# PRODUCT HUNTER: UM *FRAMEWORK* PARA DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS COM FOCO NA BUSCA E AVALIAÇÃO DE PRODUTOS DE FORMA COLABORATIVA

LUIS VICTOR COUTINHO MENEZES

# PRODUCT HUNTER: UM *FRAMEWORK* PARA DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS COM FOCO NA BUSCA E AVALIAÇÃO DE PRODUTOS DE FORMA COLABORATIVA

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Ciências Exatas da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador: Eduardo Freire Nakamura

Manaus

Maio de 2014

Menezes, Luis
A524p Product Hunter: Um *framework* para Desenvolvimento de Aplicações Móveis com foco na busca e avaliação de produtos de forma colaborativa / Luis Victor Coutinho Menezes. — Manaus, 2014
viii, 86 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal do Amazonas
Orientador: Eduardo Freire Nakamura

mobile computing, framework, crowdsensing, product rating

CDU 1234

# [Folha de Aprovação]

Quando a secretaria do Curso fornecer esta folha,
ela deve ser digitalizada e armazenada no disco em formato gráfico.

Se você estiver usando o `pdflatex`,
armazene o arquivo preferencialmente em formato PNG
(o formato JPEG é pior neste caso).

Se você estiver usando o `latex` (não o `pdflatex`),
terá que converter o arquivo gráfico para o formato EPS.

Em seguida, acrescente a opção `approval={`*nome do arquivo*`}`
ao comando `\ppgccufmg`.

Se a imagem da folha de aprovação precisar ser ajustada, use:
`approval=[`*ajuste*`][`*escala*`]{`*nome do arquivo*`}`
onde *ajuste* é uma distância para deslocar a imagem para baixo
e *escala* é um fator de escala para a imagem. Por exemplo:
`approval=[-2cm][0.9]{`*nome do arquivo*`}`
desloca a imagem 2cm para cima e a escala em 90%.

# Resumo

A cada dia dispositivos móveis são aperfeiçoados, ganham novas funcionalidades, novos sensores, hardware melhor e, com isso, maior capacidade de medir informações do meio em que são utilizados e de processá-las de forma correta. Além disso, as tarefas do dia-a-dia vêm sendo automatizadas. Com base nisso, a tendência de que, no futuro, dispositivos móveis terão a capacidade de realizar medições e substituírem a necessidade de implantação de uma infra-estrutura fixa responsável pela coleta de determinada informação, tarefas simples - como a verificação de preços - poderão ser extinta.

Com base nisso, a área de sensoriamento participativo e a aplicação de técnicas de colaboração em sistemas vêm ganhando foco e, principalmente, aprimorando a coleta de informações. Diversas são as aplicações colaborativas que informam aos seus usuários o que estes estão querendo saber e, com isso, criam uma comunidade colaborativa, capaz de ajudar com coletas e receber informações através de outros usuários.

No entanto, apesar de parecidas em sua base, não existem ferramentas capazes de auxiliar o desenvolvimento destas aplicações de forma mais fácil; reduzindo tempo e esforço; poupando recursos para o foco em melhor desenvolvimento e acabamento de suas aplicações; possibilitando a criação de melhores formas de incentivo aos usuários e, consequentemente, melhores coletas e uma comunidade ativa por mais tempo.

Por estes motivos, esta dissertação propõe um *framework* capaz de auxiliar no desenvolvimento de sistemas que integram os diversos conceitos mostrados acima, com integração em redes sociais e auxílio no desenvolvimento da aplicação. Para isso, foram levantados requisitos importantes às aplicações desta área a partir dos quais a arquitetura e projeto da ferramenta foi criada. Além disso, o trabalho apresenta um protótipo de uma aplicação utilizando o *framework*, com a finalidade de servir como prova de conceito do mesmo.

**Palavras-Chave:** mobile computing, framework, crowdsensing, product rating

# Lista de Figuras

# Lista de Tabelas

# Sumário

# Lista de Abreviações

**URI** Uniform resource identifier

**OHA** Open Handset Alliance

**PHP** Hypertext Preprocessor

**MVC** Model-View-Controller

**HTTP** Hypertext Transfer Protocol

**REST** Representational State Transfer

**W3C** World Wide Web Consortium

**HTML** HyperText Markup Language

**MOSDEN** Mobile Sensor Data Engine

**CDAS** Crowd-sourcing data analytics system

**AMT** Amazon Mechanical Turk

**SQL** Structured Query Language

**ORMLite** Object Relational Mapping Lite

**SOAP** Simple Object Access Protocol

**CAROMM** Context-Aware Real-time Open Mobile Miner

**API** Application Programming Interface

**SGBD** Sistema de Gerenciamento de Banco de Dados

**VGA** Video Graphics Array

**ART** Android Runtime

**JSON** JavaScript Object Notation

**JAR** Java Archive

# Capítulo 1

# Introdução

Atualmente, os dispositivos móveis vêm se tornando os dispositivos de computação e comunicação central na vida das pessoas (Jayaraman et al., 2013). O mercado de dispositivos só cresce, tendendo a superar o número de seres humanos na Terra em meados de 2017 (Lilly, 2013). Juntamente com estes dispositivos, novas aplicações surgem seguindo tendências do uso da Internet. A todo momento, surgem novas formas de interação entre usuários, portais de compra, canais de pesquisa de preço e possibilidades de se informar através de aplicações móveis. Seguindo as tendências, encontradas em Yang (2011), o desenvolvimento de plataformas orientadas à aplicações móveis é uma dos grandes tópicos da economia da Internet móvel. Seguindo as aplicações móveis, observa-se uma série de características

## 1.1 Dispositivos Móveis

Os dispositivos móveis, aproveitando-se da mobilidade ocasionada pela separação do acesso à informação através de cabos e do avanço tecnológico, que proporcionou avanços na composição de hardware e, consequentemente, maior poder de processamento e de execução de tarefas, têm se tornado uma central de computação e comunicação onipresente na vida das pessoas (Lane et al., 2010; Jayaraman et al., 2013). A difusão de dispositivos móveis e a criação de uma categoria de computação, denominada Computação Móvel, caracterizada como a representação de um novo paradigma computacional, que permite o acesso a serviços através do ambiente, independentemente de sua localização (Figueiredo & Nakamura, 2003).

A importância dos dispositivos móveis nos negócios atuais é representativa (Haghirian et al., 2005). De acordo com Lee & Benbasat (2003), os defensores das tecnologias móveis dizem que o crescimento e a importância do comércio móvel vai su-

perar a importância dos e-commerces. No entanto, os dispositivos móveis modificaram, também, a forma com que os usuários interagem e a forma com a qual a informação referente a produtos se difunde, independentemente da localização e posição do usuário (Haghirian & Inoue, 2007).

Os números referentes ao avanço da tecnologia e do crescimento do número de dispositivos móveis presentes no mercado demonstram a necessidade social proveniente da evolução dos dispositivos e o crescimento da importância destes para a população. Atualmente, a área de dispositivos móveis, mais especificamente a de vendas, possui sua influência na economia global caracterizada pelo aumento constante no número de venda de smartphones, apresentando um crescimento de aproximadamente 78% em 2012, em relação à 2011 (Moretti, 2013). Com um aumento de 171%, ou 2 milhões a mais de unidades vendidas, o número de tablets também representou um ganho com relação ao ano anterior. Baseado nestes dados, calcula-se um total aproximado de 16 milhões de smartphones e 3 milhões de tablets, totalizando 19 milhões de dispositivos móveis vendidos em um ano, comparando com 6,7 milhões de desktops e 8 milhões de notebooks (14,7 milhões no total). Estes dados refletem o aumento da intenção do usuário de ser mais móvel, mais livre. E estes valores se tornam ainda mais óbvios se considerarmos a existência de notebooks no contexto dos dispositivos móveis. Estes dados podem ser melhor visualizados na tabela 1.1, onde se resume os valores apresentados.

| Categoria | Dispositivo | Unidades Vendidas |
|---|---|---|
| Dispositivos Móveis | Smartphones | 16.000.000 |
| Dispositivos Móveis | Tablets | 3.000.000 |
| Computadores / Dispositivos Móveis | Notebooks | 8.000.000 |
| Computadores | Desktops | 6.700.000 |

**Tabela 1.1.** Números referente às vendas de dispositivos móveis e computadores em 2012. (Moretti, 2013)

De acordo com estimativas de empresas de consultoria nos EUA, como dito em eMarketer (2013b), por exemplo, os gastos mundiais na área de dispositivos móveis atingiram cerca de 8,41 bilhões de dólares em 2012, contra cerca de 4 bilhões em 2011 e apenas 2,34 bilhões no ano de 2010. Sendo que na América Latina, este valor representa no Brasil cerca de 90% contra a média continental de 86%. Já segundo Gartner (2013), em estudo relacionado aos gastos das empresas brasileiras, indica que um valor de 134 bilhões de dólares estava previsto para ser gasto com TI no ano, totalizando 24,3 bilhões de dólares, pertencentes ao total, para o setor que inclui PCs,

tablets e celulares.

Além de todo o investimento e gastos com dispositivos móveis, um estudo, liberado pela[1], diz que no ano de 2016 serão cerca de 10 bilhões de aparelhos móveis conectados à rede (smartphones, tablets, computadores portáteis), ao passo que um valor estimado de apenas 7,3 bilhões de pessoas no mundo, segundo estimativas das Nações Unidas. Já para o ano de 2017, as previsões são ainda mais contundentes. Segundo Lilly (2013), o número de tablets, combinado ao de smartphones, ultrapassará o número de habitantes no planeta. Atualmente, estimativas apontam que existem um total de 14 bilhões de dispositivos conectados à Internet, sendo a grande maioria de computadores e notebooks, o que aumenta a estimativa, já que os 16 bilhões estimados pela Cisco são apenas para Smartphones e Tablets, desconsiderando o restante.

A partir destes dados, diversas empresas vêm investindo fortemente em formas de criar aplicações sem um conhecimento na na área de programação. Um exemplo destes casos é o Google, que criou o App Inventor para desenvolvimento de aplicações móveis para Android, destinado àqueles que não possuem conhecimentos de programação, como apresentado em Abelson et al. (2010). Esta ferramenta tem como objetivo o desenvolvimento e a exploração de aplicações móveis por parte daqueles que não possuem tanto conhecimento técnico para a implementação de sistemas como estes.

## 1.2 Aplicações Móveis

Com o crescimento do setor de dispositivos móveis, surgiram as aplicações móveis, que se baseiam na execução de tarefas em dispositivos móveis. Este campo de desenvolvimento presenciou uma rápida evolução nas últimas duas décadas. No passado, o desenvolvimento e o controle de dispositivos móveis era feito apenas por empresas fabricantes dos dispositivos e pelas companhias de telecomunicações (Mahmoud & Popowicz, 2010). A evolução do hardware dos dispositivos móveis (Duan et al., 2011), trouxe à realidade novas possibilidades para o desenvolvimento de aplicação. Desta forma, o que se tem agora são 4 atores responsáveis pela criação de aplicação móveis: a operadora de telecomunicações, o fabricante do dispositivo, o provedor de serviços e o desenvolvedor de software, segundo Mahmoud & Popowicz (2010). Ainda assim, o desenvolvimento de aplicações móveis enfrenta alguns desafios no decorrer do caminho (Unhelkar & Murugesan, 2010), dentre eles, os problemas ocasionados pela mobilidade dos dispositivos, como a necessidade da criação de aplicações eficientes na questão consumo de bateria, como tratado por Duan et al. (2011).

[1] www.cisco.com.br

## 1.3 Sistemas Colaborativos

A mobilidade, a interconectividade e a colaboração dos usuários são conceitos que têm grande interseção e que se fortalecem com o aprimoramento de tecnologias e na mudança das formas de interação do usuário para com a Internet. Segundo Eagle (2011), os smartphones têm potencial para gerar uma quantidade de dados sem precedentes. Este fato, agregado à utilização de mídias sociais e a construção coletiva do conhecimento, fortalecem todos os conceitos aos quais estas características se conectam.

Os sistemas colaborativos, e suas várias nomenclaturas, representam uma quebra no paradigma, aproximando-se do conceito de criação coletiva do conhecimento e da informação, removendo um objeto centralizador do papel de difusor do conhecimento coletivo. Estas características representam a necessidade real do usuário, que utiliza o sistema, de socializar e transmitir conhecimento, assim como a recíproca é verdadeira, ajudando em vários fatores, incluindo tomada de decisão e conhecimento e informações atualizadas.

Se analisarmos pequenas e médias empresas que fazem uso de sistemas colaborativos como parte de suas ferramentas de trabalho, notaremos que estas ferramentas são utilizadas para aumentar a produtividade através da troca de informações e também do aumento da relação entre seus funcionários, melhorando a realização de tarefas e, também, a produtividade geral, já que os funcionários passam a ser incluídos em decisões e informações da empresa.

Saindo um pouco do ambiente empresarial, notamos que, de forma prática, os sistemas colaborativos estabelecem pontos de encontro na web, para as pessoas interagirem como um time ao passo que criam, gerenciam, acessam e compartilham informações relacionadas ao seu contexto de trabalho. Além de promoverem a rápida localização de informações no âmbito corporativo, favorecem a interação entre pessoas e processos de negócios por meio de fluxos de trabalho e formulários eletrônicos.

Com a evolução da interação entre usuários na Internet, a tendência de que sistemas colaborativos tomem conta de boa parcela da informação é grande, basta notar que grande parte deles utiliza ferramentas que trazem mobilidade, interconectividade e colaboração para o sistema. Estes fatores são perfeitamente relacionados à utilização de dispositivos móveis.

## 1.4 Objetivos

### 1.4.1 Objetivo Geral

Este projeto tem como objetivo a elaboração de um *framework*, modelado com o intuito de prover eficiência e rapidez no desenvolvimento de aplicações móveis que utilizem conceitos de busca de informações e avaliação de produtos através da coleta proativa de dados por parte de usuários presentes em comunidades colaborativas.

### 1.4.2 Objetivos Específicos

Para se alcançar o objetivo geral deste trabalho, observa-se a necessidade de atender, primeiramente, a alguns objetivos específicos que necessitam de atenção no processo até o estágio final do *framework*.

**Caracterizar os Frameworks Existentes** - Explorar os *frameworks* existentes e, a partir de uma pesquisa, determinar quais as formas de implementação são mais importantes e necessárias para a criação de um *framework* capaz de prover eficiência e rapidez no desenvolvimento de aplicações.

**Caracterizar Necessidades das Aplicações da Área** - Explorar as aplicações já existentes e separar suas principais funcionalidades.

**Criar o *Framework*** - Separar as características presentes no passo anterior e reuní-los na criação do *framework*.

**Análise e Correção de Defeitos** - Analisar e corrigir defeitos ocasionados na utilização do *framework*.

**Desenvolver Prova de Conceito : *DiapersHunter*** - Implementar um caso de uso que servirá como prova de conceito da ferramenta. Este estudo de caso será um sistema colaborativo para a exibição, ranqueamento, informações e opiniões de uma comunidade de usuários sobre determinado produto, que neste caso específico, é fralda.

## 1.5 Organização da Proposta

O capítulo 2 apresenta a fundamentação teórica que deu origem às escolhas e definições que o *framework* implementa. No capítulo 3, apresentamos os trabalhos relacionados, dos quais retiramos parte das ideias e definições para a construção da ferramenta. No

capítulo 4, apresentamos o *Product Hunter Framework*, o produto final deste trabalho. No capítulo 5, apresenta-se o *DiapersHunter*, um estudo de caso de implementação de uma aplicação colaborativa com base no *framework* que originou este trabalho. No capítulo 6, apresenta-se as considerações finais, as limitações e, por fim, os trabalhos futuros.

# Capítulo 2

# Fundamentos e Ferramentas

Este capítulo apresenta, além dos conceitos necessários para a compreensão e elaboração deste trabalho, as ferramentas utilizadas para possibilitar obter uma ferramenta final, capaz de oferecer os benefícios requeridos.

## 2.1 Sistemas Móveis e a Mobilidade

De acordo com B'far (2004), a mobilidade está diretamente ligada com Sistemas Computacionais Móveis, que são sistemas que podem facilmente ser movidos fisicamente ou cujas capacidades podem ser utilizadas enquanto estes estão em movimento. Desta forma, o conceito de mobilidade se define como a possibilidade de mover determinado sistema computacional sem a necessidade de se estar atrelado à fios que o obriguem a executar tal função em estado estacionário. A mobilidade e os sistemas que se aproveitam desta característica trazem recursos e características que não podem ser encontradas em sistemas comuns, como:

- Preocupações com os gastos de energia;
- Aplicações Móveis; e
- Acesso à informação de forma mais fácil

A utilização de Dispositivos móveis, conceituado na seção 2.2 possibilita a utilização do máximo da mobilidade que um usuário pode ter.

## 2.2 Dispositivos Móveis

Os dispositivos móveis são dispositivos capazes de executar sistemas computacionais na palma das mãos, sendo chamados de *handhelds* em inglês, por fazer referência à sua utilização na palma das mãos. Eles possuem, normalmente, uma tela, uma ferramenta capaz de trazer a possibilidade de entrar com dados manualmente. Utilizam-se da Mobilidade, que é a possibilidade de mover determinado sistema computacional sem a necessidade de estar conectado à fios, para transformar dispositivos comuns em verdadeiros computadores portáteis. Sua função é, basicamente, trazer para o usuário a possibilidade de executar tarefas de forma fácil e prática, onde quer que estes estejam. Os dispositivos móveis dividem-se, geralmente, em *smartphones*, *PDAs*, *notebooks*, Computadores Portáteis e tudo aquilo que possa executar tarefas computacionais na palma das mãos. Alguns dos dispositivos móveis mais utilizados atualmente são os smartphones e os tablets, como mostrado na figura 2.1. Smartphones e Tablets são definidos a seguir:

– **Smartphone.** Um celular comum com funcionalidades avançadas, que podem ser extendidas por meio de programas (aplicativos) executados por meio de seu sistema operacional. Os sistemas operacionais dos smartphones são abertos, do ponto de vista de desenvolvimento, ou seja, qualquer pessoa pode desenvolver aplicativos que podem funcionar nestes aparelhos.

– **Tablets.** Dispositivo em formato de prancheta que pode ser utilizado para acesso à internet, organização, produtividade, visualização de fotos e vídeos, entretenimento e acesso à informação de um modo geral. Apresenta uma tela sensível ao toque, servindo como dispositivo principal para a entrada de dados.

**Figura 2.1.** Dispositivos Móveis mais utilizados na atualidade.

Algumas de suas características são responsáveis por impulsionar o desenvolvimento do segmento e, consequentemente, da tecnologia empregada neles. Características como: a mobilidade, a execução de multi-tarefas, novas possibilidades de facilitar a comunicação e, também, novas formas de entretenimento trouxeram ao público facilidades que anteriormente não eram tão acessíveis, mas atualmente são indispensáveis.

Mesmo com seu grande poder de facilitar a vida dos usuários, diversas são as limitações a serem administradas no momento da utilização destes dispositivos, são elas:

- **Tamanho de Tela Reduzido.** Os dispositivos possuem uma tela pequena para muitas das tarefas, constituíndo um problema para as aplicações, que precisam exibir o conteúdo de forma compacta e que facilite sua visualização por parte do usuário.

- **Menor capacidade de Processamento e Armazenamento.** Possuem uma menor capacidade de processamento e de armazenamento comparado à computadores mais robustos. No entanto, a tendência é que estes dispositivos ganhem, cada vez mais, em desempenho, possibilitando a execução de tarefas mais complexas.

- **Limitação Energética.** Os dispositivos possuem uma limitação quando o assunto é bateria. Mesmo com toda a evolução dos outros componentes de hardware, a bateria e sua real utilização se manteve com um nível de crescimento abaixo do esperado em comparação com os outros componentes. Por este motivo, é necessário ter um cuidado diferenciado com a utilização de aplicações que consumam este recurso de forma demasiada.

## 2.3 Computação Móvel

A computação móvel é uma categoria da computação que une a processamento, mobilidade e comunicação em busca de caracterizar um novo paradigma computacional (Figueiredo & Nakamura, 2003), que permite o acesso à serviços diversos independentemente da localização e da forma de acesso. Surge como a quarta evolução da computação, antecedida pelos centros de processamento de dados da década de 60, o surgimento dos terminais nos anos setenta e as redes de computadores na década de 80. Este paradigma é, basicamente definido pela mobilidade.

Aproveita-se da melhoria de hardware trazida com o tempo para os dispositivos móveis, bem como de sua mobilidade. Possibilita a execução de um número cada vez

maior de tarefas e utiliza formas de comunicação como o Wi-Fi, 3g ou o Bluetooth, o que traz maior mobilidade à execução de tarefas. No entanto, essa comunicação é, em certos casos, atrapalhada por uma largura de banda de comunicação reduzida em relação àquela cabeada e a uma maior taxa de erros na comunicação, devido à fragilidade das redes móveis.

A computação móvel necessita de uma infra-estrutura que pode ser separada entre Interna e/ou Externa. A interna refere-se a áreas locais, em ambiente fechado e as tecnologias utilizadas são, basicamente o Wi-Fi (802.11), o Infra-Vermelho e o Bluetooth (Figueiredo & Nakamura, 2003). A infra-estrutura externa refere-se à locais onde não há uma conexão direta com um roteador, em uma empresa ou em um domicílio. As áreas metropolitanas ou globais aproveitam-se de tecnologias como as Redes Móveis, Wi-Max e Satélites, basicamente.

## 2.4 *Crowdsensing*

O *crowdsensing* é o processo em que indivíduos, dotados de sensores em seus equipamentos, colaborem com um sistema alimentando uma base de dados única que pode ser utilizada por outros indivíduos para realizar pesquisas ou obter informações (Burke et al., 2006). É um paradigma emergente que ativa a coleta distribuída de informações por participantes auto-gerenciáveis. Esta coleta permite um número crescente de interação entre os usuários de forma a prover uma maior quantidade de informações relevantes para uma dada comunidade, como afirma Cristofaro & Soriente (2013).

O termo em questão é dado ao conceito de uma comunidade de elementos contribuindo com informações de sensores em prol de uma pesquisa atividade específica. Ganhou grande importância devido à necessidade de avaliar a qualidade de vida e a situação do meio ambiente, pela falta de infra-estrutura natural e dificuldade de acesso a algumas regiões, tornando-se necessário procurar maneiras alternativas de se coletar dados que não sejam com sensores estáticos adicionados em campos de testes sem mobilidade e a um custo relativamente alto. Na figura 2.2, pode-se visualizar um exemplo de uso de um sistema de *crowdsensing*, onde vários dispositivos móveis estão empenhados em coletar informações e enviá-las para um servidor que será responsável por armazenar as coletas para uso futuro.

Como observado em (Christin et al., 2011), a presença de sensores nos dispositivos móveis atuais possibilita a criação de novos tipos de aplicações móveis capazes de mensurar diversas informações à respeito do ambiente. Com isso, é natural que apli-

cações colaborativas surjam e, apoiando-se no *crowdsensing*, passem a se difundir no mercado, criando novas comunidades. Cada usuário pode estar colaborando com um sistema da forma que mais lhe for conveniente, aumentando a capacidade de coleta de informações, reduzindo custos e trazendo benefícios para os usuários que da atividade de coleta participam.

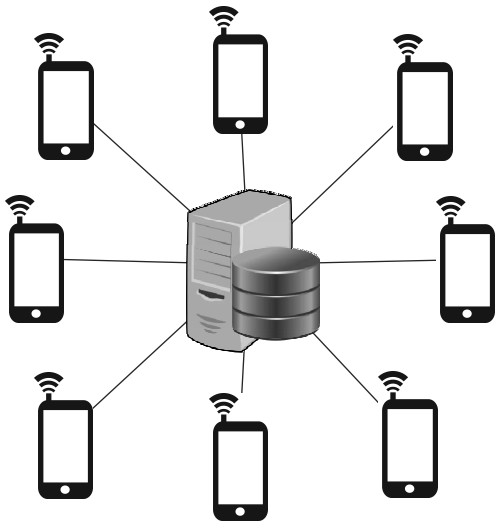

**Figura 2.2.** Interação entre dispositivos móveis e um servidor centralizado.

## 2.5 *Framework*

Segundo Johnson & Foote (1988), um *framework* é um conjunto de classes que constituem um design abstrato para soluções de uma família de problemas. Segundo Mattsson (1996), um *framework* é uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização. Para Buschmann et al. (1996) e Pree (1995), um *framework* é definido como um software parcialmente completo projetado para ser instanciado. O *framework* define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Apesar de diferentes em certos pontos, estas definições não são contraditórias. Sendo assim, um *framework* pode ser definido como uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o *framework* quem dita o fluxo de controle da aplicação.

## 2.6 Incentivos

Esta seção trata de algumas formas de incentivo à participação em aplicações de sistemas colaborativos. Sem colaboração não há comunidade colaborativa e, é através deste foco, que as formas de incentivo passaram a ser estudadas mais profundamente. Em levantamentos prévios sobre formas de incentivo, encontramos, além das formas já mencionadas nas seções 3.2.1.3 e 3.2.2.3, algumas outras, de grande valor para uma aplicação desta categoria. Estes incentivos devem ser implementados e podem ser considerados o carro-chefe do crescimento da comunidade. Basicamente, o que as formas de incentivo fazem é mirar no tema de interesse do usuário e, assim, aumentar sua motivação em troca de recompensas que o incentivem a continuar contribuindo.

Dentre os projetos analisados nesta área, uma coisa é certa: o importante é o Retorno do Investimento, como considera (Lee & Hoh, 2010). Esta é uma variável que representa uma simples equação: investimento - benefício, valor que representa o pensamento do usuário em quanto ele precisou se esforçar para cumprir as tarefas e o quanto ele ganhará em troca. A variável referente ao investimento é constituída pelo tempo e esforço que o usuário investiu para realizar sua coleta, bem como gastos de bateria do dispositivo utilizado, deslocamento e, em alguns casos, a privacidade, como abordado na seção 3.2.

Diversas são as variedades de forma de incentivo, como veremos a seguir, mas precisam possuir relevância suficiente no contexto da aplicação para serem bem sucedidas. Trabalhos como o descrito por Deng & Cox (2009) retratam uma forma de incentivo, de certa forma, imperceptível, onde é retratado o uso de uma aplicação onde os usuários informam preços de produtos e recebem informações atualizadas sobre os preços coletados por outros usuários, trazendo como informações adicionais como, por exemplo, os estabelecimentos onde estes se encontram. O incentivo, neste caso, consiste na obtenção de informações privilegiadas pelo simples motivo de contribuir com a aplicação de forma ativa.

Por outro lado, existem trabalhos que citam formas de incentivo mais diretas, com ganhos monetários, inclusive. Do ponto de vista dos provedores de conteúdo que coletam e utilizam dados coletados, o incentivo monetário tem grandes chances de aumentar a participação do usuário, podendo atrair grandes quantias de participantes e incrementar não só a quantidade de coletas, mas também sua qualidade (Krontiris & Albers, 2012). Sistemas que implementam este tipo de recompensa podem ser encontrados como, por exemplo, em Jaimes et al. (2012), onde o sistema tende a pagar por coletas com preços mais baixos, fazendo uma espécie de leilão de coletas inverso, onde o preço mais baixo ganha. No trabalho de Lee & Hoh (2010), se pro-

põe e se avalia um modelo de incentivo baseado em um sistema de leilão com coletas com preços dinâmicos. Nele, é sugerido que participação do usuário é o elemento mais importante para que a aplicação obtenha sucesso. Sendo assim, o modelo de incentivo foca em minimizar e estabilizar o custo de incentivo enquanto mantém, até onde pode, um número adequado de participantes contribuintes, prevenindo desistências. O trabalho compara, ainda, o seu mecanismo de incentivo com um outro, denominado Seleção Randômica com preço Fixado (Random Selection with Fixed Prices (RSFP)), obtendo como resultado a informação de que o novo modelo não apenas reduz o custo de incentivo, mas também aumenta o senso de justiça na distribuição das recompensas e do bem-estar social da comunidade, bem como o Retorno do Investimento atrelado à cada coleta e à campanha de sensoriamento. A adoção da abordagem dinâmica de preços deve-se à possibilidade de adequar os preços ao mercado de forma mais eficaz. Com a dinamicidade dos preços, é normal que alguns dos coletores acabem por desistir da coleta, aumentando os preços pagos devido à falta de oferta. No caso da desistência, o sistema busca superar o desafio aplicando formas de crédito virtual à seus usuários, como forma de incentivo à participações futuras, aumentando, com isso, sua possibilidade de ganhar. Holzbauer et al. (2012) utiliza um mecanismo reverso de leilão para tratar tanto a manutenção de usuários ativos quanto o contexto social advindo da participação. Os usuários dão lances sempre que o provedor do serviço precisa de novas coletas. Após a finalização dos lances de todos os usuários que desejarem vender suas coletas, o provedor escolherá quais dos mesmos receberá o valor do lance. Este trabalho faz referência, ainda, a alguns outros mecanismos presentes na literatura, como por exemplo:

- **First Price Auctions.** Um mecanismo onde todos os participantes estão que estão competindo por incentivo fazem lances simultâneamente para prover uma de N coletas requeridas pelo dono da campanha. O servidor, então, esperará até que todos os concorrentes tenham finalizado seus lances e, após isso, escolhe as N coletas mais baratas e as compra. No entanto, segundo Holzbauer et al. (2012), este tipo de mecanismo torna difícil o entendimento por parte do usuário, considerando que, a partir do momento em que os vencedores recebem pelo valor de seus lances, os lances não refletem seus reais custos e o retorno do investimento não compensa, inviabilizando este mecanismo.
- **RADP-VPC.** Este mecanismo é uma modificação do **First Price Auctions**, tendo em mente o conceito de leilão reverso em mente (Lee & Hoh, 2010). Em cada round consecutivo em que o participante perder, seu lance correspondente sofre um decrescimo baseado em uma constante $\alpha$. O provedor ignora o valor de

αcaso o lance do usuário seja um dos vencedores, sem reduzir o valor a ser ganho com a venda. Esta mudança faz com que o usuário passe a ter mais chances de vencer o leilão, sem que para isso, precise receber menos. Para fazer com que o sistema seja mais compreensível para o usuário, é possível ser notificado sobre o maior lance pago para os usuários.

– **Participation Incentive Generalized Vickrey Auction (PI-GVA).** Este mecanismo é, na verdade, uma alternativa para o **First Price Audio**, pois computa um placar para cada participante, que encapsula o número atual e o esperado de vezes que o usuário se saiu vencedor do leilão, bem como os lances correspondentes Lee & Szymanski (2009). O parâmetro μ, calculado pelo mecanismo, determina o peso da média dos lances. O provedor utiliza, então, o placar para calcular os preços que serão escolhidos como os vencedores. Segundo o autor, esta é um mecanismo compatível com incentivo, pois ajuda o usuário a entender suas chances de vitória e o processo executado na escolha dos vencedores.

Holzbauer et al. (2012) propõe, então, uma mudança no **First Price Auction** na forma com a qual este escolhe os vencedores. O mecanismo, denominado "Privacy, Power and Participation aware Auction Mechanism (P3AM)"e possui um novo parâmetro, $P_{cheapest}$, que determina quantos vencedores serão escolhidos baseado nos menores lances. O restante dos vencedores será escolhido baseando-se nos que possuírem menos Retorno de Investimento até então. Este mecanismo encoraja os usuários que estão com risco de pararem de coletar informações a continuar contribuindo.

## 2.7 Ferramentas

### 2.7.1 Android

O Android[1] é um sistema operacional para dispositivos móveis baseado no núcleo do Linux, desenvolvido pela Open Handset Alliance (OHA), liderada pelo Google Inc. A OHA é uma aliança de várias empresas com a intenção de criar padrões abertos para a telefonia móvel. Entre as empresas participantes estão o Google, a HTC, a Dell, a Intel, a Motorola, a Qualcomm, a Samsung, a LG, a T-Mobile, a Nvidia, entre outras. O total de empresas participantes da aliança é de aproximadamente 87 empresas, verificado em fevereiro de 2014, segundo a OHA. Por ser desenvolvido por uma aliança tão grande, o sistema operacional está disponível nos gadgets de diversos fabricantes de dispositivos

[1] www.android.com

móveis e, segundo o Google, em 2011, a cada dia, meio milhão de novos aparelhos eram ativados com o sistema.

As especificações técnicas do sistema são diversas e dependentes da versão do sistema. No entanto, no que diz respeito à parte visual das aplicações, a plataforma é adaptada tanto para dispositivos Video Graphics Array (VGA) maiores, gráficos 2D, bibliotecas gráficas 3D baseadas em OpenGL ES especificação 2.0 e os layouts mais tradicionais dos smartphones. Com relação ao armazenamento das informações, utiliza-se o SQLite, que é uma biblioteca que implementa um motor de base de dados auto-contido, sem configuração e transacional. Programas que usam a biblioteca SQLite podem ter acesso ao banco de dados sem executar um processo específico do Sistema de Gerenciamento de Banco de Dados (SGBD), possibilitando que as aplicações android rodem sem a dependência de um processo de banco de dados separado consumindo memória do sistema.

As aplicações escritas em Java são compiladas em bytecodes Dalvik e executadas usando a Máquina Virtual Dalvik, que é a máquina virtual especializada apenas no uso em dispositivos móveis como tablets e celulares. Isso permite que os programas sejam desenvolvidos e distribuídos em formato binário (bytecode) para serem executados em qualquer dispositivo com o sistema operacional Android, sem a necessidade de se saber qual o processador está sendo utilizado no dispositivo. Atualmente, uma nova máquina virtual está sendo desenvolvida para otimizar o carregamento dos aplicativos. Esta nova máquina virtual se chama android Android Runtime (ART) e está disponível apenas no Android 4.4 e superiores.

No que diz respeito às proporções globais da adoção de sistemas operacionais em dispositivos móveis, o Android é o sistema operacional móvel mais utilizado no mundo, segundo sbicheno (2013), da StrategyAnalytics[2], embora a Apple[3] descorde desta informação. Segundo a empresa, a plataforma móvel do Google lidera o mercado com 81%, seguido pelo iOS, com 13.4% e pelo Windows Phone, com 4.1%.

Atualmente, as versões presentes na tabela 2.1 são as que foram anunciados pelo Google até o presente momento.

### 2.7.2 CakePHP

O CakePHP é um *framework* que utiliza a linguagem de programação PHP e tem como principais objetivos oferecer uma estrutura que possibilite aos programadores da linguagem desenvolverem aplicações robustas de forma rápida, sem a perda da

[2] www.strategyanalytics.com
[3] www.apple.com

| Versão | Nome | Data de Lançamento |
|---|---|---|
| 1.0 | Astro | setembro de 2008 |
| 1.5 | Cupcake | abril de 2009 |
| 1.6 | Donut | setembro de 2009 |
| 2.0 | Eclair | outubro de 2009 |
| 2.1 | Eclair | janeiro de 2010 |
| 2.2 | Froyo | maio de 2010 |
| 2.3 | Gingerbread | dezembro de 2010 |
| 3.0 | Honeycomb | fevereiro de 2011 |
| 4.0 | Ice Cream Sandwich | outubro de 2011 |
| 4.1 | Jelly Bean | julho de 2012 |
| 4.2 | Jelly Bean | novembro de 2012 |
| 4.3 | Jelly Bean | julho de 2013 |
| 4.4 | KitKat | outubro de 2013 |

**Tabela 2.1.** Versões lançadas do Android até fevereiro/2014.

flexibilidade que o Hypertext Preprocessor (PHP) proporciona a seus usuários. Segue as convenções da arquitetura Model-View-Controller (MVC), modelado após o conceito do Ruby on Rails e distribuído sob a licença MIT.

O *framework* usa conceitos bem conhecidos de engenharia de software e padrões de projeto, como formas de convenção, MVC, Active Record, Association Data Mapping e Front Controller. O projeto foi iniciado no ano de 2005, quando o programador Polonês Michael Tatarynowicz escreveu uma versão mínima de um *framework* em PHP. Disponibilizando classes e objetos adicionais com o objetivo de proporcionar maior extensibilidade e reuso, para que os desenvolvedores possam adicionar funcionalidades à base da arquitetura de suas aplicações. Algumas destas extensões são Extensões do Modelo, Extensões do Visão, Extensões do Controlador.

### 2.7.3 Hypertext Transfer Protocol (HTTP)

A World Wide Web Consortium (W3C) define o Hypertext Transfer Protocol (HTTP) como um protocolo de nível de aplicação para sistemas de hipermídia, colaborativos e distribuídos. É um protocolo genérico, sem estado e orientado a objetos que pode ser usado para diversas tarefas, tais como servidores de nomes e sistemas de gerenciamento de objetos distribuídos, através da extensão de seus métodos de requisição.

O HTTP é caracterizado por ser um protocolo sem estados, pois não há o armazenamento de informações sobre o estado atual entre as requisições feitas ao servidor. Desta forma, caso um servidor receba várias requisições ao mesmo tempo, não será pos-

sível separá-las por cliente de forma a saber como uma requisição influencia na outra. Portanto, é impossível desenvolver uma aplicação, por exemplo, de comércio eletrônico utilizando apenas o protocolo HTTP, outros recursos são utilizados para manter as informações como sessões e cookies.


**Figura 2.3.** Interação entre cliente e servidor.

A figura 2.3 demonstra o processo de requisição e recebimento da resposta. O cliente envia uma requisição ao servidor que, assim que processa, responde ao cliente com a informação requisitada processada. A tabela 2.2 também demonstra este comportamento através de uma requisição ao servidor utilizando telnet.

| Comando | Resposta | Função |
|---|---|---|
| telnet www.google.com 80 | Trying 173.194.37.18...<br>Connected to uol.com.br.<br>Escape character is '^]'. | Inicia o **telnet** e avisa-o para conectar no servidor "www.google.com" na porta 80. |
| GET / HTTP/1.1 | <HEADER><br><HTML> | Recebe como resposta as variáveis presentes no cabeçalho da requisição acompanhadas do código HTML da página requisitada e o código de status da requisição. |

**Tabela 2.2.** Exemplo de requisição HTTP ao servidor do Google.

O exemplo da tabela 2.2 fornece uma solicitação HTTP e uma resposta parcial de um servidor da web. Neste exemplo, iniciamos a solicitação com o telnet especificando o nome de domínio do servidor da web (neste caso, o servidor do Google[4]) e a porta (80). O Telnet responde primeiro com uma resolução de Sistema de Nomes de Domínio (DNS) do nome de domínio para um endereço IP, e então, indica que estou conectado ao servidor. Feito isso, especificamos uma linha de solicitação (que contém o método HTTP GET, um caminho para o recurso no servidor e o protocolo a ser utilizado

[4]www.google.com

(HTTP/1.1). A requisição é seguida, então por uma linha em branco, que indica ao servidor da web que a solicitação está concluída. O servidor fornece uma resposta, indicando o protocolo usado e fornecendo um código de status e cabeçalhos de resposta adicionais. Após isso, o que vemos é a representação do recurso ao qual requisitamos, que em boa parte dos casos é um documento HyperText Markup Language (HTML).

No exemplo acima, apenas o método GET foi exibido. No entanto, existem diversos outros, com características e utilidades diferentes. A tabela 2.3 apresenta uma lista completa dos métodos utilizáveis em uma requisição HTTP.

| Método | Descrição |
|---|---|
| OPTIONS | Requisita informação sobre as opções disponíveis para comunicação. |
| GET | Obtém um recurso requisitado junto ao servidor. |
| HEAD | Obtém apenas metainformações sobre o recurso requisitado junto ao servidor. |
| PUT | Cria ou substitui determinado recurso com uma representação. |
| POST | Aumenta um recurso existente com uma representação. |
| DELETE | Exclui um recurso que o identificador especifica. |

**Tabela 2.3.** Métodos de requisição para o protocolo HTTP/1.1.

#### 2.7.3.1 Representational State Transfer (REST)

Representational State Transfer, ou REST, é um estilo de arquitetura para a comunicação entre cliente e servidor baseada na web, permitindo às aplicações cliente se comunicarem com os servidores de maneira singular (Navón & Fernandez, 2011), como mostra a figura 2.4. Particularmente, REST representa algum recurso presente no servidor, através de uma Uniform resource identifier (URI), que é uma cadeia de caracteres compacta usada para identificar ou denominar um recurso na Internet, tornando a implementação da arquitetura REST mais simples no protocolo HTTP.

No centro da arquitetura RESTful está o conjunto de recursos. Além deste conjunto de recursos, que é a base da arquitetura, temos um conjunto de operações através das quais é possível manipulá-los. De forma mais concreta, podemos definir estes recursos como representações de objetos de dados utilizando uma variedade de tipos, como com JSON, por exemplo. A requisição é feita através de urls, utilizando todo o conjunto de operações do protocolo HTTP conhecidas na seção 2.7.3. A utilização do REST sobre o protocolo HTTP simplifica o desenvolvimento das arquiteturas RESTful, porque utilizam um protocolo bem conhecido e estável, além de estar amplamente disponível e não requerer nenhuma nova configuração.

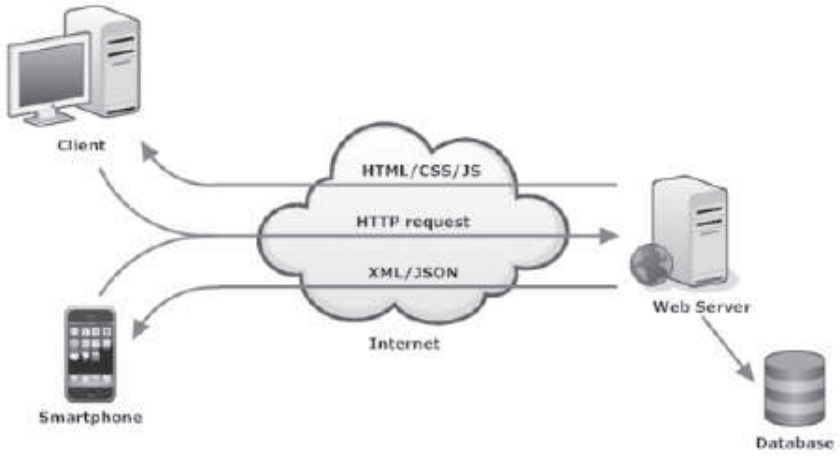


**Figura 2.4.** Exemplo de requisição feita com REST.

Apesar de sua liberdade inquestionável, é necessário seguir um conjunto de características para a implementação de uma arquitetura inteiramente RESTful. Os clientes não acessam os recursos de forma direta, mas, ao invés disso, uma representação para o recurso requisitado. Um aspecto essencial das arquiteturas RESTful é que elas fazem com que o servidor não mantenha nenhum contexto de cliente entre suas transações, sendo necessário que cada transação envie também as informações necessárias para a possibilidade de executar a solicitação em particular. Esta forma de encarar as requisições auxilia a tornar as aplicações mais estáveis, confiáveis e escaláveis.

# Capítulo 3

# Trabalhos Relacionados

Esta seção apresenta os trabalhos relacionados com o projeto deste trabalho. Iniciamos a apresentação dos trabalhos relacionados com alguns *frameworks* que visam, assim como o proposto neste trabalho, apresentar soluções que auxiliem os desenvolvedores a utilizar suas funcionalidades de forma a obter ganho na forma de auxílio à criação de aplicações relacionadas com algum contexto específico. Após a apresentação destes *frameworks*, apresentamos, também, algumas aplicações que serviram como base no sentido de trazer características importantes para aplicações colaborativas que utilizem *crowdsensing*.

## 3.1 *Frameworks*

### 3.1.1 MOSDEN

O Mobile Sensor Data Engine (MOSDEN) é um *framework* que busca facilitar a visualização mais ampla das capacidades do *crowdsensing* através da criação de uma plataforma comum que busca facilitar o desenvolvimento e a implantação de aplicações móveis que se utilizam dos conceitos de *crowdsensing*. Para isso, busca o desenvolvimento de uma plataforma autônoma, escalável, interoperável e que suporte coleta, processamento, armazenamento e compartilhamento dos dados de forma eficiente. Quando o trabalho Jayaraman et al. (2013) refere-se à construção de um sistema autônomo, relaciona o fato de ser possível realizar coletas e armazená-las até que o dispositivo esteja online, sem que estas se percam por questões de fragilidade de conexão. Além disso, a preocupação com questões de consumo de energia e de utilização da banda da conexão propriamente dita é evidente desde a elaboração do esboço do trabalho. De acordo com Jayaraman et al. (2013), a plataforma é capaz de funcionar em uma

variedade de dispositivos com recursos limitados, incluíndo smartphones. O *framework* permite, ainda, que os usuários, ou desenvolvedores, tenham acesso à customização de algoritmos e modelos, dependendo dos requerimentos da aplicação. Como princípio, baseia-se na utilização dos sensores presentes nos

### 3.1.2 CAROMM

O Context-Aware Real-time Open Mobile Miner (CAROMM), proposto por Sherchan et al. (2012), é proposta uma plataforma para suportar coleta de dados para *crowdsensing* a partir dos sensores dos dispositivos móveis utilizando conceitos de data mining para provar sua eficiência e escalabilidade na coleta de dados para aplicações deste tipo. Como meta, o framework planeja reduzir a quantidade de dados enviados, bem como o consumo de energia, provendo um nível de precisão comparável com modelos tradicionais de sensoriamento e envio de coletas.

### 3.1.3 CDAS

O CDAS é um exemplo de *framework* destinado ao *crowdsensing*. A plataforma permite a implantação de várias aplicações destinadas ao *crowdsensing* que necessitam de intervenção humana para a verificação de tarefas simples para permitir uma boa precisão no que diz respeito às coletas. A abordagem tomada pelo trabalho proposto por Liu et al. (2012) trata o problema das coletas em duas fases:

- A campanha, ou a pesquisa, é feita por um computador de alta performance, que processa o resultado e o quebra em subpartes, que são enviadas para humanos para que estes possam avaliar o resultado através da utilização do Amazon Mechanical Turk (AMT)[1].
- A segunda parte começa ao se obter os resultados da utilização do AMT, onde combina-se os resultados das avaliações dos usuários e, através de análises complexas, chega-se ao resultado final.

A figura 3.1 demonstra a organização do sistema, composto, basicamente, pela aplicação do usuário, a lista de tarefas a serem completadas pela plataforma de *crowdsensing*, o computador de alta performance, e os usuários trabalhadores, que farão a verificação das coletas e darão ao sistema o resultado. Analisando estas informações,

[1] http://aws.amazon.com/pt/mturk/

nota-se que a funcionalidade básica do CDAS é a de dar suporte a utilização de determinada funcionalidade, neste caso a utilização de formulários, para a utilização de *crowdsensing* para o resultado das tarefas.

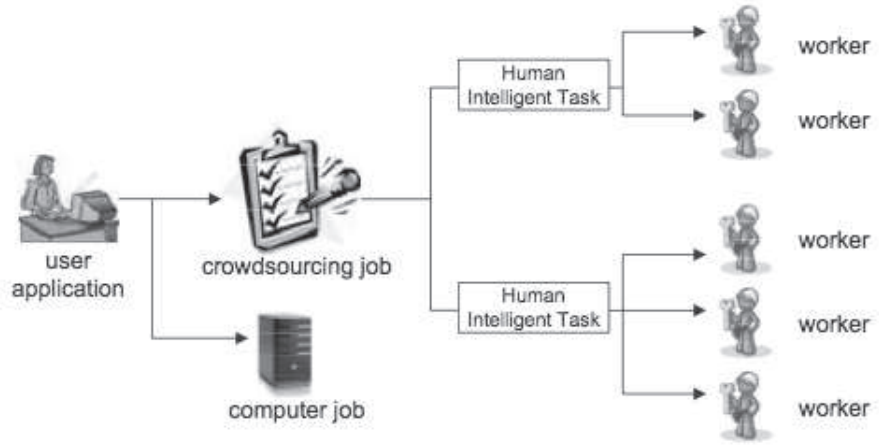


**Figura 3.1.** Exemplo de funcionamento do CDAS. (Liu et al., 2012)

### 3.1.4 Comparação entre os *Frameworks*

A seguir, na tabela 3.1, é possível verificar a comparação entre as características de cada um dos *frameworks* observados por estre trabalho. No geral, vemos algumas diferenças básicas entre eles, no entanto a estrutura básica e a arquitetura remete às funcionalidades necessárias para obter rapidez de desenvolvimento e facilidades para o usuário.

| *Framework* | Preocupação com Energia | Crowdsensing | Aplicações Móveis |
|---|---|---|---|
| MOSDEN | Não | Sim | Sim |
| CAROMM | Sim | Sim | Sim |
| CDAS | Não | Sim | Não |

**Tabela 3.1.** Tabela comparativa entre os *frameworks* relacionados.

No que diz respeito à preocupação com energia, ou seja, a utilização de recursos feita de forma razoável, considerando os limites de bateria de dispositivos móveis, contamos com apenas um (CAROMM) dos *frameworks* referenciando tal questão e buscando resolver o problema de forma a minimizar a utilização dos recursos e, assim, economizando energia desnecessária. Já no que diz respeito à utilização de *crowdsensing* como finalidade da utilização dos *frameworks*, contamos com todos os relacionados

referindo-se à tal conceito. Dos trabalhos relacionados, apenas o CDAS não refere-se diretamente a utilização de dispositivos móveis, como é o tema deste trabalho.

## 3.2 Aplicações

Nesta sessão, são apresentadas, além dos *frameworks* que podem servir como base para a criação de um *framework*, aplicações que utilizem conceitos de sensoriamento participativo e colaboração em computação móvel. A análise e o levantamento de requisitos fará com que as funcionalidades mais necessárias façam parte da ferramenta disponibilizada por este trabalho.

### 3.2.1 Waze

O Waze[2] é um dos maiores aplicativos de trânsito e navegação do mundo baseado em uma comunidade. Com esta aplicação, é possível unir-se a outros motoristas próximos e compartilhar informações de trânsito das vias em tempo real. O Waze foi criado em 2008, com o intuito de trazer a localização de lugares com base no mapeamento das cidades, além da indicação de trajetos baseados nas informações de trânsito, sendo capaz de proporcionar as melhores rotas para o usuário. O aplicativo é capaz de determinar as velocidades médias em cada trecho e com isso calcular as trajetórias mais rápidas tendo, como base, as informações atualizadas do trânsito. Para conseguir isso, os usuários enviam recados, mensagens e, inclusive, diversos alertas. Alguns outros usuários são capazes de editar o mapa, quando for necessário, alterar o preço do combustível nos diversos postos ao longo da cidade. O Waze está disponível para Android [3], iOS [4] e, também, para Windows Phone [5].

#### 3.2.1.1 Serviços do Waze

Como um aplicativo colaborativo, o Waze tem uma série de serviços disponibilizados a seus usuários. Estes serviços estão diretamente relacionados com as suas necessidades, bem como com a possibilidade de coleta das informações. O waze, especificamente, tem serviços de informação de preços de combustível nos postos, trânsito, alertas de policiamento encontrado, acidente em determinado ponto da cidade e, por fim, alertas de perigos em trechos da estrada. Algumas destas informações podem ser visualizadas

[2]https://www.waze.com
[3]https://play.google.com/store/apps/details?id=com.waze&hl=pt_BR
[4]https://itunes.apple.com/br/app/waze-gps-social-e-transito/id323229106?mt=8
[5]http://www.windowsphone.com/pt-br/store/app/waze/f07f83eb-a8a4-49fd-8946-c67a9349e062

na figura 3.2, que representa os alertas que os usuários podem visualizar e enviar para os outros usuários do serviço. Como forma de futura análise destas informações, vamos separar os serviços disponibilizados a seguir:


**Figura 3.2.** Imagem representando a tela de tipos de informações enviadas pelos usuários no Waze.

**Navegação por Rotas.** O Waze disponibiliza formas de navegar entre os pontos por diversas rotas. Baseia-se nas informações de tráfego atualizadas por seus próprios usuários, em tempo real. Normalmente, é possível visualizar tempo, situação do trânsito, distância real e uma imagem, simulando os pontos referentes à problemas no percurso e o momento no qual o usuário passará por cada um deles. Caso o usuário deseje utilizar outro caminho, o Waze é capaz de calcular alternativas para o usuário procurar pela que melhor se encaixe com suas necessidades. Ao selecionar uma rota, o usuário tem detalhações via voz ao longo do caminho e informacões sobre os problemas no percurso, bem como sobre limite de velocidade. Estas informações facilmente se adequam com o local ao qual o usuário se encontra no momento determinado, sendo totalmente possível de o sistema se reajustar em caso de o usuário ir por uma rota não informada pelo sistema, como uma via errada, por exemplo.

**Visibilidade do Usuário.** No sistema, é possível permitir, ou não, que outros usuários visualizem sua real posição no mapa. Esta posição é precisa, dado que o sistema necessita visualizá-la para lhe proporcionar rotas reais. No entanto, diversos problemas

estão relacionados à isso. Muitos usuários têm medo que, por algum motivo, sua segurança esteja comprometida em função do compartilhamento de informações pessoais em sistemas colaborativos. Um dos maiores problemas dos serviços de sensoriamento e colaboração móvel é o de proporcionar ao usuário transparência o suficiente para que este não precise se preocupar com sua segurança. No entanto, algumas aplicações precisam que o usuário informe sua localização para outros usuários, para proporcionar novas formas de ajudá-lo ou para qualquer outro fim no qual esta informação seja crucial. No caso do Waze, a função de visibilidade veio como uma forma de o usuário poder escolher entre informar ou não sua localização, deixando a cargo deste a função de definir sua vontade. Com isso, é possível que usuários que estão próximos ao Wazer (nome que se dá ao usuário do sistema) tenham a capacidade de vê-lo e, com isso, seja possível uma interação entre ambos, e não somente a informação sobre o trânsito ou sobre o que o sistema se pré-dispõe a informar. O tema de visibilidade é abordado em todos os relatórios sobre privacidade do usuário, como, por exemplo, no trabalho de Cornelius et al. (2008).

**Integração com Redes Sociais.** Nesta aplicação, a integração com as redes sociais se dá de forma simples. É possível entrar na aplicação utilizando uma conta no facebook[6], assim como conectar nesta para publicar informações na perfil do usuário. Além do Facebook, o Waze contém também integração com Twitter[7] e Foursquare[8]. A integração com o facebook serve, basicamente, para mostrar à amigos informações do seu perfil, por onde você está dirigindo e para onde você está indo. Com isso, é possível organizar "caronas" para pessoas que estão no seu círculo de amigos. No Twitter, os usuários podem optar por postar automaticamente informações sobre o trânsito em determinado local, assim como o Trânsito Manaus[9] faz, de forma manual, bem como estatísticas de direção do usuário. A integração com o Foursquare possibilita ao usuário fazer check-in (processo de informar aos amigos que o usuário está no local citado) de fora rápida em determinado local. Além disso, com estas integrações, o Waze consegue fazer propaganda de seu aplicativo, ao torná-lo visível em várias redes sociais, utilizadas por milhares de pessoas ao longo do mundo.

**Customização da Interface.** A aplicação possibilita ao usuário visualizar a interface do sistema de diversas formas diferentes. Além do padrão, é possível utilizar um esquema de cores diferentes, além de chavear entre exibição de mapas em 2D e 3D. Ainda

[6] www.facebook.com
[7] www.twitter.com
[8] www.foursquare.com
[9] www.transitomanaus.com.br/

na parte de configurações, é totalmente possível escolher diferentes vozes, linguagens e o leque de possibilidades é bem vasto. Isso permite que o usuário utilize o sistema da melhor forma possível, sentindo-se confortável com a opção que escolher. É possível, inclusive, customizar a forma com a qual o usuário aparece no mapa, sendo possível selecionar um "avatar" dentre vários disponibilizados pela plataforma.

**Alertas.** Os usuários podem, e devem, alertar aos outros sobre problemas na vida e informações importantes para a comunidade. Esta é a base de um sistema colaborativo e o Waze implementa isso de forma muito inteligente, fazendo com que seus usuários se tornem cada vez mais pró-ativos, disponibilizando todo o apoio à informação e à interação do usuário com o sistema. Atualmente, existem 9 tipos de alertas, como mostra a figura 3.2. Cada um destes tipos de alertas são explicados a seguir:

- **Trânsito.** Contém informações sobre engarrafamentos e trânsito lento em determinada via da cidade. O usuário pode classificar entre 'Moderado', 'Intenso' e 'Parado', além de informar o sentido da via em questão, uma mensagem descritiva da situação e o envio de uma foto para comprovar o fato.

- **Polícia.** Assim como para o Trânsito, o usuário é capaz de informar aos outros usuários sobre a existência ou não de policiamento em determinada via, podendo classificar entre 'Visível' ou 'Oculta', além de uma mensagem descritiva, do sentido da via e de uma foto para comprovar o fato.

- **Acidente.** O usuário pode reportar um acidente, fazendo com que os outros usuários tenham cuidado ao trafegar pela via. Classifica-se um acidente como leve ou grave, além de uma mensagem descritiva para o fato, o sentido da via e uma imagem para comprovar a informação.

- **Perigo.** A informação de perigo informa ao usuário sobre a existência de perigo na via em questão, podendo este ser realmente na via, no acostamento ou até relacionado ao clima. As outras informações para comprovar o fato também se fazem valer nesta categoria.

- **Radar.** A opção de alerta de radar, assim como a de policiamento é bastante controversa entre as pessoas que utilizam o sistema. Alguns pensam ser saudável a informação de forma aberta, outros pensam que favorecem que usuários mal intencionados utilizem-na para cometerem infrações de trânsito.

- **Outros.** Além dos alertas citados, existem outros, como por exemplo o de preços nos postos de combustíveis, o de bloqueio de vias e de erro no mapa. Estas

informações possibilitam ao usuário uma informação mais rica e, ao waze, recomendações de rotas melhores.

#### 3.2.1.2 Reputação no Waze

O Waze utiliza um sistema de reputação para motoristas, que é dividido em níveis: Waze Bebê, Waze Adulto, Waze Guerreiro, Wazze Cavaleiro e Waze Rei. Para subir de nível, o motorista precisa cumprir algumas missões, ganhando pontos com isso, como por exemplo dirigir 800 km em uma semana, editar uma parte do mapa, fazer amigos, e por aí vai. Uma outra forma de aumentar sua reputação, é dirigir por ruas pouco conhecidas, que aparecem pontilhadas no mapa. Com isso, o Waze premia o usuário com pontos e aumento de reputação e conhece, de forma direta, algo sobre a via informada, confirmando que ela existe. Wazers também ganham pontos ao alertar o Waze que uma via não existe.

Com estes pontos sendo ganhos, o Waze começa a confiar no usuário, permitindo-lhe executar operações que antes não eram permitidas. Basicamente porque o sistema de pontuação é capaz de prever se um Wazer é, ou não, confiável. Alguns dos níveis de reputação só podem ser alcançados quando o usuário está entre os usuários mais ativos de cada país. O Waze considera a pontuação de todos os usuários e verifica quais os 10% mais ativos, por exemplo, para dar determinado título a um usuário.

Editores de mapa têm formas de reputação diferentes. De acordo com a quantidade de edições, sua reputação vai subindo de forma que o usuário, ou editor, possa escolher novos avatares e benefícios proporcionados pelo sistema.

#### 3.2.1.3 Formas de Incentivo no Waze

O Sistema determina as formas de incentivo baseadas na reputação de seus usuários. Quanto mais alta for a reputação, maior os benefícios. Estes benefícios se baseiam na confiança que o sistema tem pelos usuários através da verificação de sua reputação. No entanto, a maior parte dos incentivos fica por parte da possibilidade de modificar o avatar do usuário. No entanto, vemos como incentivo indireto o fato de o usuário ter acesso privilegiado às informações do sistema simplesmente por participar. Editores de mapa contam com avatares diferenciados do resto dos usuários. Esta medida faz com que os usuários passem a interagir com a plataforma de forma mais direta, ajudando a construir e atualizar as informações referentes ao mapa. Esta funcionalidade realmente requer um grau de confiança maior no usuário, pois os mapas são o centro da aplicação e sem eles nada mais existe.

### 3.2.2 Vivino

O Vivino[10] é uma aplicação colaborativa com informações sobre vinho. Nesta aplicação, você pode saber informações sobre onde comprar, quanto custa e avaliações sobre os vinhos. Em Julho de 2013 chegou a cerca de 1 milhão de vinhos completamente catalogados em seu banco de dados. Assim como os vinhos, o Vivino fica cada vez melhor com o passar do tempo, pois novos vinhos e informações importantes são adicionados pela comunidade de usuários da aplicação. Algumas das características básicas do Vivino são a possibilidade de obter recomendações personalizadas de vinhos de acordo com os gostos do usuário, a possibilidade de seguir usuários, obtendo inspiração, conhecimento e, com isso melhores vinhos, a possibilidade de encontrar amigos através do twitter, e-mail ou pelo facebook. Além disso, é possível adicionar vinhos à sua lista de desejos, organizar seus vinhos por data, preço e/ou avaliação e, ainda, adicionar notas personalizadas para cada um de seus vinhos.

#### 3.2.2.1 Serviços do Vivino

Assim como Waze, descrito na seção 3.2.1, o vivino também faz uso de informações coletadas por seus usuários para prover novas e melhores informações aos usuários que participam da comunidade. No Vivino, especificamente, estes conceitos e serviços estão diretamente relacionados à disponibilidade de o usuário poder ter uma forma de avaliar e ver avaliações de vinhos. No geral, a aplicação tem como base que os usuários estão realmente interessados nos vinhos que veem na aplicação e, através deste pré-requisito, disponibiliza a eles formas de avaliar, adicionar notas, informações relevantes e tudo o que agregar valor à informação geral da aplicação. Nas figuras 3.3 e 3.4, vemos a forma com a qual o aplicativo informa ao usuário as características de um vinho específico. Na primeira imagem, temos as informações básicas do produto, como nome, avaliação, preço e tipo do vinho, além de opções de adicionar lugares onde o vinho é comercializado e o preço destinado a tal produto naquele estabelecimento. Na segunda imagem, observamos, de forma separada, quantos usuários avaliaram o produto e quantas estrelas eles acharam que aquele vinho merecia. Outras informações importantes que a aplicação dá é a representatividade deste vinho em termos globais, informando a importância deste através de rankings por país, por vinícola, por região, entre outros. Assim como fizemos na seção 3.2.1.1, separamos os serviços a seguir:

– **Integração com Redes Sociais.** Logo na primeira tela, observamos que a aplicação possui integração de contas com o facebook, o google plus[11] e possibilita

[10] www.vivino.com
[11] http://plus.google.com


**Figura 3.3.** Tela de informações sobre vinho no vivino.

a criação de conta através do e-mail pessoal. De qualquer forma, esta integração, pelo menos neste ponto, serve como base para a aplicação ter contato com o perfil do usuário. Após selecionar um dos meios de cadatro, algumas informações básicas como nome, sobrenome, país, cidade e foto são adicionadas à tela do usuário, sendo possível modificá-las ou prosseguir com o cadastro. Cadastrado e logado na aplicação, o usuário tem a possibilidade de, através de uma ou várias redes sociais, começar a seguir amigos, além dos usuários em destaque. Neste contexto, seguir um usuário significa visualizar suas últimas avaliações e notas dadas a determinado vinho.

– **Georeferenciamento.** Através do GPS do usuário, o sistema tem as informações suficientes para adicionar novas características a um produto e com isso preencher ainda mais a base de dados. Com a possibilidade de adicionar informações de estabelecimentos que vendam determinado produto, o usuário possui uma ferramenta importante na busca por vinhos: a possibilidade de encontrar estabelecimentos próximos que o vendam. Esta informação pode impulsionar as vendas e adicionar competição entre os estabelecimentos mais próximos, já que é possível também visualizar o preço de determinado produto no local de venda.

Feito isso, o usuário tem a opção de verificar quais dos estabelecimentos estão com os melhores preços em uma cidade e, além disso, saber a média geral da cidade.

- **Avaliação dos Produtos.** A avaliação de produtos é uma característica importante dentro da aplicação. Um produto bem avaliado pela comunidade informa aos outros que aquele produto é bom e é um meio natural de propaganda para pessoas que não estão habituadas com o produto. Com a possibilidade de filtrar produtos pela avaliação que estes receberam, o sistema tem uma forma de dar maior visibilidade aos produtos melhores avaliados pelos usuários, assim como os que receberam notas piores têm o inverso. Baseado em suas avaliações, o sistema descobre quais características do vinho votado tem com relação com outros vinhos cadastrados no sistema. Essa informação se faz importante para a recomendação de vinhos ao usuário. No caso, o usuário precisa avaliar no mínimo 3 produtos para que o sistema seja capaz de "casar" os dados e encontrar boas recomendações para o usuário.

- **Perfil Visivel ou Privado.** Cada usuário tem seu perfil dentro do sistema. Assim como outros sistemas colaborativos, é imperativo que a aplicação permita ao usuário definir suas opções de privacidade. No caso do Vivino, o usuário pode claramente informar ao sistema se ele quer que seu perfil seja público ou que suas informações se mantenham privadas. Isso faz com que usuários, que não se sentem seguro com questões de localização tenham sua privacidade preservada através destas configurações.

- **Reconhecimento de vinhos por foto.** Para se encontrar determinado produto e, então, saber a informação deste, o sistema permite que o usuário envie para o servidor a imagem e receba como a informação de qual produto fora visto. Isso é importante para situações onde o usuário não conhece o produto e quer acesso rápido à informação.

#### 3.2.2.2 Reputação no Vivino

A reputação no Vivino é algo que não parece tão decisivo para as formas de incentivo que a aplicação dá. Basicamente, a reputação se dá através das avaliações do usuário, é claro. No entanto, o que temos é um ranking geral e absoluto, nada de informação relativa. Aparentemente, isso causa no usuário, pelo menos a um primeiro momento, uma forma de fazer com que o usuário não se sinta empolgado com a participação, pois uma única avaliação ajuda muito pouco a subir no ranking.

#### 3.2.2.3 Formas de Incentivo no Vivino

A reputação, como explicada na seção 3.2.2.2, não tem grande influência nas formas de incentivo utilizadas pelo Vivino. Basicamente, o grande incentivo se dá de forma indireta, com informações relativas à novos vinhos e avaliações dadas pelos usuários. O sistema trabalha com um esquema de Badges, ou emblemas, dadas à seus usuários de acordo com seu progresso na aplicação. Este progresso está diretamente relacionado com a quantidade de avaliações, a quantidade de vinhos digitalizados, o tempo ao qual o usuário faz parte do sistema e outras características relativas à utilização.

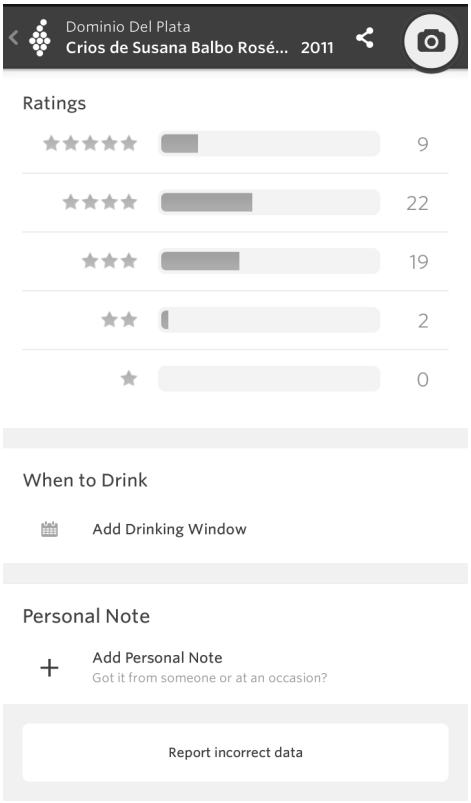


**Figura 3.4.** Tela de avaliações de vinho no vivino.

## 3.3 Comentários Parciais

Os trabalhos relacionados servem como fonte de requisitos e diretrizes a seguir no projeto de criação do *framework*. As aplicações e os mecanismos de incentivo trouxeram problemas que devem ser tratados em uma aplicação correspondente à informações enviadas por usuários comuns. Desta forma, destacam-se alguns dos pontos mais importantes e que, seguindo as tendências, serão implementadas no *framework* a seguir:

– **Integração com as Redes Sociais.** Tanto Waze, quanto o Vivino implementam funcionalidades referentes à integração do usuário com as redes sociais e, através disso, será possível ter uma gama maior de usuários, aproveitando os usuários já presentes em aplicações sociais. As formas de incentivo também fazem uso desta implementação, já que muitos dos mecanismos estudados tentam justamente manter um bom número de usuários competindo por incentivos e visibilidade na aplicação.

– **Customização da Interface.** Mesmo estando presente apenas no Waze (possibilitando alteração entre mapas 2D e 3D e suas cores) esta funcionalidade pode ser mais explorada no sentido de possibilitar que o desenvolvedor não esteja preso a um template específico do *framework* e, além do mais, a facilidade que a plataforma oferece ajuda com que esta funcionalidade esteja presente em um *framework* desta categoria.

– **Alertas e Notificações.** São parte importante de uma aplicação que tenta atrair a atenção de seu usuário. Notificações sobre determinado produto ou sobre a avaliação de outro, bem como ganhos de pontuação (ajudando as formas de incentivo a serem notadas) são formas bem específicas de se afirmar que a necessidade de gerenciadores de alertas e notificações devem estar presentes na ferramenta.

– **Reputação.** Este item pode facilmente se relacionar com as formas de incentivo, que fazem parte da base de qualquer aplicação com sensoriamento participativo. Desta forma, deverá ser implementado mecanismos que auxiliem, ao desenvolvedor, a desenvolver mecanismos de reputação e incentivo aos seus usuários.

– **Georeferenciamento.** Por possibilitar mais informações referentes ao usuário que envia uma avaliação ou uma coleta específica, entende-se que este é um mecanismo que não poderia faltar em uma aplicação de avaliação de produtos de forma colaborativa. Seja para fins de pesquisa de estabelecimentos ou para enviar sua localização e, assim, outros usuários saberem o preço de um produto em determinado estabelecimento, esta funcionalidade estará no *framework*. Importante notar, que esta funcionalidade precisa ser implementada respeitando a privacidade do usuário. Caso o mesmo não deseje compartilhar suas informações de localização, esta não deverá ser utilizada.

– **Avaliação de Produtos.** Da mesma forma que o Vivino implementa funcionalidades de avaliação e preços de produtos, o *framework* será capaz de auxiliar o

desenvolvedor a implementar estas funções sem grandes dificuldades através de managers responsáveis por fazer a comunicação da aplicação com o servidor, que será o responsável por armazenar estas informações.

– **Perfil.** O princípio básico da reputação consiste na visualização da pontuação ou no título de honra de determinado usuário. Desta forma, a capacidade de visualizar perfis de outros usuários está intimamente ligada às aplicações e, com isso, será implementada e disponibilizada para o desenvolvedor.

– **Reconhecimento de Fotos.** A ferramenta oferecerá suporte ao envio de imagens para o servidor, que buscará por imagens semelhantes em sua base de dados, retornando ao usuário o produto ao qual aquela imagem mais se assemelha. Esta funcionalidade será implementada no *framework* em função dos benefícios que esta facilidade oferece aos desenvolvedores e, posteriormente, aos usuários da aplicação que desejam saber informações de um produto ao qual não se sabe o nome, por exemplo.

# Capítulo 4

# Product Hunter

Este capítulo apresenta o projeto do Product Hunter, o *framework* desenvolvido neste trabalho. A implementação e a modelagem seguem o padrão dos *frameworks* encontrados no capítulo anterior, onde apresentamos os trabalhos relacionados e as aplicações que servem de instrumento para se obter funcionalidades a serem implementadas neste projeto. Como visto no capítulo 3, os *frameworks* implementados fazem referência à utilização de *frameworks* para o auxílio à construção de aplicações de *crowdsensing*, analisando conceitos importantes como a utilização de dispositivos móveis e suas limitações, como consumo de banda e de energia, recurso este, que é explorado pelo Product Hunter de forma a reduzir o consumo de energia em função de suas requisições diretas para o servidor. Um outro diferencial de nosso trabalho é em relação ao domínio de aplicações aos quais o *framework* se destina, pois, em nosso caso, tratamos as funcionalidades como sendo se fossem imprescindíveis em aplicações que relacionam determinado produto com suas informações, preços e, inclusive, avaliação.

## 4.1 O *Framework*

Como explicado na seção 2.5, um *framework* é um arcabouço conceitual para o desenvolvimento de aplicações que seguem um mesmo fluxo pré-definido e contenham características semelhantes a partir de uma mesma base. Esta base consiste em classes com funções bem definidas e utilizadas pelas aplicações a serem desenvolvidas com base no *framework*. No entanto, como ficou claro na seção 2.5, existem diferenças entre os conceitos de *framework*. No entanto, o conceito abordado por este trabalho foi o definido por Buschmann et al. (1996) e Pree (1995), que diz que um *framework* é definido como um *software* parcialmente completo projetado para ser instanciado.

Este *framework* utiliza algumas ferramentas básicas para servirem de base para

o desenvolvimento. Para isso, separamos a implementação das aplicações cliente com relação ao servidor, que será responsável por armazenar as informações e enviá-las para os outros usuários da comunidade colaborativa. Para o cliente, implementamos o código e o destinamos à criação de aplicações para Android, por ser o Sistema Operacional para dispositivos móveis mais utilizado no mundo, bem como pela boa integração que seus dispositivos e o sistema possuem com os sensores que podem ser úteis para a aplicação. Com relação ao Servidor, utilizamos CakePHP, por motivos de simplicidade, sem funcionalidade. Além do mais, o CakePHP é uma das ferramentas mais utilizadas por programadores PHP. De acordo com a figura 4.1, vemos como funciona a separação entre os dois 'nós' básicos do Product Hunter: À esquerda, o Servidor, utilizando CakePHP, e à direita o Dispositivo Móvel, utilizando Android.

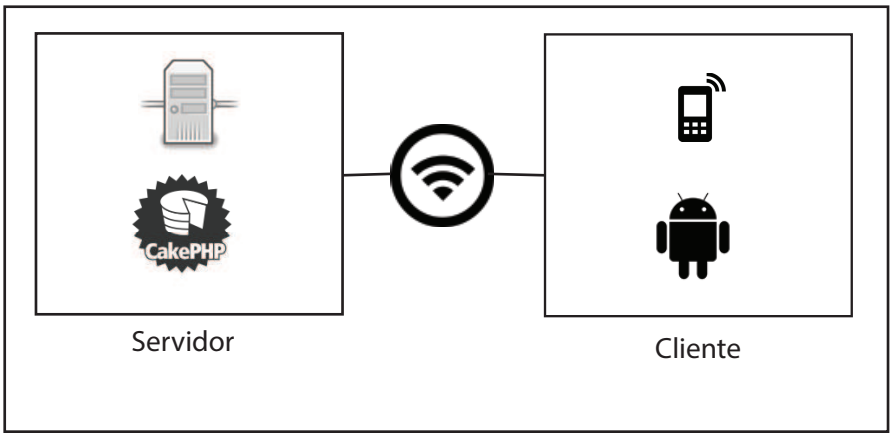


**Figura 4.1.** Exemplificação do *Framework* e das aplicações desenvolvidas a partir do projeto.

## 4.2 A arquitetura do *framework*

A arquitetura do *framework* consiste, de forma direta, na utilização de duas implementações prontas para prover agilidade no desenvolvimento de sistemas colaborativos, pois integra funcionalidades para as aplicações móveis e, também, as funcionalidades prontas para receber às requisições no Servidor.

Para que ambos os lados da comunicação sejam estabelecidos de forma proveitosa, o Servidor disponibiliza alguns métodos responsáveis pela execução das funções presentes no servidor. Como o CakePHP já implementa uma arquitetura MVC e a parte Servidor já se utiliza da arquitetura básica do CakePHP, utilizamos sua arquitetura básica para responder às requisições do cliente. O cliente, então, provê a utilização de

métodos que acessem diretamente a Application Programming Interface (API) liberada pelo servidor.

A implementação do lado servidor é mais simples e, basicamente, requer apenas a adição de funcionalidades para que passe a funcionar corretamente. Sendo assim, a parte Cliente necessita de métodos que ajam de forma correta, executando suas requisições diretamente para os métodos disponibilizados pelo Servidor. Com isso, separamos o Cliente em 3 camadas, para refletir a arquitetura criada pelo CakePHP e herdada por nosso *framework*. Para isso, criamos uma arquitetura (figura 4.2) baseada em Classes Abstratas (AbstractClasses), Classes Gerenciadoras das Requisições da Aplicação (Managers) e as Classes que unem requisições semelhantes ou que precisem ser executadas juntas para se obter ou enviar uma informação para o servidor (Modules).

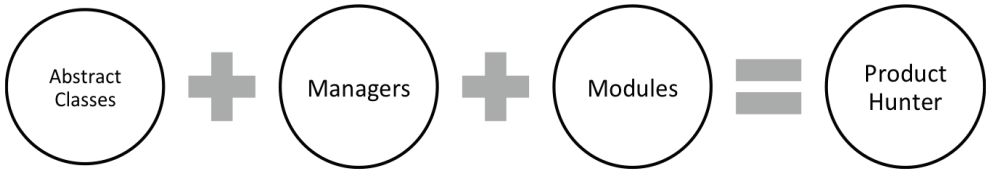


**Figura 4.2.** Camadas que formam o *Product Hunter Framework*.

A figura 4.3 exemplifica a comunicação entre as camadas. Os Managers e os Módulos podem acessar os membros das AbstractClasses de forma direta. A camada de Managers faz chamadas diretas às AbstractClasses, assim como as Modules classes, que apresentam uma forma de modular determinada funcionalidade que utiliza várias funcionalidades presentes nos gerenciadores (Managers) individuais. Sendo assim, de uma forma geral, conceituamos:

– **AbstractClasses.** São classes responsáveis por armazenar informações pertencentes ao que é utilizado na aplicação. Como AbstractClasses básicas, definimos, no *Product Hunter*: AbstractRate, AbstractUser, AbstractProduct, AbstractPlace, AbstractPrice e AbstractRanking.

– **Managers.** São classes que possuem a implementação básica das requisições para o servidor, transformando o que é recebido como resposta em Objetos de classes pertencentes às AbstractClasses e reparrando estas informações para a classe que originar a requisição.

– **Modules.** São classes que unem diferentes requisições presentes nos Managers de forma a simplificar o processo de busca e envio de informações, tendo a mesma função, no sentido de iniciar requisições, dos Managers.

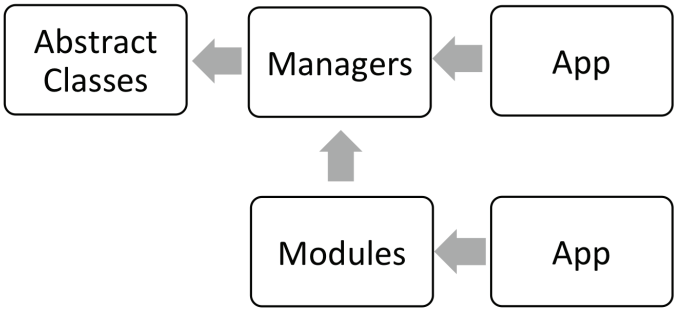


**Figura 4.3.** Comunicação entre as camadas do *Product Hunter Framework*.

Após cada requisição feita, o Servidor processa os dados de entrada e, após obter os resultados, retorna a informação para a aplicação em forma de JSON. Um exemplo de JSON pode ser visto na figura 4.4, onde a resposta em questão refere-se ao cadastro de um usuário. Esta informação é processada pelo *framework* e passada como resultado da requisição.

## 4.3 Modelagem do *Framework*

Nesta seção, apresentamos as peculiaridades da implementação deste *framework*. A figura 4.5 mostra o modelo utilizado para a definição das classes deste *framework*, através de um Diagrama de Classes.

A figura 4.5 demonstra as classes utilizadas para armazenar as informações relevantes do *framework* e explica a relação entre os as classes a serem utilizadas para armazenar as informações que serão utilizadas na aplicação final. Usuários se relacionam diretamente com suas avaliações, seus preços enviados e seus estabelecimentos informados. Um usuário pode ter várias avaliações feitas, sobre diversos outros produtos e o mesmo ocorre com preços e estabelecimentos. Desta maneira, produtos também

```
{
    "message":"Usuário criado com sucesso.",
    "exception":0,
    "query":
        {
            "User":
            {
                "nome":"Luis Menezes",
                "login":"luis_menezes",
                "url_image":"/img/user/19812749b.jpg",
                "dateTimeCreated":"2014-05-07 07:32:59",
                "ranking":1,
                "id_user":"28"
            }
        }
}
```


**Figura 4.4.** Exemplo de JSON, representando o cadastro do usuário.

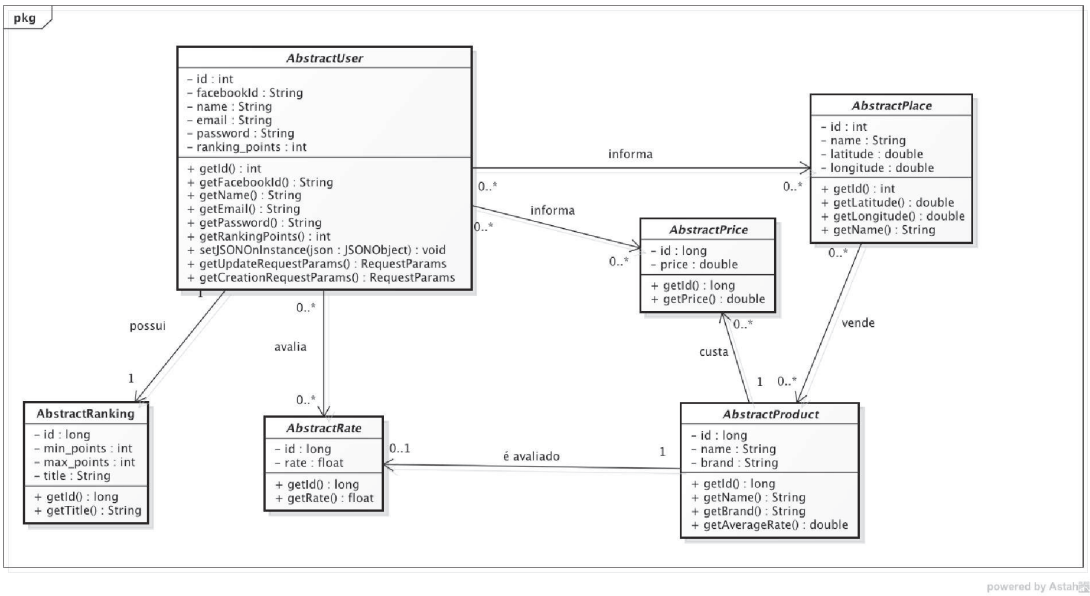


**Figura 4.5.** Diagrama de Classes do *framework*.

se relacionam com avaliações, preços e estabelecimentos, já que as informações feitas pelos usuários estão diretamente atreladas ao conceito de informações e avaliação de produtos. Com este diagrama, mostramos, basicamente, no que o *framework* se sustenta para possibilitar acesso às informações de forma relacionada. Todas as funcionalidades presentes no mesmo fazem estas relações de forma direta, provendo informações importantes para a aplicação. Todas as classes são definidas como abstratas, e contêm métodos que devem obrigatoriamente ser implementados na classe que a extender. Por exemplo: a classe AbstractUser obriga, à classe que a estende, a implementação de in-

formações como `setJSONOnInstance(JSONObject json)`, que informa ao *framework* os parâmetros a serem utilizados em caso de alteração da classe na classe implementada na aplicação final.

## 4.4 Análise de Requisitos

Como parte importante da construção da ferramenta e elaboração de funcionalidades que permitam ao desenvolvedor um real ganho de tempo, é necessário que se faça uma análise dos requisitos baseados nas funcionalidades observadas por aplicações que se assemelham àquelas as quais devem ser desenvolvidas pelo *framework*, bem como pela análise dos *frameworks* relacionados. Esta análise foi feita no capítulo 3 e esta seção irá mostrar quais os pontos relevantes a serem abordados neste projeto.

Como requisitos básicos, é importante que toda a parte de requisição e acesso ao servidor seja implementado como forma básica de conexão, se tornando a fonte principal de comunicação para as outras funcionalidades, inclusive. Desta forma, julgou-se necessário a criação de um gerenciador de requisições, tendo como base requisições pré-definidas para o servidor, sendo possível passar a URI, os parâmetros e o método de acesso à requisição. Estas requisições oferecem suporte suficiente para a execução de outras funcionalidades, que podem ser visualizadas a seguir.

- **Controle de Banco de Dados.** Esta funcionalidade refere-se ao armazenamento de informações no banco de dados. Mesmo sendo uma funcionalidade 'pouco visual', já que os dados estão armazenados e o usuário comum não tem acesso à esta informação de forma direta, relaciona-se e muito com qualquer tipo de aplicação que possua um grande número de informações a ser lido e/ou passado adiante.

- **Internacionalização da Aplicação.** Esta funcionalidade refere-se à possibilidade de tornar uma aplicação traduzível para vários idiomas. É necessário que o sistema seja capaz de traduzir seu conteúdo para conseguir chegar a diversos usuários sem que a barreira da linguagem seja um real problema.

- **Integração com Serviços de GeoReferenciamento.** Esta funcionalidade refere-se ao fato de utilizar informações de localização do usuário (normalmente latitude e longitude) para realizar buscas e adicionar informações às coletas feitas. Muitas aplicações utilizam esta funcionalidade para incrementar suas informações e outras para prover mais informação acerca de determinado estabelecimento ou local próximo a ele.

– **Controle de Usuários.** Esta funcionalidade é observada nas aplicações relacionadas (Seção 3.2), concentra-se no controle de informações e requisições sobre uma informação do usuário. Deste grupo, fazem parte as funcionalidades de Cadastro de Usuário, Login, Logout, Visualização de Informações, Visibilidade de Usuário, Controle da Sessão atual do usuário, entre outras funcionalidades relacionadas ao tema.

– **Controle de Produtos.** Esta funcionalidade é observada no Vivino (Seção 3.2.2), onde é possível ter um controle sobre os produtos. Esta funcionalidade permite: Visualizar Produtos, Adicionar Produtos (moderado pela aplicação), Avaliar Produtos, Ver Informações, Saber os Preços, entre outras possibilidades.

– **Informações sobre Estabelecimentos.** Esta funcionalidade permite acesso às informações obtidas através do georeferenciamento, possibilitando ter acesso a informações relacionadas aos estabelecimentos mais próximos, sabendo quais produtos são vendidos por este, bem como seus preços e informações especiais.

– **Adição e Visualização de Avaliações de Produtos.** Esta funcionalidade refere-se à proposta básica do *framework*, que é a de dar ao usuário a possibilidade de avaliar os produtos de forma colaborativa e, com isso, obter informações privilegiadas a partir da colaboração de outros usuários.

– **Integração com Redes Sociais.** Esta funcionalidade refere-se ao controle de algumas funcionalidades que relacionam a aplicação à utilização de informações ou funcionalidades presentes nas redes sociais. Serve, de forma direta, para relacionar usuários às informações que a aplicação deseja saber sobre ele, como por exemplo suas informações pessoais. Além disso, facilita acesso à aplicação e permite visualização de amigos de forma a trazê-los para a aplicação e, assim, fazê-la mais 'amigável' no sentido interativo.

– **Incentivo e Reputação.** Esta funcionalidade se faz muito necessária em aplicações colaborativas. Tanto Waze (Seção 3.2.1), quanto Vivino (Seção 3.2.2), utilizam meios de manter seus usuários animados para continuar contribuindo, além de dar maior visibilidade e reputação àqueles que o fazem de forma exemplar. Estas formas de incentivo e reputação dependem da aplicação à qual elas se referem, no entanto, é imprescindível adicionar meios de facilitar o desenvolvimento e o aumento da especificidade da informação para aplicações que necessitarem implementar seus próprios métodos de incentivo e reputação.

– **Busca de Produtos por Imagens.** Esta funcionalidade, observada no Vivino (Seção 3.2.2), oferece ao usuário uma alternativa importante na busca por informações de determinado produto, trazendo facilidade para encontrá-lo dentre os muitos produtos existentes em uma base de dados.

Além destas funcionalidades, a ferramenta pode auxiliar o usuário a desenvolver muitas outras, pois o básico, relacionado à conexão, para todas as outras funcionalidades, considerando uma aplicação colaborativa, já está pronto de forma nativa, sendo necessário apenas que o usuário utilize os métodos prontos e passe as informações necessárias para a execução de outras tarefas.

## 4.5 Funcionalidades Implementada

### 4.5.1 Controle dos Usuários

O Controle dos Usuários fica relacionado à uma das funcionalidades mais básicas de qualquer aplicação colaborativa. Em comunidades colaborativas, um usuário refere-se diretamente à entidade colaboradora maior de todas dentro da aplicação. Desta forma, a criação e o controle de suas informações torna-se parte primordial da aplicação, sendo necessária, ainda, para a utilização de outras funcionalidades, como por exemplo a avaliação de um produto. Qual a 'validade' ou confiabilidade empregada a esta avaliação sem a utilização da entidade Usuário? Seria apenas uma avaliação, dada por alguém anônimo, sem informações relacionadas e, muitas vezes, poderia estar completamente equivocada. Em alguns casos se faz necessária a utilização de entidades anônimas para detalhar determinada informação, por questões de segurança. Mesmo assim, a aplicação provê formas de se fazê-lo, sem perda de qualidade para os outros usuários.

É para resolver estes problemas que o controle de usuários foi implementado. Com ele, é possível Cadastrar Usuário, Iniciar e Finalizar a sessão no aplicativo, obter informações referentes às suas colaborações, notas informadas, preços enviados e, com isso, criamos uma relação maior entre as informações presentes na aplicação e a existência de alguém não anônimo para a aplicação colaborativa.

Com estas funcionalidades já implementadas, retira-se do desenvolvedor a necessidade de implementá-las novamente. Este tipo de implementação não difere muito entre as aplicações. O máximo que pode existir é o acréscimo ou a não utilização de determinado atributo. No entanto, a base sempre se mantém, facilitando o foco do desenvolvedor na aplicação final.

### 4.5.2 Controle de Sessão

Ao caracterizar uma entidade colaboradora como um usuário, é imperativo dar capacidade suficiente à aplicação de prover gerenciamento da sessão do usuário atualmente "logado" na aplicação. Para isso, a ferramenta provê uma forma correta de obter o usuário recém logado na aplicação e, através disso, informar à aplicação final qual o usuário que está atualmente fazendo uso da mesma. Sendo assim, permitimos que a aplicação utilize um dos métodos providos pelo *framework* e trabalhe com o usuário da forma que achar melhor.

A implementação desta funcionalidade de forma nativa ajuda ao desenvolvedor a se preocupar com outras funcionalidades mais importantes, deixando o básico para o sistema base, que por si só já é capaz de caracterizar as informações referentes ao usuário que está utilizando o sistema no momento de uma requisição.

### 4.5.3 Gerenciamento de Requisições

Uma característica importante de todas as aplicações móveis colaborativas é o tratamento das conexão para envio e recebimento das informações provenientes do serviço provedor. Normalmente, as conexões móveis são bem frágeis e sujeitas à quedas e desconexões. Sendo assim, é completamente aceitável e, muitas vezes, requerido, que a aplicação conte com ferramentas capazes de tratar e trabalhar de forma adequada com as conexões em cada espécie de requisição feita pelo usuário.

Este gerenciamento permite ao desenvolvedor atentar-se apenas àquilo que difere das requisições básicas, mas não o limita à utilizá-la de forma específica na aplicação.

### 4.5.4 Gerenciamento de Banco de Dados

Aplicações colaborativas trabalham de forma direta com muitas informações. Desta forma, é imprescindível que haja um controle direto às informações de forma facilitado. O Controle de Banco de Dados utiliza uma biblioteca muito utilizada na atualidade, o Object Relational Mapping Lite (ORMLite). Este controle mais direto permite, à aplicação, eliminar a atenção à implementação de classes destinadas unicamente à criação do banco utilizando Structured Query Language (SQL) puro, reduzindo a possibilidade de bugs na base de dados e, com isso, a perda de informações.

### 4.5.5 Integração com Redes Sociais

Assim como outras características, a integração de aplicações colaborativas com o crescente uso das redes sociais deve receber o seu devido valor. Isso se deve ao fato de que as redes sociais impulsionam a utilização das aplicações colaborativas de forma que, por motivos sociais, os usuários que fazem parte da rede, começam a se sentir influenciados a participar da aplicação, e, com isso, visam querer estar em evidência do seu círculo social. Além disso, com o grande número de usuários presentes nas redes sociais, é possível que a criação de serviços capazes de importar usuários, presentes nestas redes, para cadastros em aplicações colaborativas impulsione o crescimento quantitativo da nova comunidade, ajudando a aumentar o fluxo de informações e, consequentemente, de coletas. Esta integração deve fazer parte de um módulo pré-definido, que encapsule as informações do usuário e provenha as mesmas para a aplicação final.

### 4.5.6 Integração com Serviços de Georeferenciamento

Diversas aplicações colaborativas fazem uso do georeferenciamento para encontrar pontos de referência próximos ao local da coleta. Além disso, é possível informar um usuário sobre quais coletas foram feitas considerando os locais de coletas mais próximos a ele. Tudo isso pode ser feito de forma básica pelo *framework*. O georeferenciamento pode ser utilizado de várias formas em um sistema colaborativo, mas os módulos encontrados nas aplicações relacionadas e, que, serviram como base para a elaboração das funcionalidades do *framework*, dispunham da utilização do georeferenciamento para encontrar locais de coleta mais próximos ao usuário e também para adicionar a informação sobre o atual local da nova coleta. Com isso, é imprescindível que o *framework* seja capaz de deixar ao desenvolvedor a missão de utilizar ou não os métodos relacionados à esta funcionalidade. No entanto, é necessário que ele possibilite ao mesmo utilizar os métodos previamente implementados na base do sistema.

### 4.5.7 Internacionalização

Atualmente, a maioria das aplicações presentes nos dispositivos móveis já conta com métodos funcionais de internacionalização. O próprio Android já utiliza alguns mecanismos de transformar o texto que é exibido ao usuário em diversas linguagens. Para isso, basta que o desenvolvedor informe o texto correto nas linguagens que deseja e, após isso, delimitar qual a linguagem padrão para o sistema. Estas informações levam em consideração o idioma atual do dispositivo do usuário. No entanto, diversos dispositivos não dispõem de nenhum idioma além do inglês, o que leva ao usuário coletor

ficar preso neste idioma, mesmo que o desenvolvedor da aplicação tenha possibilitado a utilização da mesma em outras linguagens. Uma forma de solucionar este problema é deixar o usuário escolher o idioma com o qual a aplicação irá funcionar, podendo alterar esta configuração ao longo da aplicação, no momento que achar melhor. Para isso, é necessário que o *framework* contenha métodos de tradução do que é exibido na tela para a linguagem que o usuário escolher, no momento que este escolher ou tendo como pré-requisito que o mesmo reinicie a aplicação para que o efeito seja visualizável. Este processo de internacionalização permite aos usuários uma forma a mais de estar utilizando a aplicação, pois possibilita que leigos em um determinado idioma não estejam presos à outro e, com isso, não entendam determinada informação contida na aplicação final. Isso tudo poderia ser tratado pelo desenvolvedor, no entanto, como a função do *framework* é facilitar, que isso seja feito pelo *framework* então.

### 4.5.8 Incentivo e Reputação

Assim como observado nas aplicações analisadas na seção 3.2.1 e 3.2.2, nos *frameworks* relacionados (seção 3.1) e nas formas de incentivo (seção 2.6), aplicações colaborativas, geralmente, fazem uso de formas de incentivo e reputação para trazer usuários e, mais do que isso, mantê-los de forma pró-ativa no que diz respeito à contribuição das coletas para a comunidade colaboradora. Sendo assim, é imprescindível que formas de incentivo e reputação já estejam presentes no *framework* no momento de sua instanciação, para que novas aplicações façam uso do fluxo estabelecido pelo *framework* com o objetivo de implementar estas funcionalidades de forma mais fácil.

### 4.5.9 Busca de produtos por imagem

Uma das funcionalidades observadas através da análise do vivino, na seção 3.2.2, diz respeito ao desenvolvimento de um canal de busca de determinada informação no servidor através do envio de uma imagem específica. Como este *framework* diz respeito à criação de aplicações colaborativas para busca e avaliação de produtos, é imperativa a criação de uma forma de procurar estes produtos através de imagens capturadas pelo usuário. Sendo assim, tanto o *framework* para a aplicação móvel quanto o servidor devem estar preparados para encontrar os produtos que mais se parecem com a imagem enviada pelo usuário. Isso ajuda a encontrar mais fácil as informações requisitadas, sem a necessidade de digitar para encontrá-las.

Através desta necessidade, utilizou-se o ImageMagick[1], descrito em Taylor (2009)

[1] www.imagemagick.org

e Taylor (2013), como ferramenta de tratamento das imagens a fim de procurar por semelhanças dentre as presentes na base. Esta ferramenta é utilizada no lado servidor do *framework*, implementado através de um módulo disponibilizado para PHP que permite a utilização de suas funções de forma prática. Deste modo, o que se faz é, através da aplicação cliente, upload da imagem para o servidor, que compara esta imagem com uma lista de imagens dos produtos já presentes na base de dados e retorna aquela com a maior semelhança encontrada dentre as comparadas na lista.

## 4.6 Modelo do Framework

### 4.6.1 AbstractClasses

Grande parte do *framework* trabalha em função dos dados presentes em objetos que são passados e modificados pela aplicação principal. Estes objetos são direcionados a realizar operações envolvendo Produtos, Usuários e o que mais estiver sendo utilizado pelo *framework*. Com isso, se faz necessária a criação de objetos abstratos, obedecendo um papel genérico na compreensão das informações passadas de uma ponta a outra da comunicação entre os clientes e o servidor. Neste trabalho, as requisições são feitas analisando as informações de classes instânciadas herdando as seguintes classes abstratas: AbstractPlace, AbstractPrice, AbstractProduct, AbstractRate, AbstractUser, AbstractRanking. Cada uma possui uma peculiaridade, listada a seguir:

- **AbstractPlace.** Refere-se às informações relacionadas aos locais utilizados na aplicação. Uma classe que herde de **AbstractPlace** deve, obrigatoriamente, implementar o método `setJSONOnInstance(JSONObject object)` para converter os elementos presentes no JSON em um objeto da classe.

- **AbstractPrice.** Refere-se às informações relacionadas aos preços dos produtos da aplicação. Uma classe que herde de **AbstractPrice** deve, assim como a classe anterior, implementar o método `setJSONonInstance(JSONObject object)` de forma a fazer o parse do JSON em objeto da classe.

- **AbstractProduct.** Refere-se às informações relacionadas a uma das principais classes básicas da aplicação: O Produto. Requer a implementação do método `getCreationRequestParams()`, que retorna uma instância de RequestParams e será utilizado para enviar os dados via POST para o servidor e a implementação do método `setJSONOnInstance(Object object)` para fazer o parse dos elementos do JSON em instância da classe herdada.

– **AbstractRate.** Refere-se às informações relacionadas à avaliação de produtos baseadas nos gostos do usuário. Assim como os outros, requer a implementação do método `setJSONonInstance(JSONObject object)` para fazer o parse do JSON.

– **AbstractUser.** Assim como o **AbstractProduct**, é uma das principais classes da aplicação, controlando os usuários que da aplicação tiram proveito. Como os outros, requer a implementação do método `setJSONonInstance(JSONObject object)` para fazer a conversão do JSON em user, do método `getUpdateRequestParams()` para obter a instância de RequestParams necessária para o update do usuário no servidor e, por fim, a implementação do método `getCreationRequestParams()` para obter a instância de RequestParams para analisar os parâmetros a serem enviados via POST para o servidor.

– **AbstractRanking.** Refere-se às informações referentes ao Ranking de um usuário. É utilizada como mecanismo básico de incentivo implementado neste *framework*. Seu uso está diretamente relacionado com a classe **AbstractUser**, pois ela guarda, a um primeiro momento, a informação necessária para determinar o Ranking do usuário em questão. Caso a aplicação utilize outra forma de incentivo e reputação, é totalmente possível desconsiderar a utilização deste auxílio, possibilitando inclusive remover o atributo referente à pontuação do usuário.

### 4.6.2 Managers

Os *Managers* são os gerenciadores dos métodos básicos da aplicação. São responsáveis por executar as requisições que não dependem da implementação do desenvolvedor. Com eles, é possível executar login na aplicação, requisitar informações das redes sociais, de geolocalização, entre outros. São importantes para separar aquilo que deve ser modificado no ato do desenvolvimento da aplicação do que já é nativo e disponibilizado para uso sem qualquer alteração.

#### 4.6.2.1 SocialManager

A classe *SocialManager* é a classe que gere todas as interações da aplicação com as redes sociais. Trabalha de forma simples em conjunto com a API para desenvolvedores disponibilizada pelas próprias plataformas. Possui métodos que podem ser acessados através de uma instância da classe, conseguida diretamente através da chamada estática ao método `getInstance()`. Os métodos disponibilizados através da instância tratam

o login do usuário e a listagem de amigos. A seguir, listamos estes métodos e como eles trabalham.

- **requestUserLogin.** Recebe como entrada o listener da classe `OnRequestUserInfoListener`. O método faz uma requisição através da API da rede social, abrindo inclusive uma tela exclusivamente para o usuário informar os dados de sua conta, caso não esteja logado na rede social, e retorna as informações do mesmo através do listener, por processar a informação de forma assíncrona.
- **requestFriends.** Recebe como entrada o listener da classe `OnRequestFriendsListener`. Utiliza a sessão do usuário atualmente logado para fazer as requisições por amigos, retornando de forma direta, através do listener, todos os usuários que utilizam ou não a aplicação em listas separadas.

#### 4.6.2.2 LocationManager

A classe LocationManager é responsável por gerenciar a pesquisa de estabelecimentos presentes no serviço de localização. Possibilita acesso aos métodos através da chamada estática ao método `getInstance()`. Nesta versão, o serviço de localização e de estabelecimentos é o foursquar e seu único método traz uma forma de obter a lista de estabelecimentos através de parâmetros que são passados para a API. O método **getVenues** recebe como entrada o nome do estabelecimento, a latitude e a longitude, o raio de busca e o listener de retorno. Todos os parâmetros, com excessão do listener são opicionais, mas devem ser utilizados para aumentar a precisão da busca de locais. De qualquer forma, como resposta, temos a lista de estabelecimentos que preenchem as condições da query enviada ao servidor do foursquare. Importante notar que existem dois parâmetros que devem ser adicionados ao Strings.xml da aplicação principal e que serão utilizados para, juntamente com a query, realizar a requisição no momento certo. Estes parâmetros são: framework_configuration_foursquare_venue_category_id, que informa a categoria do estabelecimento a ser pesquisado, e framework_configuration_foursquare_oauth_token, que informa o token de acesso à plataforma, disponibilizado pelo foursquare.

#### 4.6.2.3 LanguageManager

A classe LanguageManager gerencia a configuração da linguagem da aplicação a ser utilizada pelo usuário. Normalmente, o Android utiliza a linguagem padrão do sistema para fazer a internacionalização por aplicação. No entanto, não é possível gerenciar

esta configuração de forma fácil e em tempo de execução da aplicação. Sendo assim, o LanguageManager trata formas alternativas de alterar o idioma da aplicação de forma dinâmica, sem o tratamento padrão implementado pelo sistema. Os métodos presentes nesta classe dizem respeito à mudança e carregamento de recursos de determinado idioma presente nos recursos da aplicação:

- **loadLanguageResources.** Recebe como entrada o contexto da aplicação em questão. Este contexto será utilizado para carregar os recursos da aplicação em si, carregando a linguagem definida pelo método a seguir e atualizando as informações na presente tela do usuário.
- **changeLanguageResource.** Recebe como entrada o enum LanguageEnum (LANGUAGE_PORTUGUESE, LANGUAGE_ENGLISH, LANGUAGE_SPANISH) referente à nova linguagem da aplicação e o fragmento herdado de FrameworkFragment atualmente ativo na aplicação. O método altera a configuração do recurso da aplicação e atualiza o conteúdo exibido em tela para o usuário final.

#### 4.6.2.4 SessionManager

A classe SessionManager gerencia o usuário atualmente logado na aplicação. Guarda informações sobre a instância de **AbstractUser** e sobre o tempo de permanência do usuário no sistema através do armazenamento da variável loggedUserSessionStart, que informa o momento no qual o usuário foi adicionado à sessão. Possui métodos diretamente relacionados às suas funções, sem a execução de tarefas extras:

- **getSessionUser.** Não recebe nada como entrada. Apenas tem como retorno a instância de **AbstractUser** atualmente logada no sistema.
- **closeSession.** Não recebe nada como entrada. Apenas limpa as variáveis referentes à sessão atual.
- **getSessionStartTimeInMs.** Não recebe nada como entrada. Apenas retorna o *timestamp* do momento no qual o usuário foi adicionado à sessão ativa.
- **setSessionUser.** Recebe como entrada uma instância de **AbstractUser** referente ao usuário logado no sistema. Este método define o usuário logado na aplicação e informa o *timestamp* no qual o usuário entrou.

#### 4.6.2.5 TitleNotificationManager

A classe TitleNotificationManager gerencia notificações geradas pela aplicação colaborativa. Contém métodos mais simples de adição e remoção de notificações na barra de status do sistema. Seus métodos referem-se à criação e alteração dos parâmetros da notificação para criar um padrão entre as informações geradas para os usuários da aplicação:

- **removeNotification.** Recebe como entrada o ID da notificação a ser removida. Sua função é remover a notificação da barra de status.
- **removeAllNotifications.** Não recebe nada como entrada. Sua função é remover todas as notificações geradas pela aplicação.
- **beginNotificationTransaction.** Não recebe nada como entrada. Sua função é iniciar uma transação para a adição ou remoção de parâmetros à notificação a ser adicionada. Como retorno, recebe um objeto do tipo `Builder`, referente à construção da notificação. Este objeto deve ser guardado e informado na utilização dos métodos de alteração dos parâmetros da notificação em questão.
- **setSound.** Recebe como entrada a instância de **Builder**, referente à notificação, e a URI para o som a ser executado quando a notificação for criada. Este método adiciona a variável referente ao som executado à notificação original.
- **setIcon.** Recebe como entrada a instância de **Builder**, referente à notificação, e o id do recurso para o ícone. Este método adiciona a variável referente ao ícone à notificação original.
- **setText.** Recebe como entrada a instância de **Builder**, referente à notificação, o título, o texto e o texto expandido. Este método adiciona as variáveis de título, texto e texto expandido à notificação original.
- **setIntent.** Recebe como entrada a instância de **Builder**, referente à notificação, e a PendingIntent, referente à intenção a ser executada ao tocar na notificação. Este método adiciona uma ação ao toque na notificação enquanto esta permanecer na barra de status do sistema.
- **show.** Recebe como entrada a instância de **Builder**, referente à notificação, o ID da forma de remoção da notificação da barra de status e um booleano informando se, ao criar a notificação, deve-se vibrar o dispositivo. Este método adiciona a notificação criada a partir da transaction e da adição das variáveis ao longo do tempo e a exibe para o usuário.

#### 4.6.2.6 WebManager

A classe WebManager gerencia todas as conexões diretas com o servidor da aplicação. Como veremos na seção 4.6.3, diversos dos Módulos utilizados no *Framework* fazem uso de métodos presentes neste gerenciador para enviar e receber as respostas 'cruas' das requisições. Estas respostas serão recebidas na forma de JSON e passadas um nível acima, para os módulos da aplicação. Além de oferecer formas padrão de conexão com o servidor, este manager auxilia, ainda, na criação de novas conexões, utilizando um método genérico capaz de, através da URL e de uma instância de RequestParams, contendo os dados para serem enviados via POST ou GET, fazer requisições e obter respostas a serem tratadas diretamente pelo método que acabara de chamar o método.

### 4.6.3 Módulos do *Framework*

Com base nas características e funcionalidades analisadas ao longo da pesquisa por trabalhos relacionados e informadas na seção anterior, dividiu-se o *Product Hunter Framework* em módulos específicos e com características similares. Cada um destes módulos, utiliza objetos referentes ao que é utilizado pela aplicação, tendo como elementos básicos, classes abstratas de usuários, produtos, localização, avaliação e preço. Para a chamada de cada um dos módulos, é necessário que o desenvolvedor indique a este qual a classe que herda informações dos elementos abstratos e que será utilizado pelos módulos para fazer requisições e obter as respostas vindas do servidor. O módulo de usuário, por exemplo, utiliza o método `UserModule.setUserClass(Class<? extends AbstractUser) uClass` para definir qual a classe advinda da aplicação que estiver utilizado o *framework* será a responsável pelos dados a serem enviados e recebidos/tratados pela aplicação. Já o módulo de produto, utiliza o método `ProductModule.setProductClass(Class<? extends AbstractProduct) pClass` para a mesma definição, só que agora relacionando-se com os produtos que a aplicação armazena. Como módulos básicos, o *framework* traz a implementação dos módulos de usuário (UserModule), produto (ProductModule) e local (PlaceModule), além de integrações com redes sociais e entre os usuários da comunidade através dos módulos de amizade (FriendshipModule) e de redes sociais propriamente dito (SocialNetworkModule). Todos estes módulos contém métodos importantes, que fazem e tratam a requisição enviada e respondida pelo servidor da aplicação, de forma totalmente transparente, devolvendo ao usuário respostas limpas e necessárias para que este deixe um pouco de lado questões de back-end.

Para facilitar a codificação e reduzir o tempo de desenvolvimento, a resposta de cada uma das requisições já é completamente processada no interior de cada método que por ventura seja chamado pela aplicação principal. No entanto, como estas requisições são feitas de forma assíncrona, é necessário um mecanismo que esteja atento aos eventos decorrentes do envio e do recebimento da requisição, sendo possível que, desta forma, a aplicação seja notificada quando alguma informação útil for recebida. Para isso, cada um dos módulos implementados neste trabalho utiliza uma técnica de controle da requisição baseada em interfaces, que são implementadas na aplicação principal, mas que já possuem, de forma transparente, a resposta necessária para a requisição feita ao módulo. Sendo assim, a resposta da chamada de determinado método presente em um dos módulos é dada na forma de chamada de um método presente na interface implementada no momento em que o método for chamado. Isso garantirá que as requisições sejam feitas de forma assíncrona e fora da main thread, não interferindo no andamento da aplicação e sem perder informação decorrente de tratamento incorreto da thread executora da requisição.

A seguir, lista-se de forma detalhada o que cada um dos módulos é capaz de fazer, demonstrando sua finalidade e como funciona a chamada de métodos e o tratamento das respostas.

#### 4.6.3.1 Módulo de Usuário

O módulo de usuário é responsável por resolver questões relacionadas à cadastro, login, recuperação de informações do mesmo, armazenamento de sessão, recuperação de informações, comunicação direta de informações referentes à sessão com o usuário e, por fim, logout da sessão. Estas funcionalidades estão atreladas a uma classe que herda características básicas determinadas pela classe AbstractUser, que é uma classe presente no *framework* de forma a abstrair métodos e atributos comuns a todos os usuários da aplicação, ficando a cargo do desenvolvedor da aplicacão final fazer alterações e adicionar comportamentos e informações sobre o usuário de sua comunidade.

As funcionalidades mais utilizadas e requisitadas pelos desenvolvedores são as que dizem respeito ao tratamento do perfil, bem como o gerenciamento de contas do usuário. Este tratamento se dá através de métodos referentes ao cadastro, login, edição de perfil e logout. Todos eles estão diretamente relacionados e atrelados a unica e exclusivamente um usuário por sessão, considerando não ser possível utilizar mais de um usuário ao mesmo tempo ao longo de uma aplicação colaborativa, com razões suficientes para requisitar uma relação direta com o usuário em questão.

Assim como grande parte das aplicações moldadas para fins colaborativos, este

*framework* visa a integração por completo de todas as formas de integração entre seus usuários e ferramentas sociais presentes no ambiente da internet. Sendo assim, juntamente com o módulo de usuário, implementou-se métodos relacionados com a busca de informações de um perfil presente em uma rede social, neste caso o Facebook, para a cópia e consequente criação de um perfil de usuário baseado no que já fora informado por este em uma rede social mais ampla e completamente aceita pela sociedade atual. Esta integração, a um primeiro momento, é feita de forma simples e em conjunto com métodos implementados no módulo de Redes Sociais (como veremos na sessão 4.6.3.3). O usuário é capaz de se cadastrar no sistema e, posteriormente, logar no mesmo apenas utilizados informações presentes na rede social, ajudando o sistema a conhecê-lo melhor, sem a necessidade de um novo cadastro e novas informações.

A seguir, vemos uma lista de métodos providos pelo arcabouço e as explicações necessárias para implementá-los.

- **registerUser.** Recebe como entrada uma instância de **AbstractUser** e a instância do **listener**. Este método é responsável por cadastrar os usuários no servidor, de forma que os parâmetros básicos de cadastro podem ser informados pelo desenvolvedor, já que a classe AbstractUser requer a implementação de um método que retorne uma instância de RequestParams, que nada mais é do que os atributos do usuário que serão enviados para o servidor juntamente com a requisição. O listener escuta pela finalização do processamento da requisição, retornado ao método que chamou o módulo com o objeto do usuário criado informado pelo servidor.

- **loginUser.** Recebe como entrada uma instância de **AbstractUser** e a instância do **listener**. Este método é responsável por enviar os dados de login do usuário informado e obter como resposta as informações do mesmo para armazenamento na sessão. Ao final do processamento da requisição, o módulo responde para o módulo que o chamou de forma a informar qual usuário acabou de ser logado na aplicação.

- **updateUser.** Recebe como entrada uma instância de **AbstractUser** e a instância do **listener**. Este método é responsável por atualizar o cadastro dos usuários no servidor, de forma que os parâmetros básicos são informados pelo desenvolvedor, já que a classe AbstractUser requer a implementação de um método que retorne uma instância de RequestParams, que nada mais são do que os atributos de atualização passados para o servidor em conjunto com a requisição propriamente dita.

– **logoutUser.** Não recebe nada como entrada. Sua única função é limpar a variável de sessão que antes armazenada informações a cerca do usuário logado.

– **getSessionUser.** Não recebe nada como entrada. Sua função é retornar as informações usuário atualmente ativo na sessão.

– **loginFacebookUserIfRegistered.** Recebe como entrada a **ID do facebook**. Este método utiliza a ID do facebook do usuário para verificar a existência, ou não, da mesma na base de dados de usuários. Este ID é fornecido pela própria API do facebook e pode ser encontrada através do módulo de redes sociais. Caso o ID não esteja presente na base, o método retorna a mensagem de erro e informa à aplicação a necessidade de um cadastro.

#### 4.6.3.2 Módulo de Produtos

O módulo de produtos é responsável por adicionar, listar, avaliar e enviar informações referentes aos produtos presentes na base de dados da aplicação, além de tratar a adição, remoção e listagem em listas comuns em aplicações colaborativas, conhecidas como listas de desejos, ou *wish lists*. Este módulo provê, ainda, métodos relativos à busca de produtos no servidor por termos presentes no nome, na marca ou no código de barras, atributos dados como básicos em aplicações envolvendo produtos em aplicações deste tipo. Além disso, é possível, ainda, fazer buscas por imagens, que são enviadas ao servidor e analisadas de forma a informar à aplicação o(s) produto(s) que se assemelham com a imagem recém enviada, assim como visualizou-se ao analisar o Vivino (seção 3.2.2).

A função principal das listas de desejos é de informar a outros usuários, que, por ventura, visualizarem o perfil de outra pessoa, sobre os produtos desejados por outros, analisando a semelhança de gostos e de características entre os produtos com os gostos pessoais do perfil visualizado. Isso ajuda a trazer confiabilidade na escolha de um produto que ainda não fora analisado pelo usuário em questão, mas que pôde ser visto através do perfil de alguém conhecido.

A seguir, vemos uma lista de métodos providos pelo arcabouço e as explicações necessárias para implementá-los:

– **addProduct.** Recebe como entrada uma instância de **AbstractProduct** e a instância do **listener**. Este método é responsável por cadastrar um novo produto no servidor, de forma que os parâmetros básicos de cadastro podem ser informados pelo desenvolvedor, já que a classe AbstractProduct requer a implementação de um método que retorne uma instância de RequestParams, que nada mais é do

que os atributos do produto a serem enviados para o servidor juntamente com a requisição. O listener escuta pela finalização do processamento de cadastro, retornado ao método que chamou o módulo o objeto do produto recém criado no servidor ou a mensagem de erro informada no decorrer do processo.

- **addProductToWishList.** Recebe como entrada uma instância de **AbstractUser**, uma instância de **AbstractProduct** e a instância do **listener**. Este método é responsável por enviar ao servidor as informações necessárias para a adição de um determinado produto à lista de desejos do usuário presente no sistema. Ao final do processo, o método responde para o módulo que o chamou, através do listener, de forma a informar o sucesso ou a falha do processo.

- **remProductFromWishList.** Recebe como entrada uma instância de **AbstractUser**, uma instância de **AbstractProduct** e a instância do **listener**. Este método é responsável por enviar ao servidor o usuário que possui o produto em sua lista de desejos de forma a eliminá-lo de tal. O retorno se dará através do listener, que informará o sucesso ou a falha da requisição.

- **searchProductByImage.** Recebe como entrada uma **String** referente a uma imagem decodificada em BASE64 e uma instância de **listener**. Este método envia ao servidor o arquivo como uma string, para ser transformado em arquivo novamente no servidor e posteriormente comparado com as imagens de produtos presentes no banco de dados. Estas imagens passam por um processo de análise de similaridade e um ou mais produtos com imagens semelhantes (ou a falta deles) são retornados para a aplicação através das chamadas presentes no listener do método.

- **searchProductsByTerm.** Recebe como entrada uma **String** referente ao termo da busca e uma instância de **listener**. Este método envia ao servidor o termo requisitado e retorna, através do listener, a informação sobre os produtos encontrados ou a mensagem de falha da requisição, seja ela por falta de produtos ou por erros decorrentes na própria requisição.

Além dos métodos supra citados, acabou-se decidindo implementar também métodos que trabalhem com a lista de produtos visualizados pelos usuários, fazendo com que outras pessoas saibam dos produtos procurados por pessoas com perfis semelhantes, a fim de saber a tendência de produtos instaurada através do momento. Estes métodos referem-se basicamente à adição e remoção de produtos em listas de visuali-

zação de usuários, seguido por um método de listagem de produtos na lista de desejos e na lista de visualizações:

- **addProductToViewList.** Assim como **addProductToWishList**, este método recebe como entrada uma instância de **AbstractUser**, uma instância de **AbstractProduct** e uma instância do **listener**. É responsável por adicionar o produto informado à lista de produtos visualizados pelo usuário em questão. O listener responde ao método sobre sucesso ou falha na requisição.

- **remProductFromViewList.** Também como o **addProductToWishList**, este método recebe como entrada uma instância de **AbstractUser**, uma instância de **AbstractProduct** e uma instância do **listener**. Sua função é remover do servidor a informação referente à visualização de um determinado produto por parte do usuário em questão. O listener é o responsável por retornar as mensagens de sucesso ou de falha ao método principal.

- **getWishOrViewListProducts.** Recebe como entrada um boolean informando se este método refere-se à lista de desejos (verdadeiro/true) ou à lista de visualizações (falso/false), uma instância de **AbstractUser** e uma instância do listener. O booleano serve para informar qual das requisições será feita ao servidor, assim como o AbstractUser informará a qual usuário pertence as listas com as quais a busca será feita. Por fim, a resposta advinda do servidor em forma de lista de produtos é retornada e passada ao método principal através do listener.

#### 4.6.3.3 Módulo de Redes Sociais

O módulo de Redes Sociais (SocialNetworkModule) é utilizado para o tratamento de requisições em conjunto com os Managers de tratamento de informações do Facebook (FacebookManager) e do Foursquare (FoursquareManager), as duas redes sociais integradas com o *Framework* até o final desta versão do trabalho. Com isso, os métodos presentes nos managers estão intimamente relacionados com as funções disponibilizadas pelos módulos, já que disponibilizam acesso fácil à lista de amigos, de locais e a informações do usuário da rede social. Desta forma, listam-se abaixo os métodos que podem ser utilizados ao necessitar acesso à informações dos módulos:

- **getUserInfoFromFacebook.** Recebe como entrada apenas o listener da requisição, instaciando a classe `OnRequestUserInfoListener` e implementando o método os métodos `onSuccess(GraphUser user)`, que retorna um usuário do facebook e `onFail(String response)`, que retorna a mensagem de falha.

– **getFriendsFromFacebook.** Assim como o método anterior, recebe como entrada o listener da requisição, agora tendo como classe instanciada a classe `OnRequestFriendsListener`, sendo necessário implementar os métodos `onSuccess(List<GraphUser> usersUsing, List<GraphUser> usersNotUsing)`, que retorna uma lista de usuários que utilizam a aplicação e uma lista de usuários que não utilizam a aplicação, e `onFail(String response)`, que retorna a mensagem de falha.

– **getVenues.** Recebe como entrada apenas o listener e lê as informações, necessárias para a pesquisa, contidas no arquivo Strings.xml (category_id e oauth_token). Para a instanciação do listener (`OnVenuesSearchListener`), é necessário implementar os métodos `onVenuesFound(List<Venues> venuesList)` e `onVenuesNotFound()` que informam, respectivamente, os estabelecimentos encontrados ou a falta deles. É possível ainda, mas não obrigatoriamente, implementar o método `onVenuesRequestError()`, responsável por informar falha na busca.

#### 4.6.3.4 Módulo de Amizade

Este módulo é responsável por tratar as interações entre os usuários da comunidade. Traz métodos relacionados à busca de amigos, pesquisa de usuários e opções de adicionar amigos. Tem como base que a criação de uma ferramenta colaborativa requer a interação entre os usuários, bem como a criação da ligação entre eles baseada na adição ou remoção de amigos presentes na rede.

– **getFriends.** Recebe como entrada uma instância de **AbstractUser**, referente ao usuário ativo na sessão e o listener. Como resposta, obtém-se a lista de usuários que já são amigos do usuário ativo.

– **getFriendsCandidate.** Recebe como entrada uma instância de **AbstractUser**, referente ao usuário da sessão e o listener. Como resposta, obtém-se a lista de usuários que fizeram um pedido por amizade para o usuário da sessão ou a mensagem de falha na requisição.

– **requestFriendship.** Recebe como entrada duas instâncias de **AbstractUser**, uma referente ao usuário da sessão e outra ao usuário a ser adicionado, e o listener. Como resposta, através do listener, o método informará o sucesso ou a falha da requisição, informando à aplicação sobre a necessidade de aceitação da amizade por parte do usuário a ser adicionado.

– **confirmFriendship.** Recebe como entrada duas instâncias de **AbstractUser**, uma referente ao usuário da sessão e outra ao usuário a ser aceito como amigo, além do listener. Como resposta, informa-se o sucesso ou a falha da requisição.

– **removeFriendship.** Recebe como entrada duas instâncias de **AbstractUser**, uma referente ao usuário da sessão e outra ao usuário a ser removido da lista de amigos, além do listener de retorno. Como resposta, o listener informa sobre o sucesso ou a falha da requisição.

#### 4.6.3.5 Módulo de Locais

O módulo de locais tem uma missão mais simples que os demais. Seu trabalho é informar à aplicação quais os locais que vendem determinado produto. Esta informação é acessada através do método **getPlacesSellingProduct**, que recebe como entrada uma instância de **AbstractProduct**, referente ao produto a ser pesquisado nas lojas presentes na base de dados, e o listener. Como resposta, obtém-se a lista de estabelecimentos que possuem o produto em questão.

#### 4.6.3.6 Módulos de Incentivo e Reputação

A criação de um módulo para incentivo e reputação dos usuários da comunidade é amplamente recomendada. Neste trabalho, traz-se a implementação de métodos básicos, que podem ser utilizados para prover incentivo através da reputação dos usuários. Os métodos básicos criam um ambiente onde os usuários ganham pontos e com isso sobem de nível e podem começar a executar tarefas que antes não poderiam por passar a criar uma maior confiança para com os outros usuários colaboradores da aplicação:

– **getUsersRanking.** Recebe como entrada apenas o listener, responsável por retornar a lista de usuários e suas devidas pontuações no ranking, bem como o título dado a classificação do usuário.

– **getUserRanking.** Recebe como entrada apenas uma instância de **AbstractUser**, referente ao usuário ativo na sessão, e uma instância do listener de retorno. O listener retorna, ao método da aplicação principal, as informações de pontuação e o ranking atual do usuário solicitante da informação.

– **addPoints.** Recebe como entrada uma instância de **AbstractUser**, referente ao usuário ativo na sessão, a pontuação ganha ou perdida e o listener de retorno. O método se torna, então, responsável por enviar as informações ao servidor de

forma a incrementar ou decrementar a pontuação corrente do usuário da sessão ativa, retornando o sucesso ou a falha do envio, bem como as informações referentes ao novo ranking do usuário em caso de sucesso.

## 4.7 Portabilidade do *Framework*

Este *framework* foi desenvolvido, neste primeiro momento, como prova de conceito, utilizando uma plataforma desenvolvida e muito utilizada tanto por desenvolvedores quanto por usuários comuns. No entanto, é possível portá-la para qualquer outro sistema e, para isso, basta observar a modelagem, as funcionalidades e a descrição de suas funcionalidades ao longo do capítulo. Se as funcionalidades forem implementadas da mesma maneira, é possível expandir a criação de aplicações para diferentes dispositivos, sendo possível incluir sistemas como o iOS e o Windows Phone, por exemplo. Com isso, mostramos que o *framework* pode ser considerado um modelo de plataforma de desenvolvimento, não estando, de forma alguma, atrelado à uma ou outra plataforma em específico, permitindo a utilização de diferentes plataformas com o intuito de realizar cada vez mais coletas, de cada vez mais usuários e, com isso, obter uma melhor forma de colaboração dentro de um sistema de *crowdsensing*.

## 4.8 Eficiência

Para citar a eficiência obtida com a criação desta ferramenta, podemos relacioná-la com duas características: Eficiência Econômica e Eficiência de Tempo. A eficiência econômica diz respeito tanto à economia de gastos referentes ao processo de produção de uma aplicação, quanto à eficiência de consumo de bateria e de processamento referente ao produto final. A Eficiência de Tempo refere-se a redução no tempo necessário para a criação de uma aplicação colaborativa nos moldes propostos por este projeto. Em Sherchan et al. (2012), propôs-se a utilização de formas de controlar a energia gasta pela aplicação baseando-se em métodos que reduzissem a quantidade de informações enviadas. Como a base de nossa ferramenta refere-se à criação e análise de requisições, fizemos de tudo para que estas fossem menos custosas para os recursos do dispositivo. Sendo assim, para analisar o ganho de eficiência energética, observamos que a utilização do JSON em substituição ao Simple Object Access Protocol (SOAP) reduz o tráfego de rede e o tempo e processamento relacionado à análise da resposta.

De acordo com o experimento feito por Belqasmi et al. (2012), que analisa um sistema baseado na diferença entre o tempo gasto e o tamanho da resposta obtida por

diferentes funcionalidades de uma aplicação. Para isso, fez-se uma série de requisições e se obteve o tempo gasto para cada uma delas. Feito isso, observou-se que o tamanho dos dados da resposta obtidos utilizando JSON era 2,53 vezes menor em comparação às mesmas requisições feitas obtendo como resultado SOAP, sem falar no ganho de tempo de processamento dessa informação e, por fim, o tempo entre a requisição e a resposta enviada pelo servidor.

Com estes dados, provamos que a eficiência de nosso *framework* foi alcançada ao mostrarmos que a utilização de JSON é mais rápida e menos custosa para a aplicação, no que diz respeito ao tráfego de rede, consumo de energia para processar o conteúdo e redução do consumo de dados móveis em função da redução do tamanho do documento recebido do servidor.

## 4.9 Product Hunter vs Outros *Frameworks*

Em nossa abordagem, nota-se a busca por funcionalidades e ferramentas capazes de dar ao desenvolvedor, de aplicações colaborativas com foco nas informações e avaliação de um determinado produto, uma maior facilidade no que diz respeito à implementação de métodos que serão utilizadas em muitas das aplicações da área. As funcionalidades observadas e características retiradas de outras aplicações e *frameworks* relacionam-se com a busca pela facilitação do desenvolvimento e, sobretudo, a criação de campanhas e funcionalidades que permitam que a colaboração ocorra. No entanto, nenhum destes *frameworks* relacionou e fez estudos profundos acerca das funcionalidades realmente utilizadas por aplicações que vão ser desenvolvidas a partir da ferramenta. Em nosso caso, muitas de nossas implementações baseiam-se em tendências do mercado e da Internet, o que possibilita uma gama muito grande de situações que podem ser implementadas de forma simples na aplicação final.

Além disso, observamos toda a questão da arquitetura de forma a prover total comunicação entre servidor e dispositivo móvel, possibilitando que ambos conversem de forma direta e "amigável", relacionando bem as requisições solicitadas com as que foram projetadas especificiamente para atendê-las. Criamos uma série de camadas responsáveis por prover ao desenvolvedor um maior entendimento de como utilizar as funcionalidades e, se não fosse o bastante, possibilitamos que este faça alterações nos dados armazenados pela aplicação, adicionando ou removendo atributos e tratando-os da forma que mais lhe for conveniente.

No ponto de vista da eficiência, ponto esse que se tomou como base para nossos objetivos, observamos que o proposto foi atingido, já que todas as requisições básicas

são feitas da forma mais leve possível, considerando os temas que estão mais em alta na atualidade. A utilização do JSON possibilita uma forma fácil de obter e tratar as informações, sendo possível, para a aplicação, tratá-las da melhor forma possível, sem se preocupar com o overload de informações referentes à requisição em si.

Todos os outros *frameworks* dão muita atenção à busca de informações através da utilização do *crowdsensing*, mas não demonstram preocupação direta com os temas abordados na atualidade de forma a trazer ao usuário um ganho real de performance energécia e nem de instanciação do *framework* de forma a prover um meio mais rápido de desenvolvimento.

## 4.10 Comentários Parciais

A partir deste momento, temos uma ferramenta que pode ser facilmente aprimorada e utilizada para o desenvolvimento de aplicações de forma rápida e prática. Para isso, algumas decisões de projeto precisaram ser tomadas e isso reflete diretamente na aplicação que utilizar a ferramenta. As funcionalidades desejadas foram implementadas em quase toda a sua totalidade, no entanto, algumas delas foram reduzidas, por questões de tecnologia e tendências da época, mas ainda assim atenderam ao escopo inicial. São elas:

- **Integração com Redes Sociais.** Foi implementada de forma a possibilitar que os desenvolvedores utilizem seus principais métodos de forma simples. No entanto, para fins de tecnologia e do momento atual das redes sociais, no que diz respeito à utilização de uma ou de outra. Sendo assim, escolheu-se implementar apenas as funcionalidades do Facebook, que conta com cerca de 73,5% do total de usuários, contra 1,15% de usuários do Google+ no Brasil, que é um dos países que mais utilizam redes sociais (eMarketer, 2013a).

- **Georeferenciamento.** Foi implementado de forma a trazer os estabelecimentos presentes no foursquare, através de chamadas à sua API.

- **Separação entre Managers e Módulos.** A separação foi feita com o intuito de facilitar o entendimento de ambas por parte do desenvolvedor, que pode concentrar seus esforços nos módulos, que podem ser modificados de acordo com suas necessidades.

# Capítulo 5

# Estudo de Caso: DiapersHunter

Este estudo de caso refere-se à utilização do *framework* proposto na seção 4 para a criação de uma aplicação, por parte do desenvolvedor, que possibilite ao usuário participar de uma comunidade colaborativa para a troca de informações de produtos como nome, marca, preço, locais de venda e avaliação, de forma a provar a real utilização da ferramenta, possibilitando ganho de tempo no desenvolvimento da aplicação.

Nesta seção, será apresentado o DiapersHunter, um protótipo de uma aplicação que surge como estudo de caso de uma instanciação direta das facilidades obtidas através da utilização de um *framework* direcionado para técnicas de construção de aplicações colaborativas com foco na busca e avaliação de produtos, através da interação entre os usuários. Sendo assim, esta demonstração se destina a um produto bem específico: Fraldas (*Diapers*). As fraldas descartáveis são produtos que despertam grande interesse em usuários que estão para ter ou acabaram de ter um filho e ainda não estão familiarizados com as diversas marcas encontradas no mercado e, para isso, necessitam da ajuda de terceiros para receber informações sobre produtos, conhecerem novas marcas, saber a avaliação do produto através de informações enviadas por outros usuários, descobrir preços e locais de venda e, por fim, montar uma base de produtos que agrade o bolso, a comodidade e outras questões pessoais ligadas ao produto.

Nas próximas seções, será explicada toda a etapa de modelagem do sistema, dicas para um melhor aproveitamento da ferramenta, bem como formas de utilizá-la do jeito correto.

## 5.1 Modelagem da Aplicação

Antes de começar a desenvolver uma aplicação em cima do arcabouço implementado, é necessário que se faça uma boa modelagem dos pontos chaves da aplicação em questão.

Basicamente, o escopo deve ser traçado de forma a abraçar todo o conteúdo que se deseja inserir no sistema de acordo com a melhor forma de fazê-lo. Para isso, um bom passo inicial é verificar quais as necessidades da aplicação. É nessa fase que se decide sobre a necessidade ou não do registro de usuários, da utilização anônima, da forma de impor compromisso do usuário para com a aplicação e a comunidade que dela participa.

Nesta aplicação, define-se diversos fatores que culminarão com a utilização de diversas características já implementadas no *framework* proposto neste trabalho. São eles:

- **Usuários Registrados.** A obrigatoriedade, por parte dos usuários, de se cadastrarem no sistema antes de ter acesso às informações presentes e, consequentemente, contribuir com informações coletadas se faz presente para preservar, de forma criteriosa, a honestidade das avaliações e coletas feitas por cada um dos presentes na comunidade colaboradora. Aplicações que permitem a anonimidade total de seus usuários tendem a fracassar pela falta de veracidade nas informações presentes em seus bancos de dados e este definitivamente não é o foco desta aplicação.

- **Anonimidade Espacial.** A possibilidade de se tornar anônimo, no que diz respeito à localização e dados de georeferenciamento, é completamente aceitável e deve estar, de forma clara, implementado como possibilidade de o usuário escolher ou não sobre a sua privacidade.

- **Interação entre Usuários.** A interação entre os usuários, em uma aplicação colaborativa, é o que faz o sistema continuar sendo auto gerenciável do ponto de vista da adição e atualização das informações nele inseridas. A necessidade de interação se mostra presente ao se analisar a quantidade de usuários presentes nas mais diversas redes sociais da atualidade. Esta característica demonstra a capacidade de, através da quantidade, se obter uma base sólida baseada nas coletas e informações enviadas por usuários de forma pró-ativa.

- **Integração com Redes Sociais.** A integração entre os usuários da aplicação com perfis advindos de outras redes sociais, de forma rápida e integrada, é completamente requerida para fins de melhorias trazidas através da interação entre os usuários atuais e os usuários que, por já estarem presentes em outras redes sociais, provavelmente saberão como funciona a dinâmica de uma aplicação colaborativa e com isso, poderão se tornar mais dispostos a ajudar. Além disso, a possibilidade de obter informações de perfis de usuários presentes nestas

redes sociais é de fundamental importância para avaliar as colaborações destes usuários.

- **Adição de Amigos.** Ainda como forma de interação entre os usuários, a adição de amigos, e de usuários de uma forma geral, como pessoas às quais se interage de forma direta se mostra extremamente importante no contexto da aplicação proposta nesta seção. O conceito de adicionar uma pessoa em uma rede social está, para a maioria dos usuários, intimamente ligado ao conhecimento de parte das informações vindas desta pessoa. Desta forma, ao visualizar uma avaliação de determinado produto feita por um usuário sabidamente leviano ou com indícios de promover desordem e bagunça, claramente avaliaremos suas classificações de uma forma diferentes daquelas feitas por uma pessoa que sempre se mostrou confiável e respeitável do ponto de vista interativo.

- **Avaliações.** Poder avaliar os produtos aos quais se conhece é uma característica básica da aplicação.

- **Destaque para a avaliação feitas por conhecidos.** O conceito de amigo não é nada sem que um destaque seja dado a suas avaliações. Aplicações que mostram apenas os valores absolutos de uma avaliação, sem demonstrar, em termos específicos, qual o contexto da avaliação, não se mostram tão relevantes quanto àquelas nas quais o enfoque está diretamente ligado ao interesse do usuário. Sendo assim, ao darmos destaque às avaliações e comentários feitos por usuários que fazem, de certa forma, parte do círculo social do usuário, é interessante para trazer credibilidade às informações obtidas da aplicação.

- **Busca direta por Produtos.** A busca direta por produtos, com a possibilidade de pesquisar por nome, marca e/ou qualquer outra informação disponível para o produto, é importante para aplicações onde há uma grande base de dados e, por estas razões, uma busca é essencial para encontrar o produto desejado em pouco tempo.

- **Busca por produtos através de Imagens.** A busca de um produto através de uma foto qualquer tirada pelo dispositivo móvel é importante para trazer agilidade ao processo. Esta característica está presente em ferramentas colaborativas utilizadas por uma comunidade grande e pode ser considerada como um diferencial a ser implementado.

- **Ranking.** A utilização de uma maneira de dar pontos aos usuários ativos como forma de ajudá-los a subir de nível em um contexto social e, com isso, ganhar

reputação e confiabilidade, é extremamente importante de estar contido em uma aplicação colaborativa onde a avaliação dos produtos de forma correta e confiável é tão importante quanto a simples avaliação em si.

- **Incentivos.** Aplicações colaborativas e o Sensoriamento Participativo em si dependem e muito da motivação e incentivo a tal motivação por parte do usuário. Usuários que não estão entusiasmados com a aplicação não irão contribuir. É necessário existir uma forma de incentivo, nem que esta seja baseada no ranking do usuário para permitir ou não determinada ação.

- **Preços e Locais dos Produtos.** De nada adianta saber a avaliação de fraldas descartáveis se, quando o usuário procura um estabelecimento para comprá-la, não acha o produto desejado. Além disso, saber o preço é, muitas vezes, mais importante do que a própria qualidade do produto, para determinadas classes sociais. Desta forma, é necessário estabelecer como prioridade o acesso à estas informações de forma fácil.

Através da listagem de características necessárias, pode-se avaliar o impacto que a presença de um framework, com várias destas necessidades já implementadas e consolidadas, pode trazer para o desenvolvimento de uma aplicação baseada neste contexto.

## 5.2 Criação do Projeto

A criação do projeto segue a forma normal de criação de qualquer projeto para Android. No caso deste trabalho, definiu-se como ambiente de desenvolvimento as seguintes configurações:

- **Eclipse.** Utilizou-se esta IDE por possibilitar a utilização do plugin ADT, em combinação com outros tantos plugins já existentes. Poder-se-ia ter escolhido outras ferramentas, como por exemplo o Android Studio, que já trazia toda a integração compactada. No entanto, por motivos relativos à dificuldade e à falta de documentação desta nova ferramenta (que até o momento é beta), preferiu-se usar o eclipse.

- **Android 4.2.2.** É uma versão bem estável e difundida entre os usuários. Está presente na maior parte dos celular high-end e possui grande aceitação. No entanto, tomou-se cuidado para não limitar o uso da aplicação em celulares com versões do sistema operacionais mais ultrapassadas. Desta forma, utilizou-se uma biblioteca de suporte, disponibilizada pelo próprio Google, que permite a

execução das principais funções implementadas nos novos sistemas, em sistema mais antigos.

Estas configurações garantem uma forma de desenvolvimento padronizada, garantindo a geração de uma aplicação capaz de receber o *framework* e se unir a ele de forma uniforme, sem problemas. Neste instance, criou-se o projeto a partir da tela demonstrada pela imagem 5.1, que informa o nome da aplicação, o nome do projeto e o nome do pacote, além de definir como versão mínima o android 2.2, que pode ser utilizado graças à utilização da biblioteca de suporte, que permite o uso de fragmentos no desenvolvimento da aplicação.


**Figura 5.1.** Tela de criação do projeto preenchida.

Feito isso, separou-se os pacotes onde se começaria a desenvolver em 3 diferentes camadas, obedecendo a arquitetura MVC, formada pelas seguintes categorias de classes, seguindo o padrão da imagem 5.2 e das informações a seguir:

- **Model.** Representa a parte da aplicação que implementa a lógica de negócio, obtendo dados e convertendo-os em conceitos significativos para a aplicação. Tem função de processar, validar e associar as informações e tratar os dados obtidos.
- **View.** Representa a camada de apresentação dos dados modelados. É responsável por usar as informações disponibilizadas e produzir qualquer interface de apresentação que a aplicação necessite.

– **Controller.** Também chamada de camada de controle, é a responsável por lidar com as requisições feitas pelo usuário. É responsável por fazer a interligação entre os dados obtidos pela camada de modelo (Model) e as interfaces visuais da cama de visualização (View).

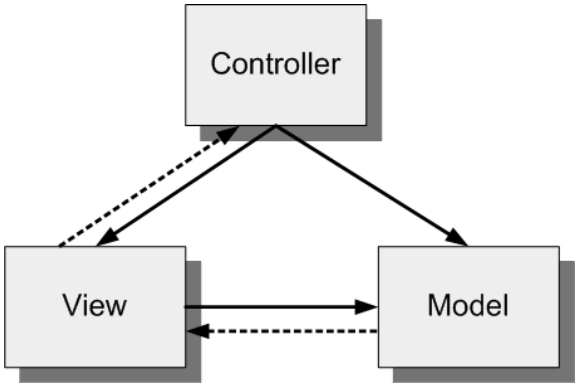


**Figura 5.2.** Arquitetura MVC.

A utilização de uma forma de desenvolvimento seguindo uma arquitetura ajuda a controlar e separar melhor o que cada uma das diferentes classes faz, ajudando no legibilidade e na reutilização de métodos importantes para o comportamento da aplicação. Sendo assim, obteve-se a separação dos pacotes como mostrado na figura 5.3. Desta forma, cada uma dos sub-pacotes possui classes relacionadas e, possivelmente, de igual direcionamento. Na camada de controller, tem-se o seguinte:

– **application.** Contém informações que precisam ser definidas na inicialização da aplicação, como, por exemplo, a instanciação de classes que serão utilizadas para controlar o cache de imagens.

– **listener.** Contém as interfaces de resposta para diversas chamadas de métodos. Os listeners são criados como interfaces para que suas funcionalidades sejam implementadas pela classe que escutar pelas ações e disparar a execução de determinada funcionalidade assim que ela ocorra.

– **view/activity.** Contém as classes destinadas a estender a classe Activity ou FragmentActivity. Neste projeto, utiliza-se o conceito de Activity como um container para a utilização dos *Fragments*. Quando necessário, é possível criar outras *activities* para utilizar ações externas à execução principal, como a chamada de uma *activity* para recortar imagens, por exemplo.

– **view/adapter.** Contém as classes destinadas a adaptar o conteúdo dos models e transformá-los em elementos necessários para a exibição dos mesmos na tela, como as *ListViews* e *GridViews*.

– **view/fragment.** Contém as classes que constroem a tela do usuário. É possível que determinada tela tenha um ou mais fragmentos.

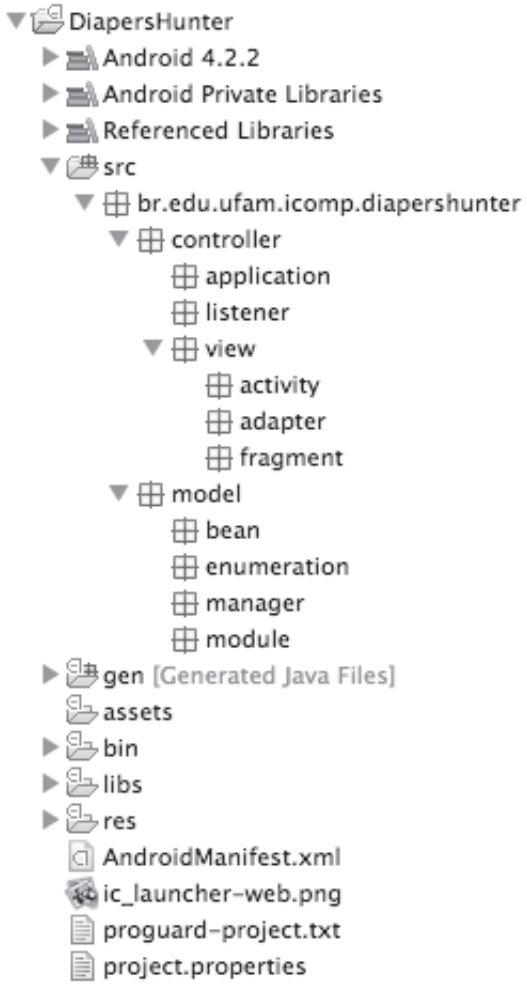


**Figura 5.3.** Visualização da hierarquia do projeto recém-criado.

Já na camada de Model, tem-se:

– **bean.** Contém as classes que herdam das classes das **AbstractClasses** e produzem ações únicas referentes à aplicação, além das já executadas pela classe abstrata.

– **enumeration.** Contém enumerations utilizadas por alguma outra classe presente na aplicação.

– **manager.** Contém os gerenciadores de alguma funcionalidade da aplicação final. Da mesma forma como se trabalha no *framework*, os managers são capazes de gerenciar ações e funções referentes à aplicação como um todo.

– **module.** Contém os módulos herdados da classe **AbstractModule** presente no *framework*. É neste pacote que estão localizados os módulos de ranking e incentivo, por exemplo.

Por fim, a camada de View é formada apenas pelos .xml que representam a interface gráfica da tela inteira, de um ponto específico ou que estarão presentes nos adapters a serem exibidos nas *ListViews* e *GridViews* da aplicação.

## 5.3 Instanciação e preparação do Framework

Para iniciar a construção da aplicação utilizando o *framework*, é necessário ligar o projeto do DiapersHunter ao *framework* em questão. A forma utilizada nesta aplicação foi simplesmente adicionar o .jar (**J**ava **AR**chive), que nada mais é do que um arquivo compactado contendo as classes e implementações disponibilizadas pelo *framework*, ao diretório '/libs' do projeto, como mostra na figura 5.4.

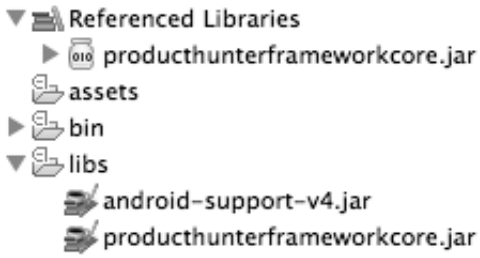


**Figura 5.4.** Adicionando a biblioteca do framework ao projeto do estudo de caso.

A partir desta etapa, inicializou-se os **Módulos** com as classes de objetos que serão utilizadas pela aplicação. Desta forma, é possível dizer a cada um dos métodos quais classes serão utilizadas em cada requisição, para que seja possível realizar a conversão dos dados obtidos a partir do servidor. Para executar esta etapa, é importante lembrar que é preciso criar Classes que referenciarão objetos relacionados com os usuários e os produtos. Desta forma, criou-se as Classes **User** (figura 5.5), estendendo **AbstractUser**, e **Product** (figura 5.6), estendendo **AbstractProduct**. Por fim, foi escolhido por definir as classes através da Application, que deve obrigatoriamente herdar de **FrameworkAppliction**, por conter métodos importantes à utilização

do módulo de conexão, através do método `getWebManagerServerBaseUrl()`, que deve retornar a URL base de todas as requisições feitas ao servidor, como mostrado na figura 5.7.

```
public class User extends AbstractUser
```

**Figura 5.5.** Exemplo da classe **User** estendendo **AbstractUser**.

```
public class Product extends AbstractProduct
```

**Figura 5.6.** Exemplo da classe **Product** estendendo **AbstractProduct**.

```
public class DiapersHunterApplication extends FrameworkApplication {

    private static DiapersHunterApplication instance;

    public static DiapersHunterApplication getInstance() {
        return instance;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;

        UserModule.setUserClass(User.class);
        FriendshipModule.setUserClass(User.class);
        ProductModule.setProductClass(Product.class);

        WebManager.getInstance().showDebug(true);
    }

    @Override
    public String getWebManagerServerBaseUrl() {
        return "http://luismenezes.kinghost.net/ParticipatoryFramework";
    }
}
```

**Figura 5.7.** Exemplo de inicialização do framework via **Application**.

## 5.4 O que o Product Hunter Framework traz?

Analisando as necessidades da aplicação, citadas na seção 5.1, pode-se inferir quais das características presentes no *framework* podem auxiliar a reduzir o tempo de programação da aplicação final. Para isso, verifica-se o que o *framework* dispõe e o que pode agregar valor no desenvolvimento. A tabela 5.1 mostra a comparação entre as

necessidades da aplicação e o que já vem implementado por padrão e disponibilizado para o desenvolvedor.

| Descrição | DiapersHunter | Framework |
|---|---|---|
| Cadastro de Usuários | Sim | Sim |
| Login de Usuários | Sim | Sim |
| Controle de Sessão | Sim | Sim |
| Anonimidade Espacial | Sim | Sim |
| Interação entre Usuários | Sim | Sim |
| Integração com Redes Sociais | Sim | Sim |
| Adição de Amigos | Sim | Sim |
| Avaliação | Sim | Sim |
| Destaque para avaliação | Sim | Não |
| Busca direta por produtos | Sim | Sim |
| Consulta de Locais | Sim | Sim (FourSquare) |
| Busca por Imagens | Sim | Sim |
| Ranking | Sim | Não |
| Incentivos | Sim | Não |
| Preços e Locais dos Produtos | Sim | Sim |

**Tabela 5.1.** Comparação entre funcionalidades do *framework* e do DiapersHunter.

Esta comparação demonstra, de forma clara, os benefícios trazidos através da utilização do produto proposto. Reduzindo o tempo de programação, pela remoção da necessidade de implementação de métodos comuns à todas as aplicações colaborativas do escopo. Sabendo disso, inicia-se o processo de criação de telas e chamada dos métodos pertencentes ao *framework* para executar as funções necessárias à aplicação.

## 5.5 Organização do Projeto e Criação da Aplicação

Obedecendo os padrões da arquitetura MVC, separamos as 3 camadas de acordo com o informado na seção 5.2. Começando pelo back-end da aplicação, implementa-se a camada de modelo e a de controle.

### 5.5.1 Model

A camada de modelo serve como base para a aplicação como um todo e é a principal geradora de dados e informações referentes ao que é visto na aplicação. No entanto, por este ser um projeto sem banco de dados, não foi necessário salvar informações e, por isso, não foi feito uso de nada relacionado a esta característica. Mesmo assim, se fosse

necessário, o *framework* já traz, como base de sua camada de modelo, as implementações feitas pelo ORMLite, uma biblioteca externa de mapeamento objeto-relacional, que traz diversas funções importantes e pré-implementadas para armazenamento e recuperação de dados no banco de dados, sem a necessidade de utilizar o SQLite puro.

Voltando à camada de Modelo, criou-se classes referentes aos objetos **User** e **Product** mostrados em seções anteriores, além de outros, referentes a outras funções aplicadas na aplicação, como, por exemplo, **Rate**, que se relaciona com as avaliações, **Ranking**, que se relaciona com o ranking do usuário, **UserRanking**, que faz o relacionamento User/Ranking, **Place**, que traz a relação entre o objeto e o local. Além disso, criou-se 2 novas classes de requisição, estendendo a classe **AbstractRequest**, presente no *framework*, que gera uma série de abstrações para trazer e tratar os dados referentes às novas requisições, como em métodos que não são nativos do WebManager, por exemplo.

No pacote de Enumerations, criou-se um Enum para controlar as formas de incentivo implementadas na aplicação. Nesta classe, foi possível definir uma mensagem padrão para determinado feito e uma pontuação pré-definida. Quando lê-se feito, deve-se pensar em ações como, por exemplo, '10 avaliações em 1 dia - 200 pontos', '5 produtos adicionados - 500 pontos' e/ou 'seguir 10 pessoas diferentes - 10 pontos'. Todas estas ações visam aumentar a interação entre produtos e usuários, aumento a visibilidade da aplicação e, consequentemente, a quantidade e qualidade de avaliações de produtos. Pode-se visualizar o enum através da figura 5.8, com os textos escritos de forma direta para facilitar o entendimento. Este enum será utilizado para adicionar pontuações ao usuário que contribuir com a aplicação, aumentando o ranking deste. Por falar em Ranking, criou-se uma tabela no servidor com a função de guardar as pontuações e títulos para cada etapa do ranking. Esta tabela contém, a princípio, os elementos da tabela 5.2

| Título | Pontuação Mínima | Pontuação Máxima |
|---|---|---|
| Iniciante | 0 | 499 |
| Aprendiz | 500 | 999 |
| Amador | 1000 | 1499 |
| Experiente | 1500 | 4999 |
| Profissional | 5000 | 14999 |
| Sênior | 15000 | 24999 |
| Rei das Fraldas | 25000 | $\infty$ |

**Tabela 5.2.** Ranking de usuários no **DiapersHunter**.

```
public enum ContributionRewardEnum {
    CONTRIBUTION_REWARD_CREATE_PRODUCT("adicionar um novo produto à base de dados", 200),
    CONTRIBUTION_REWARD_ADD_PRODUCT_PLACE("adicionar a informação de local a um produto", 50),
    CONTRIBUTION_REWARD_ADD_PRODUCT_PRICE("adicionar a informação de um preço a um produto", 10),
    CONTRIBUTION_REWARD_CONFIRM_PRICE("confirmar o preço de um produto", 10),
    CONTRIBUTION_REWARD_RATE_PRODUCT("avaliar um produto", 5),
    CONTRIBUTION_REWARD_FOLLOW_10_FRIENDS("seguir 10 pessoas diferentes", 10),
    CONTRIBUTION_REWARD_FOLLOW_25_FRIENDS("seguir 25 pessoas diferentes", 25),
    CONTRIBUTION_REWARD_FOLLOW_50_FRIENDS("seguir 50 pessoas diferentes", 50),
    CONTRIBUTION_REWARD_FOLLOW_75_FRIENDS("seguir 75 pessoas diferentes", 75),
    CONTRIBUTION_REWARD_FOLLOW_100_FRIENDS("seguir mais de 100 pessoas diferentes", 100),
    CONTRIBUTION_REWARD_RATE_5_PRODUCTS("avaliar 5 produtos", 10),
    CONTRIBUTION_REWARD_RATE_20_PRODUCTS("avaliar 15 produtos", 40),
    CONTRIBUTION_REWARD_RATE_50_PRODUCTS("avaliar 50 produtos", 100),
    CONTRIBUTION_REWARD_RATE_100_PRODUCTS("avaliar mais de 100 produtos", 100);

    private String actionRewarded;
    private int pointsRewarded;

    private ContributionRewardEnum(String actionRewarded, int pointsRewarded) {
        this.actionRewarded = actionRewarded;
        this.pointsRewarded = pointsRewarded;
    }

    // Getters and Setters...
}
```

**Figura 5.8.** Classe de enumeração de recompensas.

### 5.5.2 Controller

A camada de controle é capaz por realizar a ligação entre o que está presente na base de dados, relacionada com o Modelo, e o que está presente na interface visual da aplicação. Desta forma, utiliza-se nesta aplicação, a classe **DiapersHunterApplication**, que gerencia a inicialização dos módulos, como na seção 5.7. Além disso, criou-se uma *Activity* que gerencia os fragmentos em um container de layout específico. Os listeners servem para gerenciar a ligação entre controle e modelo e controle e view.

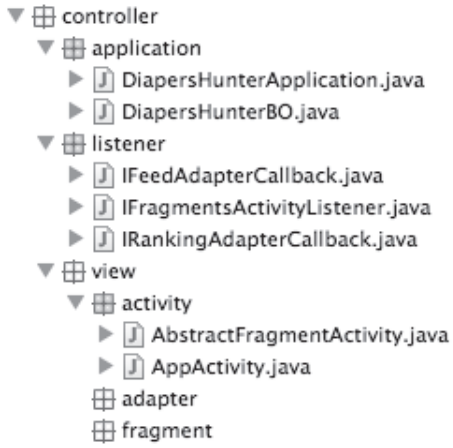


**Figura 5.9.** Organização dos pacotes da camada de controle.

### 5.5.3 View

A camada de visualização é o front-end da aplicação. É com ela que o usuário interage e gerencia as informações geradas pela aplicação. Sendo assim, esta aplicação necessita disponibilizar na camada de view as informações que o usuário gostará de acessar. Desta forma, o aplicativo deve prover as seguintes telas para cumprir os requisitos básicos feitos desde a modelagem do mesmo, na seção 5.1.

– **Tela de Cadastro.** Foi implementada com o intuito de permitir que o usuário se registre na aplicação e passe autilizá-la, como mostrado na figura 5.10.

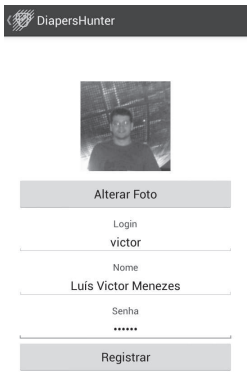


**Figura 5.10.** Exemplo da tela de cadastro do *DiapersHunter*.

– **Tela de Login.** Foi criada com o intuito de permitir que o usuário realize seu login na aplicação e comece a coletar e gerar informação. Nesta tela, também, deve ser possível logar utilizando uma conta do facebook. A figura 5.11 apresenta a tela implementada no *DiapersHunter*.

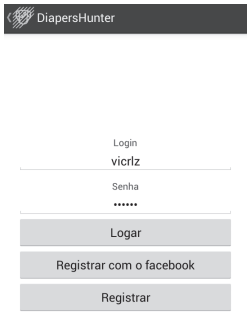


**Figura 5.11.** Exemplo da tela de login do *DiapersHunter*.

– **Tela de Adição/Remoção de Amigos.** Foi implementada com o intuito de permitir adição/remoção de amigos a partir de usuários ativos e/ou do facebook. A figura 5.12 apresenta a tela em questão.

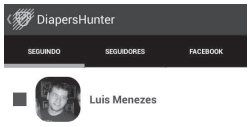


**Figura 5.12.** Exemplo da tela de adição/remoção de amigos do *DiapersHunter*.

– **Tela de Visualização de Avaliação dos Amigos.** Deve ser criada para permitir a visualização das últimas atualizações e avaliações de seus amigos (figura 5.13).

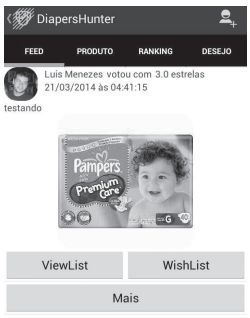


**Figura 5.13.** Exemplo da tela de visualização das avaliações destacadas pelos amigos no *DiapersHunter*.

– **Tela de Produtos.** Deve ser criada para visualizar os produtos presentes na base, bem como suas avaliações e preços. Nesta tela, deve ser indispensável por buscar individualmente por produtos por nome, marca e código de barras, ou, simplesmente, por uma imagem qualquer (figura 5.14).

– **Tela de Criação de Produtos.** Deve ser criada e, em certos casos, incentivado o uso da mesma em função do aumento da base de produtos na base de dados.

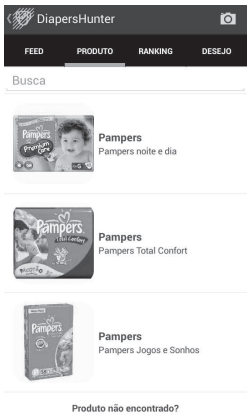


**Figura 5.14.** Exemplo da tela de listagem de produtos presentes no *DiapersHunter*.

Normalmente, esta ação deve ser incentivada com pontuação para o usuário, permitindo que este possa subir no ranking, aumentando sua confiabilidade. Esta subida no ranking permite que o usuário esteja possibilitado e, muitas vezes, confiável para passar a utilizar certas funções do aplicativo que antes não lhe eram permitidas (figura 5.15).


**Figura 5.15.** Exemplo da tela de cadastro de produto no *DiapersHunter*.

– **Tela de Avaliação de Produtos.** Foi implementada para que cada usuário tenha acesso à avaliação recebida por determinado produto e, em alguns casos, fazer suas próprias avaliações e comentários. Nesta tela, ainda, deve ser possível adicionar informação de preço e de local onde este produto foi encontrado.

– **Tela de Visualização de Ranking.** Foi implementada para viabilizar a visualização da lista de usuários baseada no ranking de contribuições. Deve conter o

título dado à posição do ranking que o usuário ocupa e sua pontuação absoluta (figura 5.16).


**Figura 5.16.** Exemplo da tela de ranking de usuários no *DiapersHunter*.

– **Tela de Lista de Desejos.** Foi criada para proporcionar formas de que outros usuários visualizem o desejo de um usuário por um produto específico e, através disso, tenha acesso aos gostos de determinado usuário, seja para dar o produto de presente ou para fazer uma análise mais profunda com base no conhecimento sobre o usuário visualizado.

– **Tela de Notificação de Recompensa.** Foi criada com o intuito de informar ao usuário quantos pontos o mesmo ganhou com uma ação benéfica para o sistema. Na figura 5.17, temos uma notificação na tela de notificações do android, informando a ação efetuada pelo usuário, bem como seu título do ranking e sua pontuação, que neste caso não foi visível pelo motivo de o mesmo já estar no nível máximo.

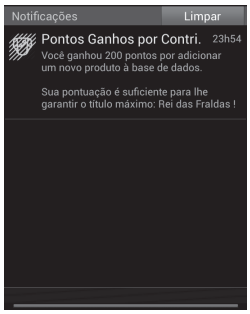


**Figura 5.17.** Exemplo da tela de notificação padrão do Android, exibindo como notificação a recompensa recebida pela ação do usuário.

## 5.6 Hierarquia final do projeto

Nesta seção, apresentamos a hierarquia final do projeto de implementação do *DiapersHunter* a partir da instanciação do *Product Hunter Framework*. As figuras 5.18 e 5.19 demonstram como ficou a hierarquia final do projeto. Com a utilização do MVC, foi fácil as implementações em diferentes pacotes, o que permitiu focar em arquivos com certas semelhanças, o que proporciona um ganho de produtividade, ao nosso ver.

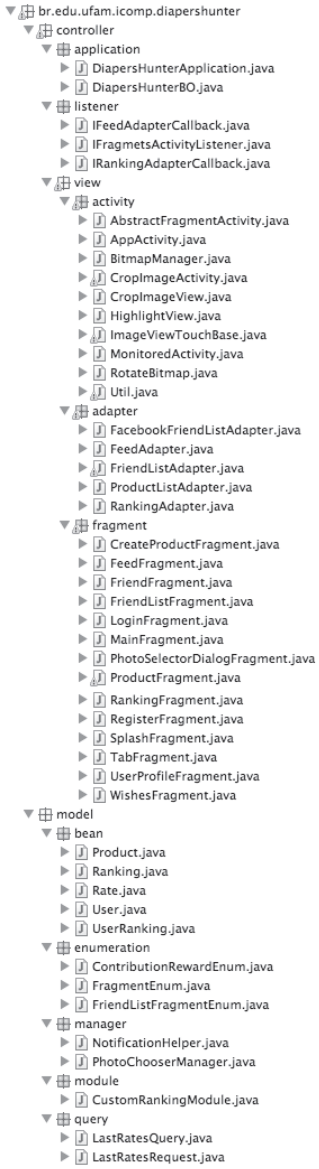


**Figura 5.18.** Demonstração da hierarquia de classes de Model e Controller do projeto de implementação do *DiapersHunter*.

Na figura 5.18, é possível visualizar a separação das classes entre Model e Controller, a camada de View está representada na figura 5.19 e deve estar atrelada a

pelo menos um controller. A camada de *Model* é separada em *Bean*, responsável por armazenar as classes que estendem as AbstractClasses, *Enumeration*, responsável por armazenar as enumerações que serão utilizadas na apliação, Manager, responsável pelas classes que estendem e alteram o papel principal dos *Managers*, Module, responsável pelas classes que estendem e alteram o papel dos *Modules* e *Query*, responsável por controlar os resultados de buscas diferentes das originais no servidor. A camada de *Controller* é separada pelas classes que controlam a aplicação de uma forma geral: *Application* refere-se aos controladores da aplicação e armazenam informações que são utilizadas ao longo da utilização do *DiapersHunter*, assim como o BO, que pode alterar informações presentes nos objetos da camada de Model, *Listener*, que são *interfaces* responsáveis por esperar por determinadas ações para dispararem ações em outras partes da aplicação, elas começam por I por serem *interfaces* e por convenção de decisão de projeto, para separá-las das classes comuns e evitar confusões desnecessárias, *View*, que são responsáveis por controlar os elementos de visualização, exibidos na figura 5.19. Os arquivos .xml, responsáveis pela construção da interface visual (*View*), são separados apenas formalmente com relação ao elemento de view ao qual eles se referem.

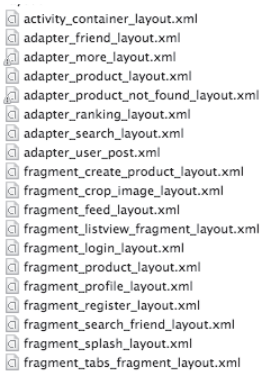


**Figura 5.19.** Demonstração da hierarquia de View utilizada para a construção do *DiapersHunter*.

## 5.7 Conclusões do Estudo de Caso

O objetivo do DiapersHunter era o de provar a viabilidade da construção de uma aplicação à partir do *framework* descrito e implementado neste projeto. Como o *Product Hunter* é destinado à criação de aplicações com foco na busca e avaliação de produtos, suas funcionalidades foram perfeitamente ajustáveis à aplicação *DiapersHunter*, que precisava de métodos para fazer integração com diversas funcionalidades que necessi-

tariam de implementação à parte, caso o desenvolvedor não utilizasse nenhuma forma de ajuda.

Os módulos de usuário foram os que mais se assemelharam à constituição dos mesmos na aplicação, pois os usuários são a constituição básica de uma comunidade colaborativa. Todos os outros módulos são secundários. Desta forma, tudo o que foi necessário é adicionar novas variáveis para representar um usuário do sistema. O manager de controle de sessão foi utilizado também para trabalhar com o usuário desde o período em que o mesmo logou-se ao sistema até o fim de seu uso. Desta forma, não precisamos fazer requisições adicionais para saber qual usuário está utilizando o sistema, apenas o token da sessão era passado para o servidor, que se encarregava de saber qual usuário estava utilizando o sistema e respondia de forma adequada à sua requisição.

O módulo de produtos sofreu poucas alterações, também. Basicamente, o que necessitou ser adicionado foram novos atributos referentes ao tipo de produto utilizado pela aplicação. No entanto, todos os métodos de busca de produtos foram utilizados a partir do *framework* base. A busca por um termo específico e pela imagem funcionaram perfeitamente. O tratamento relacionado à fonte da imagem e o que fora escolhido é totalmente feito pelo *framework* e repassado à aplicação de forma transparente. Tudo o que o desenvolvedor recebe desta ação é um bitmap responsável por exibir a imagem na tela.

Desta forma, conclui-se que a utilização de um caso de uso para provar a possibilidade de desenvolver-se uma aplicação a *framework* a partir de um arcabouço pré-implementado, facilitador de desenvolvimento, foi especialmente importante para a criação da aplicação. Provamos os benefícios decorrentes desta abordagem no tempo de desenvolvimento, que foi bem menor em se tratando de uma aplicação desta natureza. Além disso, a ocorrência de bugs também se torna muito menor, a medida que o *framework* se torna uma estrutura estável e concentra grande parte das funcionalidades básicas desta aplicação.

# Capítulo 6

# Considerações Finais

## 6.1 Conclusões

Com base no que foi apresentado neste projeto, analisamos como conclusões o fato de o *framework* trazer benefícios para desenvolvedores de aplicativos desta natureza e, como o mercado para este tipo de aplicação está em crescimento, esta se torna uma ferramenta importante para o futuro.

A criação de uma ferramenta para gerenciar a parte mais interna da aplicação se mostrou importante por gerenciar todo o back-end, deixando a cargo do desenvolvedor implementar apenas o front-end e alguns poucos extras referentes à funcionalidades que fogem às esperadas pelo *framework*.

Com o desenvolvimento do estudo de caso foi possível perceber que ele determinou um aprimoramento no código, ao possibilitar reusabilidade e confiança no que está implementado. Mesmo nos casos em que o *framework* se mostrou instável, foi possível, ao desenvolvedor, fazer alterações que o tornassem algo mais correto. Desta forma, pode-se dizer que a importância de uma ferramenta como esta auxilia àqueles que necessitam escrever aplicações inseridas no contexto de busca e avaliação de produtos por parte de um grupo de usuários empenhados em fazê-la.

Sendo assim, pode-se inferir que os resultados obtidos com a criação deste projeto obtiveram resultados importantes para a área de computação móvel. Com o desenvolvimento deste trabalho foi possível criar um *framework* de desenvolvimento de aplicações destinada à criação de aplicativos, utilizando Android (ou utilizando sua modelagem para outras plataformas), para realizar a coleta e o consumo de informações importantes sobre determinados tipos de produtos.

Os gerenciadores de comunicação, tratadores de erro, geradores e consumidores de conteúdo são necessidades óbvias destas aplicações. Tudo o que o *framework* faz

é generalizá-los o bastante, mas sem perder o foco naquilo que realmente importa: a facilidade de uso e a confiabilidade na implementação. Isso dá o direito de concluir que este trabalho tem o seu valor na área e pode impulsionar a coleta de dados de produtos para futuras avaliações e utilização plena da sociedade.

## 6.2 Limitações do Trabalho

Apesar de ter o seu valor na computação móvel, o *framework* 'amarra' alguns de suas funcionalidades às necessidades deste tipo de aplicação. O que pretendemos fazer é criar uma ferramenta capaz de ser genérica o bastante para ser utilizada em outras áreas e em outros tipos de produtos. Atualmente não se vê muitas ferramentas deste tipo disponíveis para os desenvolvedores, praticamente cada desenvolvedor desenvolve sua arquitetura e começa, a partir deste ponto, a desenvolver suas aplicações tomando como base sua arquitetura montada previamente. Outro fator importante que não foi adicionado à ferramenta é a utilização de métricas capazes de fazer estatísticas do uso do usuário. Estas ferramentas são importantes para se ter uma mensuração do real uso e necessidade do usuário. Por fim, uma limitação que pode facilmente ser resolvida diz respeito aos managers e modules de redes sociais e localização, que até esta versão implementam apenas integração com Facebook e Foursquare.

## 6.3 Trabalhos Futuros

Como base para trabalhos futuros, buscaremos otimizar o que já foi implementado e desenvolver meios de expandir a gama de aplicações que podem ser produzidas através do *Product Hunter*, que talvez receba outro nome, dada a sua nova finalidade. Desta forma, acabaremos com algumas das limitações citadas na seção anterior, como por exemplo, a necessidade de se desenvolver aplicações relacionadas à produtos e a avaliação dos mesmos. Além disso, novas tendências vão surgir até lá e, por este motivo, deveremos implementar suas novas funcionalidades para que a ferramenta não fique desatualizada e caia em desuso. Como trabalho futuro, temos também a necessidade de reduzir nossa limitação referente à integração com redes sociais e serviços de localização.

# Referências Bibliográficas


Abelson, H.; Chang, M.; Friedman, M.; Lomas, C. & Wolber, D. (2010). Workshop – google app inventor for android: Creating mobile applications as a first computing experience. *2013 IEEE Frontiers in Education Conference (FIE)*, 0:W1C–1–W1C–2.

Belqasmi, F.; Singh, J.; Melhem, S. Y. B. & Glitho, R. H. (2012). Soap-based vs. restful web services: A case study for multimedia conferencing. *IEEE Internet Computing*, 16(4):54–63.

B'far, R. (2004). *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. Cambridge University Press, New York, NY, USA.

Burke, J.; Estrin, D.; Hansen, M.; Parker, A.; Ramanathan, N.; Reddy, S. & Srivastava, M. B. (2006). Participatory sensing. Em *Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Networks and Applications*, pp. 117--134.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P. & Stal, M. (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK.

Christin, D.; Guillemet, J.; Reinhardt, A.; Hollick, M. & Kanhere, S. S. (2011). Privacy-preserving collaborative path hiding for participatory sensing applications. Em *MASS*, pp. 341–350. IEEE.

Cornelius, C.; Kapadia, A.; Kotz, D.; Peebles, D.; Shin, M. & Triandopoulos, N. (2008). Anonysense: Privacy-aware people-centric sensing. Em *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pp. 211--224, New York, NY, USA. ACM.

Cristofaro, E. D. & Soriente, C. (2013). Participatory privacy: Enabling privacy in participatory sensing. *IEEE Network*, 27(1):32–36.

Deng, L. & Cox, L. P. (2009). Livecompare: grocery bargain hunting through participatory sensing. Em *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, HotMobile '09, pp. 4:1--4:6, New York, NY, USA. ACM.

Duan, R.; Bi, M. & Gniady, C. (2011). Exploring memory energy optimizations in smartphones. Em *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pp. 1--8, Washington, DC, USA. IEEE Computer Society.

Eagle, N. (2011). Mobile phones as sensors for social research. Em Hesse-Biber, S. N., editor, *The handbook of emergent technologies in social research*, p. 492–521. Oxford University Press, Oxford.

eMarketer (2013a). Badoo becomes no. 3 social network in brazil. `http://tinyurl.com/newsrsa`. [Online; Acessado em 21 de Abril de 2014].

eMarketer (2013b). Market research on digital media, internet marketing | emarketer. `http://www.emarketer.com/`. [Online; Acessado em 23 de Fevereiro de 2014].

Figueiredo, C. M. S. & Nakamura, E. (2003). Computação móvel: Novas oportunidades e novos desafios. *T&C Amazônia*, Ano 1, No. 2.

Gartner (2013). Technology research | gartner inc. `http://www.gartner.com`. [Online; Acessado em 23 de Março de 2014].

Haghirian, P. & Inoue, A. (2007). An advanced model of consumer attitudes toward advertising on the mobile internet. *Int. J. Mob. Commun.*, 5(1):48--67.

Haghirian, P.; Madlberger, M. & Tanuskova, A. (2005). Increasing advertising value of mobile marketing - an empirical study of antecedents. *2014 47th Hawaii International Conference on System Sciences*, 1:32c.

Holzbauer, B. O.; Szymanski, B. K. & Bulut, E. (2012). Socially-aware market mechanism for participatory sensing. Em *Proceedings of the first ACM international workshop on Mission-oriented wireless sensor networking*, MiSeNet '12, pp. 9--14, New York, NY, USA. ACM.

Jaimes, L. G.; Vergara-Laurens, I. & Labrador, M. A. (2012). A location-based incentive mechanism for participatory sensing systems with budget constraints. Em *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pp. 103--108.

Jayaraman, P. P.; Perera, C.; Georgakopoulos, D. & Zaslavsky, A. B. (2013). Efficient opportunistic sensing using mobile collaborative platform mosden. *CoRR*, abs/1310.4052.

Johnson, R. E. & Foote, B. (1988). Designing reuseable classes. *Journal of Object-Oriented Programming*.

Krontiris, I. & Albers, A. (2012). Monetary incentives in participatory sensing using multi-attributive auctions. *IJPEDS*, 27(4):317–336.

Lane, N. D.; Miluzzo, E.; Lu, H.; Peebles, D.; Choudhury, T. & Campbell, A. T. (2010). A survey of mobile phone sensing. *Comm. Mag.*, 48(9):140--150.

Lee, J.-S. & Hoh, B. (2010). Sell your experiences: a market mechanism based incentive for participatory sensing. Em *2010 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 60--68. IEEE.

Lee, J.-S. & Szymanski, B. K. (2009). A participation incentive market mechanism for allocating heterogeneous network services. Em *Proceedings of the 28th IEEE conference on Global telecommunications*, GLOBECOM'09, pp. 2206--2211, Piscataway, NJ, USA. IEEE Press.

Lee, Y. E. & Benbasat, I. (2003). Interface design for mobile commerce. *Commun. ACM*, 46(12):48--52.

Lilly, P. (2013). Mobile devices to outnumber global population by 2017. `http://tinyurl.com/pbodtus`. [Online; Acessado em 13 de Abril de 2014].

Liu, X.; Lu, M.; Ooi, B. C.; Shen, Y.; Wu, S. & Zhang, M. (2012). Cdas: A crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040--1051.

Mahmoud, Q. H. & Popowicz, P. (2010). Toward a framework for the discovery and acquisition of mobile applications. *Mobile Business / Global Mobility Roundtable, International Conference on*, 0:58–65.

Mattsson, M. (1996). *Object-Oriented Frameworks - A survey of methodological issues*. Tese de doutorado, Department of Computer Science and Business Administration; University College of Karlskrona/Ronneby; Department of Computer Science; Lund University, S- 372 25 Ronneby, Sweden; Box 118; S- 221 00 Lund; SWEDEN.

Moretti, J. (2013). Um overview do mercado mobile no brasil. `http://tinyurl.com/pblxfpy`. [Online; Acessado em 17 de Fevereiro de 2014].

Navón, J. & Fernandez, F. (2011). The essence of rest architectural style. Em Wilde, E. & Pautasso, C., editores, *REST: From Research to Practice*, pp. 21–33. Springer.

Pree, W. (1995). *Design Patterns for Object-oriented Software Development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

sbicheno (2013). Android captures record 81 percent share of global smartphone shipments in q3 2013. `http://tinyurl.com/obgdyy2`. [Online; Acessado em 18 de Fevereiro de 2014].

Sherchan, W.; Jayaraman, P. P.; Krishnaswamy, S.; Zaslavsky, A.; Loke, S. & Sinha, A. (2012). Using on-the-move mining for mobile crowdsensing. *2013 IEEE 14th International Conference on Mobile Data Management*, 0:115–124.

Taylor, D. (2009). Work the shell: Messing around with imagemagick. *Linux J.*, 2009(185).

Taylor, D. (2013). Work the shell: Image manipulation with imagemagick. *Linux J.*, 2013(234).

Unhelkar, B. & Murugesan, S. (2010). The enterprise mobile applications development framework. *IT Professional*, 12(3):33–39.

Yang, S.-Z. (2011). The marketing chain in the mobile internet era. Em *ICMLC*, pp. 1058–1061. IEEE.