste unitário para restful web services. 2011. Trabalho Individual — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

SOUZA, M. V. **SACI: sistema de apoio à coleta de informações**. 2014. Trabalho Individual — Instituto Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

TEAM, A. **Android Anatomy and Physiology**. Disponível em:
<https://code.google.com/p/androidteam/>. Acesso em: junho 2015.

TELEFONICA. **Telefonica Global Millennial Survey - 2014 global results presentation**. Disponível em:
<http://www.slideshare.net/TelefonicaEurope/telefonica-2014-gms-global-master-deck-final-100614>. Acesso em: maio 2015.

VNI, C. **Cisco Visual Networking Index**: global mobile data traffic forecast update 2014–2019 white paper. Disponível em:
<http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html>. Acesso em: maio 2015.