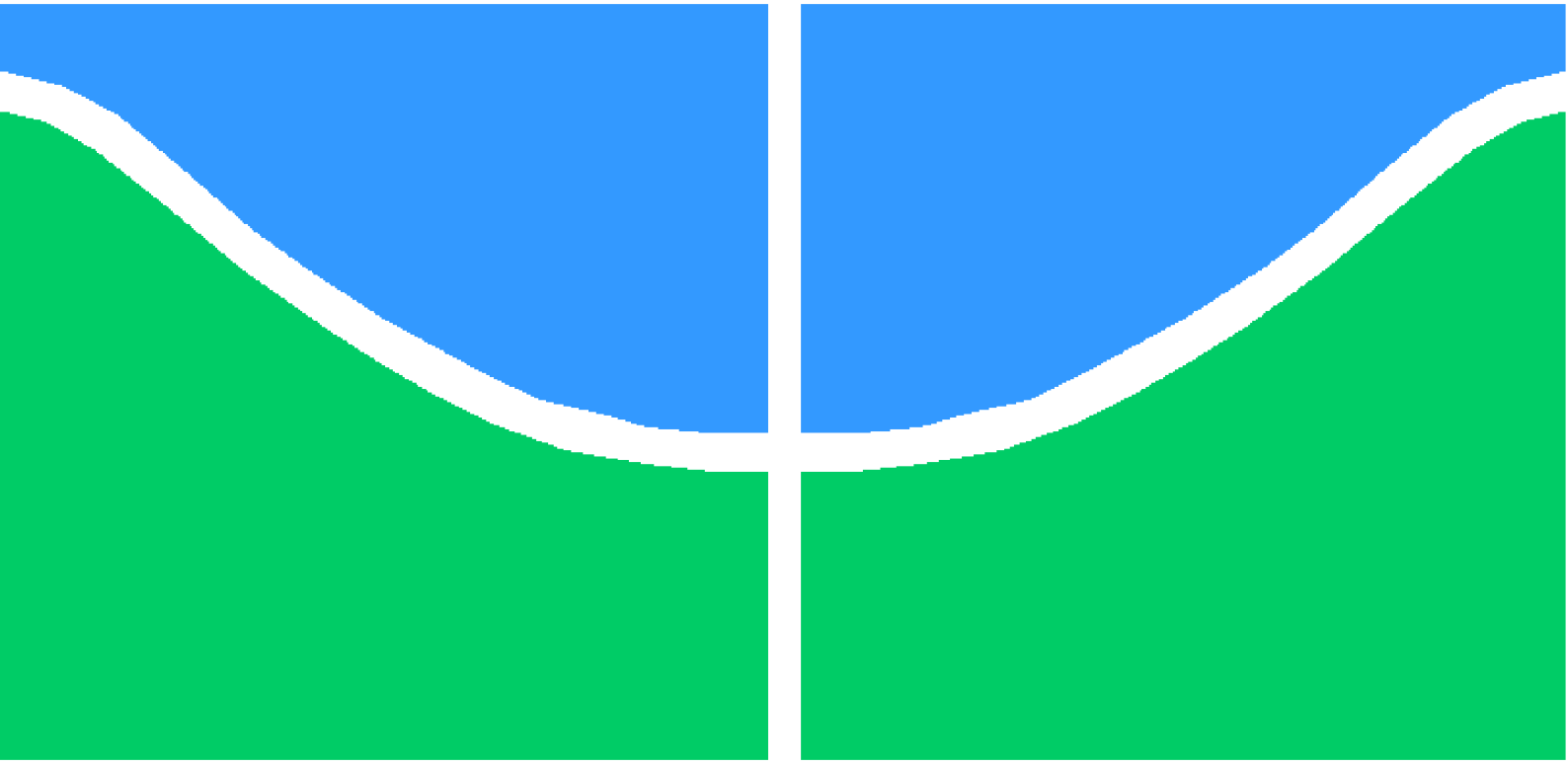

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Guimifiu: Uma aplicação para controle social sobre postos de combustíveis

Autores: João Paulo Siqueira Ribeiro
Marco William Paulo da Silva

Orientador: Prof. Msc Giovanni Almeida Santos

Brasília, DF

2017

João Paulo Siqueira Ribeiro
Marco William Paulo da Silva

Guimifiu: Uma aplicação para controle social sobre postos de combustíveis

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Msc Giovanni Almeida Santos

Brasília, DF

2017

João Paulo Siqueira Ribeiro
Marco William Paulo da Silva
Guimifiu: Uma aplicação para controle social sobre postos de combustíveis/
João Paulo Siqueira Ribeiro
Marco William Paulo da Silva. – Brasília, DF, 2017-
71 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc Giovanni Almeida Santos

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2017.

1. Postos de Combustíveis. 2. Engenharia de Software. I. Prof. Msc Giovanni Almeida Santos. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Guimifiu: Uma aplicação para controle social sobre postos de combustíveis

CDU 02:141:005.6

João Paulo Siqueira Ribeiro
Marco William Paulo da Silva

Guimifiu: Uma aplicação para controle social sobre postos de combustíveis

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 29 de novembro de 2017 – Data da aprovação do trabalho:

Prof. Msc Giovanni Almeida Santos
Orientador

Professora Dra. Edna Dias Canedo
Convidado 1

Professor Dr. Vandor Roberto Vilardi Rissoli
Convidado 2

Brasília, DF
2017

Agradecimentos

Eu, João Paulo Ribeiro, agradeço aos meus pais e às minhas irmãs pela educação e apoio ao longo da minha vida. Agradeço a todos os meus amigos que fizeram parte da minha jornada, desde os que conheci no Ensino Médio: Daniel Ribeiro, Flávio Ghiraldelli, Franck Teixeira, Matheus Sarmento e Vitor Hakme; aos que conheci na faculdade: Caio Nardelli, Matheus Godinho, Simião Carvalho e Marco William. Agradeço também aos professores que, além de me ensinar, acreditaram em mim e me deram as oportunidades que precisei para crescer, em especial os professores George Marsicano, Sérgio Freitas, Paulo Meirelles e Giovanni Almeida. Por fim, agradeço aos meus chefes no Tribunal de Contas da União Ricardo Stumpf e Vitor Hauck que acreditam em mim e me dão muito apoio para o meu trabalho no tribunal.

Eu, Marco William Paulo da Silva, agradeço ao meu pai Vicente, minha mãe Elaene, aos meus irmãos, Maxwell e Maykon, e aos meus tios Batalha e Elena por sempre me apoiarem e ajudarem em todas as minhas conquistas. Agradeço também à todos os professores que contribuiram na minha formação, dentre eles o professor Ricardo Fragelli que me motivou bastante no primeiro semestre com ótimas aulas de Cálculo 1, as professoras Rejane Maria e Milene Serrano que desde o começo do curso me apoiaram com oportunidades de pesquisas e intercâmbio, e o professor Giovanni Almeida que é o orientador deste trabalho. Agradeço também a todos meus amigos e minha namorada Raquel, por sempre me darem suporte quando necessário.

"Se a ideia de viver em um mundo onde tentar respeitar os direitos básicos daqueles que estão ao seu redor e valorizar cada um simplesmente porque existimos é tão assustadora, uma tarefa impossível, então que tipo de mundo nos é deixado? E em que tipo de mundo você quer viver?" (Diana Prince)

Resumo

Dentre os exportadores de petróleo, o Brasil é um dos países cujo preço da gasolina é mais elevado. Em 2015, foram revelados esquemas de cartel dos postos de combustíveis, mas, mesmo após a descoberta e tentativa de boicote pela população, os preços se mantiveram altos. O Guimifiu é um aplicativo multiplataforma implementado com o objetivo de incentivar a população brasileira a protestar contra os preços abusivos de combustível no país. Programado em *Ionic 2* com uma API em *Ruby on Rails*, o aplicativo está sendo desenvolvido utilizando métodos e conceitos da Engenharia de Software, como *Scrum* e entrega contínua. As atualizações do Ionic 2 impediram que o trabalho alcançasse os resultados desejados. Apesar disso, a versão final do aplicativo está funcional e pronta para entrar em produção além de ser evoluída, principalmente pelo fato da API e o módulo de administração estarem bem desenvolvidos.

Palavras-chaves: Aplicativo. Multiplataforma. Boicote. Postos de combustíveis. Engenharia de Software.

Abstract

Among the oil-exporter countries, Brazil has one of the world's most expensive gas prices. In 2015, the existance of gas station cartels was uncovered, but even after the discovery and attempted boycott by the population, prices remained high. Guimifiu is a cross-platform application implemented with the aim of encouraging the Brazilian population to protest against abusive fuel prices in the country. Programmed in Ionic 2 with an API in Ruby on Rails, the application is being developed using methods and concepts of software engineering, such as Scrum and continuous delivery. Ionic 2's updates have prevented the work from achieving desired results. Nonetheless, the final version of the application is functional and ready to go into production and be evolved, mainly because the API and the administration module are well developed.

Keywords: Application. Cross-platform. Boycott. Gas stations. Software Engineering.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GPS	Global Positioning System
HTML5	Hypertext Markup Language 5
CSS3	Cascading Style Sheets 3
API	Application Programming Interface
JSON	JavaScript Object Notation
MVC	Model View Controller
DTA	Direct Test Acess
TDD	Test Driven Development
RDS-TMC	Radio Data System-Traffic Message Channel
ILS	Integrated Library System
QG	Qualidade Geral
QE	Quantidade de Estrelas
PG	Preço da Gasolina
PA	Preço do Álcool
PD	Preço do Diesel
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos

Sumário

1 Introdução

1.1 Contexto

Em 2013, Brasil era o 39º país com a gasolina mais cara no mundo (GLOBO, 2013). Ao analisar o *ranking* dos preços de gasolina, é possível observar que os países que exportam gasolina possuem os menores preços, porém isso não acontece no Brasil (PRICES, 2017). Em 2015, a Polícia Federal e o Ministério Público do DF descobriram um esquema de cartel que combinava os valores dos combustíveis nos postos; mesmo após essa descoberta, os preços não diminuíram (BRAZILIENSE, 2015).

Para mudar esse quadro, observa-se uma necessidade de um movimento comunitário da população brasileira para desestimular a existência dos cartéis. As manifestações de 2013 foram um grande exemplo da nação unida por uma mudança fazendo efeito, uma vez que as tarifas de ônibus diminuíram devido a movimentação popular. A falta de mecanismos de divulgação de suspeitas de cartéis (PAULO, 2013) e adulteração de combustíveis, entre outras situações fazem com que estas não tenham repercussão negativa para os donos dos postos. Foram tentados boicotes aos postos na época em que se descobriu o esquema de cartel, mas, pela falta de comunicação eficaz e organização, o resultado almejado não foi obtido (BRAZILIENSE, 2016).

Diante dessa realidade, foi concebido o Guimifiu, um aplicativo de celular que incentiva a população a boicotar postos de combustíveis cujos preços sejam muito altos. O aplicativo gerará uma lista de postos e os postos boicotados, onde o usuário será incentivado a não abastecer o veículo. Além disso, outros postos com preços menores e melhor qualidade serão recomendados, como forma de incentivar a diminuição dos preços. As informações de preços sobre os postos serão mantidas por meio de informações providas pelos usuários de forma colaborativa. O usuário também receberá uma recomendação de postos próximos a uma rota traçada por ele, pois não necessariamente o posto com melhores preços e melhor qualidade é o mais conveniente. O aplicativo foi feito utilizando conceitos e métodos aprendidos na engenharia de software, como o *Scrum*, integração contínua e gerência de qualidade.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é aplicar conceitos da Engenharia de Software no desenvolvimento de um aplicativo multiplataforma que auxilie a população a avaliar os postos

de combustíveis e encontrar os de melhor qualidade nos quesitos preço, atendimento e qualidade do combustível.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Disponibilizar um mapa com os postos de combustíveis e suas informações;
- Possibilitar a avaliação dos postos de combustível pelo usuário;
- Atualizar, com auxílio de informações citadas pelos usuários, os preços praticados pelos postos de combustíveis;
- Criar uma lista de postos não recomendados, para incentivar a diminuição de preços de combustíveis;
- Recomendar postos de combustíveis dentro de uma rota definida pelo usuário.

1.3 Justificativa

A contribuição do trabalho pode ser observada em duas frentes:

- **Acadêmica**: O trabalho mostra um estudo de caso de um projeto de engenharia de software realizado com Ionic 2 e como foram solucionados os problemas encontrados, auxiliando pesquisas a avaliarem o framework.
- **Social**: O projeto auxilia as pessoas a abastecerem nos melhores postos e a diminuir o preço e qualidade dos combustíveis, desestimulando esquemas de cartéis.

1.4 Metodologia de Pesquisa

Para este trabalho, foi utilizado a pesquisa exploratória para poder determinar as melhores tecnologias, no que tange ao desenvolvimento móvel, e as melhores práticas, no que tange ao desenvolvimento de software em geral.

O levantamento bibliográfico foi realizado utilizando a base de artigos da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), que faz busca em outras bases como a da IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos), bem como os artigos encontrados no Google Academics.

1.5 Organização do Trabalho

Este trabalho está dividido em seis capítulos. O Capítulo 2 apresenta o referencial teórico com os conceitos abordados, tais como desenvolvimento móvel, softwares de navegação e os principais conceitos da Engenharia de Software utilizados durante o desenvolvimento. No Capítulo 3, detalha-se a a metodologia, descrevendo a arquitetura do sistema, bem como a organização da equipe e as adaptações feitas nas metodologias de trabalho. No Capítulo 4 são relatados os resultados do que foi desenvolvido. O Capítulo 5 apresenta o cronograma de atividades do projeto e, por fim, o Capítulo 6, encontram-se as considerações sobre os estudos realizados e o futuro deste trabalho.

2 Referencial Teórico

2.1 Software de Navegação

O sistema de navegação GPS (*Global Positioning System*) faz cada vez mais parte da vida das pessoas (HOFMANN-WELLENHOF; LICHTENEGGER; COLLINS, 1993). Aplicativos como Google Maps, Waze e HERE WeGo fazem uso desse sistema para auxiliar o usuário na decisão de rotas em seu dia-a-dia. Para enviar a informação *online* e em tempo real das ruas, principalmente na hora de criar rotas, o RDS-TMC (*Radio Data System-Traffic Message Channel*) é comumente utilizado (KOPITZ; MARKS, 1998).

Um exemplo de software de navegação é o Waze, um dos maiores aplicativos de navegação do mundo desenvolvido pela *Waze Mobile* de Israel e posteriormente adquirido pelo Google. Sua base de dados é sustentada pelos usuários que, além de indicar quais ruas estão engarrafadas, mostra também onde encontram-se *blitz* da polícia, radares e acidentes pela cidade. Além disso, é possível ver a localização de postos de combustível (WAZE, 2017).

Outro exemplo de software de navegação é o Google Maps, desenvolvido pelo Google; esta aplicação traça rotas em tempo real de acordo com informações de trânsito. Além disso, ele também permite ver as ruas com o *Street View*. O Google Maps mostra formas de deslocamento, via carro, metrô, ônibus e até mesmo a pé. Além disso, ele indica a localização de postos de combustíveis perto das rotas selecionadas (GOOGLE, 2017b).

2.2 Desenvolvimento Móvel

Atualmente, os dois sistemas operacionais para desenvolvimento móvel mais utilizados são Android e iOS (GARTNER, 2016). O Android possui um *kernel* do Linux modificado utilizando várias bibliotecas em C/C++ (SÁNCHEZ et al., 2014) e o principal *framework* de desenvolvimento oferecido pela Google usa o Java como linguagem de programação (GOOGLE, 2017a). Baseado no Mac OS X, o iOS é a plataforma da Apple para desenvolvimento móvel e usa como linguagem primária o Objective-C (SÁNCHEZ et al., 2014).

Para o desenvolvimento de aplicações móveis multiplataformas, existem duas formas de implementação: a nativa, utilizando ferramentas que usam funcionalidades específicas do sistema operacional, e a híbrida, usando *frameworks* que fazem o uso de HTML5, CSS3 e JavaScript (LIM, 2015).

Aplicações nativas são específicas do sistema operacional e podem ser desenvol-

vidas para múltiplas plataformas com um *framework* que compile um único código para todos os dispositivos (SERRANO; HERNANTES; GALLARDO, 2013). A performance de aplicativos nativos é melhor pelo uso de código nativo (DALMASSO et al., 2013). Existem várias ferramentas para desenvolvimento multiplataforma nativo como o *Xamarin*, *Appecelator Titanium*, *Corono* e *Unity*, entre outros. Exemplos de aplicativos nativos são *Angry Birds* e *Shazam*.

Aplicativos híbridos são desenvolvidos utilizando ferramentas de desenvolvimento *web* em junção com elementos nativos (SERRANO; HERNANTES; GALLARDO, 2013). Eles modificam páginas HTML pré-empacotadas, mudando a interface de acordo com a plataforma (DALMASSO et al., 2013). A interface de usuário de um aplicativo híbrido possui em tempo menor de desenvolvimento de composições de natureza dinâmica (LIM, 2015). Existem vários *frameworks* para desenvolvimento híbrido, tais como o Ionic, QT e Adobe Air, dentre outros. Exemplos de aplicativos híbridos são Netflix e Linkedin.

2.3 Scrum

O Scrum é um *framework* de desenvolvimento ágil que consiste em um time, suas atividades associadas, artefatos e regras que servem para entregar uma série de incrementos de produto dentro de um determinado período (geralmente de duas a quatro semanas) chamado *sprint* (RUBIN, 2012). O time é formado por:

- ***Product Owner***, responsável por manter os requisitos e dividi-los para o resto da equipe (HAYATA; TOMOHIRO; HAN, 2011);
- ***Scrum Master***, responsável por liderar e ajudar a equipe de desenvolvimento, além de ser a ponte entre o *product owner* e a equipe para conduzir as reuniões mantendo a sincronia do projeto (BASS, 2014);
- **Equipe de desenvolvimento**, profissionais responsáveis por desenvolver o produto e entregá-lo ao fim de cada *sprint* (GANNON, 2013).

O Scrum começa com a visão do product owner sobre o produto e a criação das funcionalidades do sistema em uma lista priorizada chamada product backlog. Depois, são separadas tarefas do backlog para compor a sprint (de acordo com o que a equipe acredita conseguir desenvolver durante o ciclo) e assim a sprint se inicia. No final, é feita uma revisão da sprint, onde os itens desenvolvidos pela equipe são revisados por todos as partes envolvidas e; uma retrospectiva, apontando pontos positivos e negativos da sprint é feita, com o objetivo de melhorar a próxima (RUBIN, 2012). A Figura 1 mostra o fluxo das atividades do Scrum.

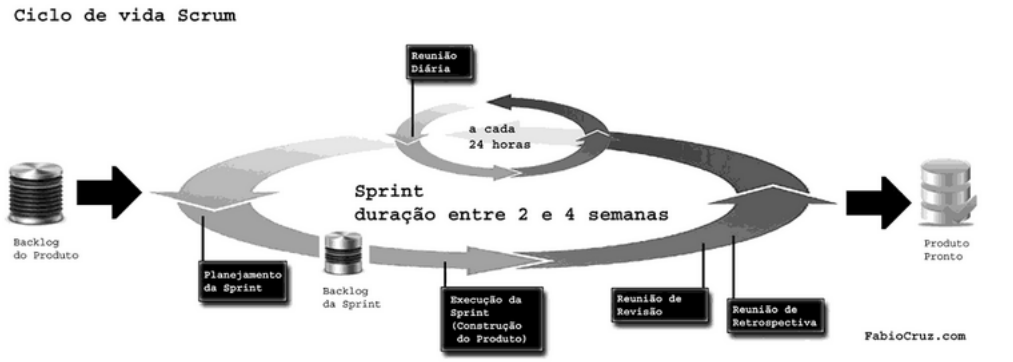

Figura 1: Fluxograma do Scrum. Fonte: (CRUZ, 2013)

2.4 Métricas de Software

Monitorar a qualidade de software é fundamental para qualquer projeto (MEIRELLES, 2013). De acordo com a ABNT (2003), métrica é uma união de procedimentos com o objetivo de definir escala e métodos para medidas. Segundo Mills (1988), uma métrica é considerada boa quando ela ajuda a medir os parâmetros de qualidade de um software.

Muitos desenvolvedores utilizam métricas para ter uma noção da completude ou não dos seus requisitos, se o design está bom e se o que foi desenvolvido foi feito com qualidade. Métricas de software também podem ser utilizadas para tomar decisões sobre o futuro de um projeto (FENTON; BIEMAN, 2015).

Atualmente, métricas de orientação a objetos são mais utilizadas do que outros tipos de métricas (RADJENOVIć et al., 2013). Para o Ruby, existem algumas ferramentas como o Flog, que verifica a complexidade do código-fonte. Quanto maior a complexidade, mais difícil é o teste e a manutenção (DAVIS, 2017). Outra ferramenta muito usada para acompanhar a qualidade do código é o Reek, que reporta os *code smells* encontrados no código (RUTHERFORD, 2017b). *Code smells* mostram onde, no seu código Ruby, pode ser encontradas dificuldades de leitura, manutenção e evolução, mas não indicam se o código funciona ou não (RUTHERFORD, 2017a).

2.5 Integração Contínua

Com o advento dos métodos ágeis, a entrega rápida de sofware funcional e de qualidade para o cliente teve sua importância aumentada. Uma das práticas mais importantes para atingir esse objetivo é a integração contínua (BECK; ANDRES, 2004).

Integração contínua é a prática onde o time integra o código periodicamente (de preferência em tempos curtos) para encontrar erros da forma mais rápida possível (FOWLER, 2006)

Uma *build* produzida em uma integração passa pelos seguintes passos (BECK; ANDRES, 2004):

- Compilar um código-fonte;
- Rodar uma suíte de testes;
- Verificar a qualidade do código via análise estática;
- Produzir uma versão instalável de componentes pré-construídos.

No desenvolvimento de uma ILS (*Integrated Library System*), um produto multiplataforma produzido utilizando uma adaptação descentralizada do Scrum, a integração contínua contribuiu significativamente para a produtividade do trabalho, mostrando a importância da prática em projetos dessa natureza (SUTHERLAND et al., 2007).

2.6 Entrega Contínua

O processo normal de entrega de software consiste em entregar vários artefatos e realizar diversas tarefas para colocar o produto produzido em produção (MANTYLA; VANHANEN, 2011). Geralmente, devido à customização necessária para implantar um software de cliente pra cliente, o processo de entrega é normalmente feito manualmente (HUMBLE; FARLEY, 2010).

A entrega contínua de um software é um reflexo desses problemas, bem como do primeiro princípio do manifesto ágil: *"Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado."*(BECK et al., 2001). A entrega contínua foca em *build*, *deploy*, teste e processo de *release* (HUMBLE; FARLEY, 2010). Existem cinco aspectos importantes sobre a entrega contínua (WORLD, 2013):

- Redução dos custos com entregas;
- Redução do tempo de entrega para o cliente;
- Redução dos riscos;

- Aumento da qualidade do produto;
- Foco na automação.

2.7 Testes Automatizados

Testar software é uma atividade que consome muito tempo, mas muito importante na garantia de qualidade do produto (DWLLO et al., 1987). Os objetivos principais do teste são mostrar para o cliente e o desenvolvedor que o software implementado atende aos requisitos e descobrir falhas ou defeitos no software, apesar de não garantir a não existência delas (SOMMERVILLE, 1982). Existem várias estratégias para testes; dessas estratégias derivam vários tipos de teste, tais como testes unitário, funcionais, de integração, de stress, entre outros (MYERS, 2004).

Uma unidade de software é o menor módulo possível de um software, geralmente sendo um método ou uma função (RUNESON, 2006). Segundo Luo, Probert e Ural (1995) os testes geralmente só são iniciados após uma grande parte do software já estar implementada. Para estes casos, recomenda-se o método de DTA (*Direct Test Acess*), que consiste em acessar diretamente a unidade de software sem removê-la de seu contexto.

Porém, nem sempre os testes se iniciam após o desenvolvimento do software. O TDD (*Test Driven Development*) é uma técnica que também faz uso de testes unitários, mas inverte a ordem normal, implementando primeiro os testes e depois as funcionalidades necessárias para a aprovação dos testes (WILLIAMS; MAXIMILIEN; VOUK, 2003).

Myers (2004) define teste de integração como a verificação das interfaces das partes do sistema. Esse tipo de teste consegue identificar erros que não seriam encontrados apenas em testes unitários. Essas interfaces de diferentes partes do sistema são, por exemplo, uso de arquivos compartilhados e chamadas de métodos de outros sistemas, entre outras (SPILLNER, 1995).

2.8 Prototipação

Segundo a ISO (1997), usabilidade é uma medida de efetividade, eficiência e satisfação de um usuário ao utilizar um produto. Para ajudar a medir a usabilidade, além de ajudar a elicitar e validar requisitos, a prototipação é um método muito utilizado na engenharia de software (SOMMERVILLE, 1982).

Protótipos variam de acordo com a fidelidade ao software final. A fidelidade pode variar em interatividade, visual e conteúdo. Quanto mais elementos clicáveis, maior a proximidade do visual do protótipo ao visual final e quanto maior a proximidade com o processo da aplicação, maior a fidelidade do protótipo (NIELSEN, 2017). Protótipos

de alta fidelidade, que são mais interativos, fazem com que o designer não precise se preocupar tanto em fazer o protótipo funcionar e observar melhor o usuário testando, para poder tirar melhores conclusões. Em contrapartida, protótipos de baixa fidelidade são mais fáceis de serem alterados e dão aos *stakeholders* uma noção melhor do progresso do trabalho (NIELSEN, 2017).

A técnica conhecida como *Crazy Eights* é um passo do processo de design sprint da empresa Google Ventures. O *Crazy Eights* é uma técnica onde os participantes possuem 5 minutos para desenhar as oito telas principais do aplicativo, sendo assim 40 segundos para cada uma. O objetivo principal é ser bem direto e, como o tempo é curto, apenas os pontos cruciais para o aplicativo serão desenhados (KNAPP, 2012).

3 Metodologia

Neste capítulo, será discutida a metodologia adotada para desenvolver este trabalho. Inicialmente será mostrado como foi utilizado o Scrum no projeto, o *hardware* utilizado pela para desenvolver o projeto, as ferramentas usadas e por fim a arquitetura desenvolvida no projeto.

3.1 Scrum

3.1.1 Papéis

Como a equipe de desenvolvimento é composta apenas por duas pessoas, definiu-se que o papel de *Scrum Master* não era necessário para o projeto, sendo suas atribuições divididas entre os dois membros da equipe. O papel de *Product Owner* esteve a cargo do professor orientador.

3.1.2 *Sprints*

Para o desenvolvimento da aplicação as *sprints* terão duração de duas semanas. As atividades que compõem as *sprints* são:

- **Planejamento de *Sprint***, no qual são separados os itens do *product backlog* que são desenvolvidos durante o período de duas semanas;
- **Revisão da *Sprint***, na qual são mostradas e discutidas junto ao *Product Owner* as histórias de usuário que foram implementadas durante a *sprint*;
- **Retrospectiva**, na qual são discutidos os pontos positivos, negativos e possíveis melhorias para as próximas iterações;
- **Reuniões semanal**, nas quais são discutidas, junto ao professor orientador, os caminhos a serem seguidos no projeto;

A equipe desenvolve utilizando a estratégia do *pair programming*, mas, quando a dupla não pode estar presencialmente reunida, um desenvolvedor implementa a história e o outro faz uma revisão.

3.2 Hardware e Software de Desenvolvimento

Foram utilizados dois computadores diferentes para o desenvolvimento desse projeto, um Linux e outro Mac OS. Além disso, foram utilizados dois modelos de celular, um Android e o outro iOS. As especificações podem ser encontradas na Tabela 1

Tabela 1: Especificação dos equipamentos utilizados.

Tipo	**Sistema Operacional**	**Modelo**
Computador	Linux Debian 8	Dell Inspiron 5558
Computador	Mac OS Sierra 10.12.5	Macbook Pro 15 Retina
Celular	Android 6.0	Motorola Moto G (3ª Geração)
Celular	iOS 10.3.2	Iphone 5s

3.3 Ferramentas Utilizadas

3.3.1 Linguagens e *Frameworks*

O aplicativo Guimifiu está sendo desenvolvido utilizando o *framework* Ionic 2, que por sua vez utiliza TypeScript como linguagem de programação. O TypeScript é uma linguagem que facilita o desenvolvimento em JavaScript em larga escala. É uma linguagem fortemente tipada, que possibilita a aplicação de orientação a objetos em aplicações JavaScript. O TypeScript é compilado e transformado em JavaScript (MICROSOFT, 2017). O Ionic 2 é um *framework* livre criado pela *Drifty Co.* em 2012 que empacota o Cordova e o TypeScript para utilizar funções nativas dos aparelhos móveis e criar aplicativos multiplataforma a partir de um único código (DRIFTYCO, 2017).

Em conjunto com o desenvolvimento do aplicativo, foi desenvolvida uma API RESTful utilizando a linguagem de programação Ruby com o *framework* Ruby on Rails. Criado em 1995, o Ruby é uma linguagem de tipagem dinâmica *open source* criada por uma junção das melhores partes das linguagens Perl, Smalltak, Eiffel, Ada e Lisp. O Ruby permite *multithreading* independentemente do suporte do sistema operacional (RUBY, 2017). O Ruby on Rails é um *framework* criado em 2004 para desenvolvimento de aplicações *web* que faz com que o programador use o paradigma de convenção sobre configuração (HANSSON, 2017). Vários softwares são desenvolvidos em Ruby on Rails, tais como o GitHub, Shopify, Hulu e Twitch. A representação dos dados da API é feita em formato JSON (*JavaScript Object Notation*).

3.3.2 Testes

Para a aplicação *mobile*, foram utilizados os *frameworks open source* Jasmine para escrever os testes e o Karma para rodar a suíte. O Karma é um ambiente de teste para

JavaScript que os desenvolvedores conseguem *feedbacks* rápidos do código que estão desenvolvendo (KARMA, 2017). O Jasmine é um *framework* que utiliza conceitos de *behavior-driven development* para escrever testes (LABS, 2017). O CodeCov foi utilizado para adquirir a cobertura de código dos testes.

No desenvolvimento dos testes da API utilizou-se o RSpec, que é uma ferramenta para escrever e executar a suíte de testes (RSPEC, 2017) e o Coveralls que adquire a cobertura de código dos testes (INDUSTRIES, 2017).

3.3.3 Métricas de Código-fonte

Para as métricas de código-fonte da aplicação *mobile*, utilizou-se o TSLint, que checa códigos TypeScript em questões de manutenabilidade, leitura e erros funcionais (PALANTIR, 2017).

No código-fonte da API utilizou-se o CodeClimate, uma ferramenta que baixa o código do GitHub e checa em seus servidores a complexidade do código, duplicação, estilo e segurança (CODECLIMATE, 2017). O CodeClimate utiliza as seguintes métricas para gerar o relatório final:

- **Reek:** para verificação de *code smells*, que são sintomas no código-fonte que podem gerar problemas;
- **Flog:** para a verificação da complexidade do código Ruby;
- **Rubocop:** que verifica estilo e qualidade do código;
- **Brakeman:** para uma verificação estática de segurança do código.

Além disso, existe também uma ferramenta própria do CodeClimate para checagem de duplicações de código.

O Rubocop foi utilizado também fora do CodeClimate para medir a complexidade do código. Além de complexidade, o Rubocop possui diversas outras ferramentas embutidas que verificam outros aspectos, com base no guia de estilo do Ruby mantido pela comunidade (BATSOV, 2017). Os padrões mínimos ou máximos dessas outras ferramentas são analisados junto automatizar desenvolvimento do projeto.

3.3.4 Integração e Entrega Contínua

O TravisCI é uma ferramenta grátis para projetos *open source* que faz a integração contínua de projetos em várias linguagens. As *builds* são feitas imediatamente após os *pushes* na ferramenta de controle de versão e, por meio de um script criado pelo usuário,

o Travis configura e roda a suíte de testes do *commit*. Além disso, também pode fazer *deploys* no Heroku caso a *build* esteja passando (TRAVISCI, 2017).

O Heroku é uma plataforma que permite criar, entregar, monitorar e escalar aplicativos. Comumente utilizado para o *deploy* de aplicações web, a plataforma também serve para hospedar serviços na nuvem, facilitando a arquitetura de aplicativo-API (HEROKU, 2017).

O Fastlane é uma ferramenta *open source* para automatizar a entrega contínua de aplicativos iOS e Android. Ele conecta diretamente com a App Store e a Play Store para fazer *deploys* das versões do aplicativo, inclusive versões beta exclusivas para testes (FASTLANE, 2017).

3.3.5 Prototipação

Após a finalização do *Crazy Eights*, foram escolhidas as melhores telas para serem levadas como base da construção do aplicativo. Após a validação dos requisitos, foram desenhados protótipos de baixa fidelidade para as histórias de usuários que seriam desenvolvidas. Sendo assim, a prototipação fez o uso de duas ferramentas:

- **Marvelapp:** utilizado para simular o fluxo das telas geradas pelo *Crazy Eights*, onde é possível tirar fotos dos protótipos desenhados e criar transições nas telas desenhadas de forma a mostrar o fluxo da aplicação, que pode ser tanto web quando para Android e iOS (MARVEL, 2017);
- **NinjaMock:** utilizado para a prototipação de baixa fidelidade das histórias de usuário. É uma ferramenta de criação de *wireframe designs* para plataformas móveis e web (NINJAMOCK, 2017).

A Figura 2 mostra o protótipo de papel da tela de login da aplicação. O login pode ser feito via Facebook, Google ou uma conta local.

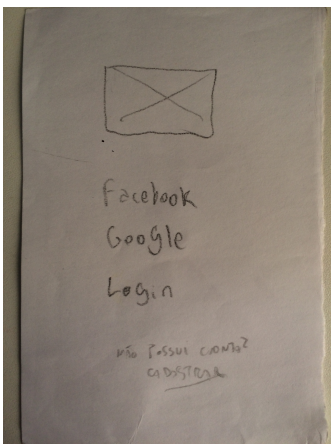

Figura 2: Protótipo de papel da tela de login. Fonte: autores

A Figura 3 mostra o protótipo de papel da tela inicial da aplicação. Um mapa será mostrado com algumas informações do usuário.

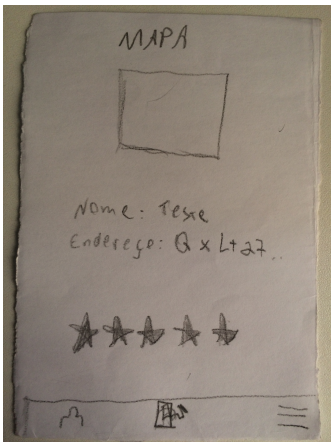

Figura 3: Protótipo de papel da tela inicial. Fonte: autores

A Figura 4 mostra o protótipo de papel do menu lateral da aplicação.

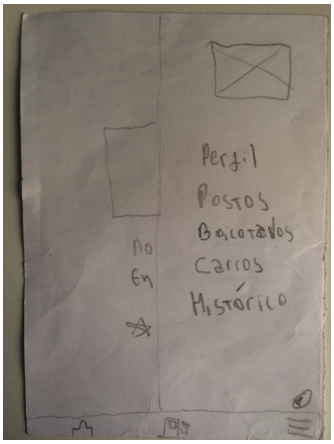

Figura 4: Protótipo de papel do menu lateral. Fonte: autores

Os demais protótipos se encontram no Apêndice D.

3.3.6 Comunicação

A comunicação da equipe foi em sua maioria pela ferramenta Slack. O Slack é uma ferramenta de comunicação através de canais de mensagens. O objetivo é que cada canal tenha o seu respectivo assunto para organizar o projeto, que pode ser público, ou seja, para todos os membros da equipe; ou privado para apenas uma parte da equipe (SLACK, 2017). Além disso, é possível integrar outras ferramentas ao Slack, para mostrar eventos como *commits* do GitHub e resultados de *builds* do TravisCI, entre outras, evitando ter que abrir múltiplas ferramentas para ver os resultados dos trabalhos dos outros. As ferramentas integradas com o Slack foram:

- GitHub;
- TravisCI;
- CodeCov;
- Coveralls; e
- Fastlane;

Todas as ferramentas foram integradas em um único canal para facilitar a organização dos eventos durante o desenvolvimento.

Os requisitos estão sendo mantidos no GitHub como *issues* para controlar o *status* no *kanban* do Guimifiu (2017b) na ferramenta Waffle.io. O Waffle é uma ferramenta de gestão de projetos que utiliza as *issues* do GitHub para manter os requisitos de um projeto em um *kanban* online automaticamente. Além disso, o Waffle também gera gráficos para monitorar o projeto (TECHNOLOGIES, 2017).

3.4 Arquitetura

3.4.1 Representação da Arqitetura

A forma como o *framework* Ionic está sendo utilizado pode ser representada por um diagrama de pacotes (Figura 5):

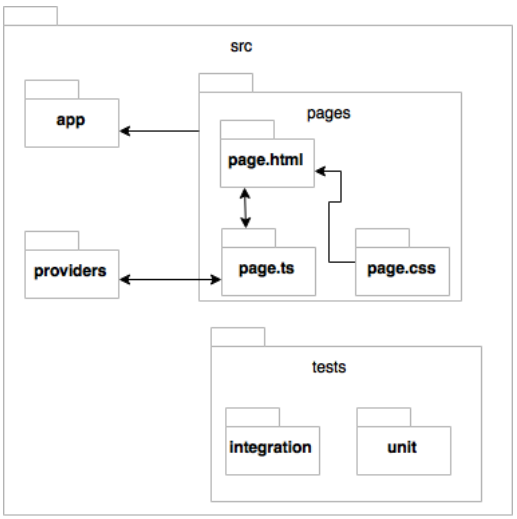

Figura 5: Diagrama de pacotes. Fonte: autores

- ***src***: é o diretório fonte da aplicação, onde os principais recursos serão salvos.
- ***app***: onde o *framework* reúne todos os recursos a serem compilados.
- ***pages***: as páginas do aplicativo, como o login, a página de registro de contas, entre outras. Cada página é composta de HTML e CSS, que compõem o visual da página e o Typescript que é onde está a parte lógica;
- ***providers***: os métodos que fazem a comunicação da API com o aplicativo.
- ***tests***: os diferentes tipos de testes de software realizados no applicativo.
- ***integration***: testes de integração.
- ***unit***: testes de unidade.

A arquitetura da API é baseada na arquitetura MVC (*Model View Controller*) e foi definida como ilustra a Figura 6:

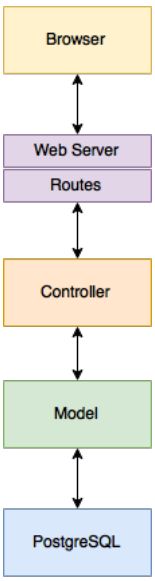

Figura 6: Arquitetura da API. Fonte: autores

As classes modelos se comunicam com o banco de dados *PostgreSQL* e com as classes controladoras, que por sua vez enviam e recebem dados para as rotas que são disponibilizadas por um servidor web; o usuário pode acessar esses dados via navegador ou outro tipo de aplicação cliente.

3.4.2 Modelo de Dados

Utilizando a ferramenta MySQL Workbench, foi criado um modelo de dados da aplicação. O dicionário de dados pode ser encontrado no Apêndice A. O modelo da aplicação é representado pela Figura 7.

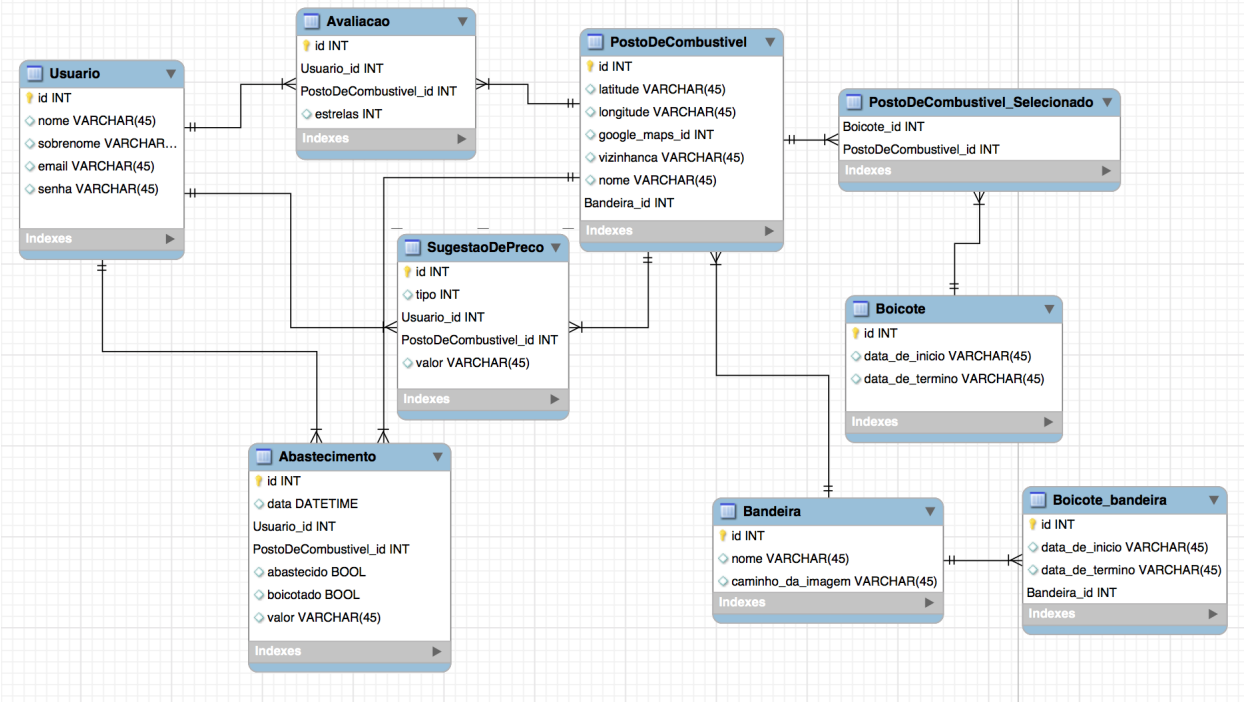

Figura 7: Modelo de dados Fonte: autores

4 Resultados

Neste capítulo, serão discutidos os resultados obtidos durante as duas entregas. Primeiramente, serão mostrados os requisitos elicitados pela equipe e, logo depois, serão mostrados os resultados da aplicação desenvolvida, onde serão passados os requisitos já implementados, o método de decisão de boicote, os testes automatizados, as métricas coletadas do código-fonte e a integração e *deploy* contínuo do projeto.

4.1 Aplicação desenvolvida

As histórias de usuário foram elicitadas pela equipe de desenvolvimento junto ao professor orientador.

Para a segunda entrega do aplicativo, foram desenvolvidos os requisitos já elicitados na primeira entrega conforme mostra a Tabela 2.

Tabela 2: Status dos Requisitos elicitados na primeira entrega

História	**Desenvolvida?**
Eu, como motorista, gostaria de me cadastrar no sistema para acessá-lo	Sim
Eu, como motorista, gostaria de me autenticar no sistema para utilizar suas funcionalidades	Sim
Eu, como motorista, gostaria de ver os postos dentro de uma rota para facilitar a minha escolha de posto	Sim
Eu, como motorista, gostaria de avaliar os preços já existentes para colocar preços verdadeiros em cima e falsos em baixo	Não
Eu, como motorista, gostaria de avaliar o posto onde eu abasteci para atribuir uma nota para sua qualidade	Sim
Eu, como motorista, gostaria de informar se estou ou não abastecendo no posto para contribuir com os dados coletados pelo aplicativo	Sim
Eu, como motorista, gostaria de ver uma lista de postos com melhor custo/benefício para abastecer no melhor posto para mim	Não
Eu, como motorista, gostaria de ver uma lista com todos os postos boicotados para me programar onde abastecer	Não
Eu, como motorista, gostaria de saber quando eu estou em um posto de combustíveis para interagir com ele	Sim
Eu, como motorista, gostaria de ver as informações do posto onde estou, para saber se quero abastecer nele ou não	Sim
Eu, como motorista, gostaria de registrar o preço de gasolina, álcool ou diesel do posto onde eu me encontro para que o preço esteja sempre correto	Sim
Eu, como desenvolvedor, desejo mapear os postos de combustíveis para que sejam tratados no sistema	Sim
Eu, como motorista, gostaria de editar o meu cadastro para manter minhas informações atualizadas	Sim
Eu, como motorista, desejo abrir a rota de um posto de combustível com outros softwares de navegação para conseguir chegar à ele	Sim

A Figura 8 mostra a tela de login da aplicação. O login pode ser feito via Facebook, Google ou criando uma conta local.

Figura 8: Tela de login. Fonte: autores

Como ilustrado na Figura 9, são mostrados na aplicação todos os postos de combustíveis em um raio de 5 km da localização do usuário.

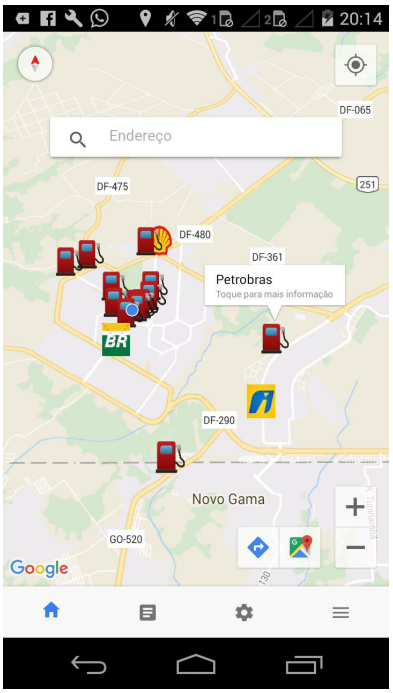

Figura 9: Postos de combustíveis próximos à localização do usuário. Fonte: autores

A Figura 10 mostra uma tela da aplicação assim que se chega em um posto boicotado.

Figura 10: Tela de chegada no posto boicotado. Fonte: autores

Além desses requisitos, foram elicitados as novas histórias que se mostraram imprescindíveis para a aplicação. O requisitos e seus respectivos status se encontram na Tabela 3

Tabela 3: Status dos Requisitos elicitados na segunda entrega

História	**Desenvolvida?**
Eu, como administrador, gostaria de me autenticar no sistema para ajustar suas configurações básicas	Sim
Eu, como administrador, gostaria de adicionar as bandeiras dos postos para facilitar a identificação pelo usuário	Sim
Eu, como administrador, gostaria de alterar as informações dos postos cadastrados para garantir a verossimilhança dos dados vindos de fontes externas ao sistema	Sim
Eu, como administrador, gostaria de criar um boicote por bandeiras para que o boicote seja efetivo diretamente à uma bandeira específica	Sim
Eu, como motorista, gostaria de visualizar um gráfico contendo os gastos mensais com combustível para controlar melhor meu orçamento	Sim
Eu, como motorista, gostaria de visualizar um histórico de postos onde passei para controlar melhor os meus gastos	Sim
Eu, como desenvolvedor, gostaria de criar um algoritmo para definir quais serão os postos boicotados	Sim

As funcionalidades de administrador se mostraram imprescindíveis para facilitar a identificação dos postos de combustíveis pelo usuário através de ícones, indicando as bandeiras. Durante o segundo período do projeto, observou-se que algumas informações fornecidas pela API do Google possuíam erros. Com o módulo de administração, é possível corrigir esses erros pontuais. Além disso, como a primeira versão do aplicativo que vai para a Play Store não possuirá as funcionalidades que garantirão a qualidade dos dados de preços dos combustíveis, a possibilidade de editar os dados de sugestão de preço manualmente, de forma mais prática, mostrou-se importante. A Figura 11 mostra a página inicial do módulo de administração.

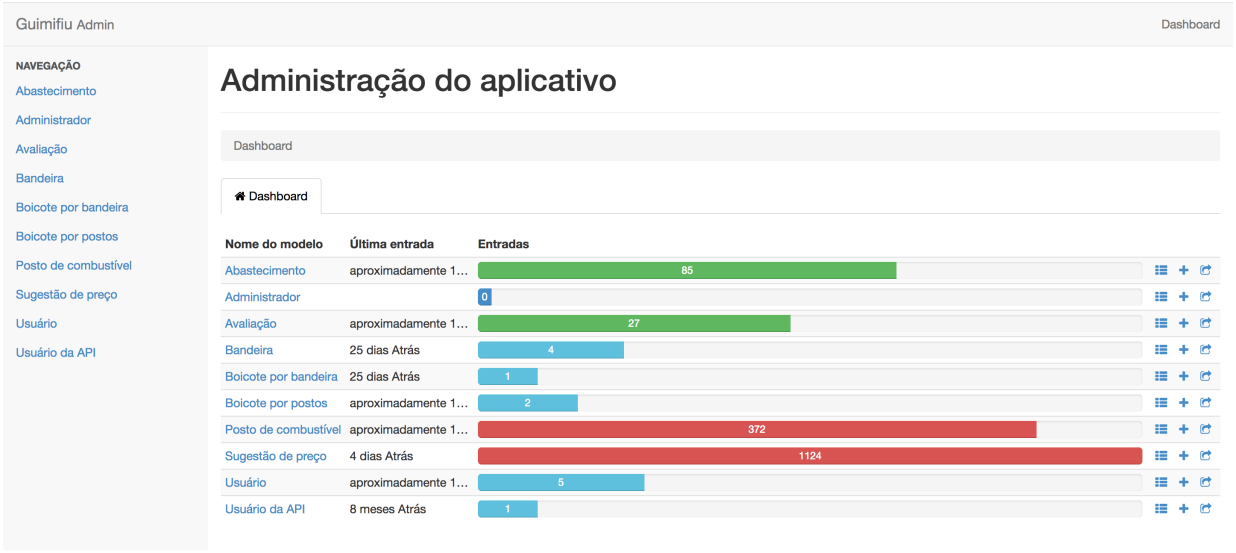

Figura 11: Página inicial do módulo de administração. Fonte: autores

Observou-se também a importância de manter um controle na aplicação sobre o controle de gastos do motorista e como essa funcionalidade é um diferencial em relação aos aplicativos atuais. A Figura 12 ilustra o histórico de postos abastecidos pelo usuário, enquanto a Figura 13 mostra o gráfico de gastos mensais.

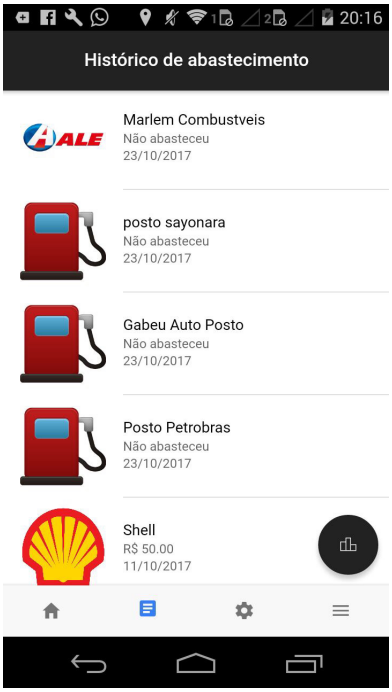

Figura 12: Página inicial do módulo de administração. Fonte: autores

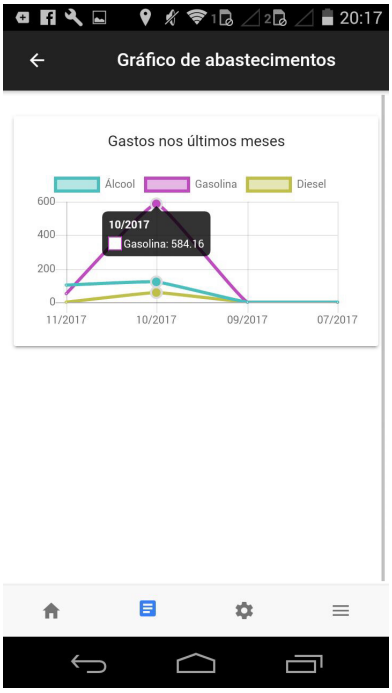

Figura 13: Página inicial do módulo de administração. Fonte: autores

Com as funcionalidades implementadas, a comunicação dos sistemas desenvolvidos se dá como ilustrado pela Figura 14:

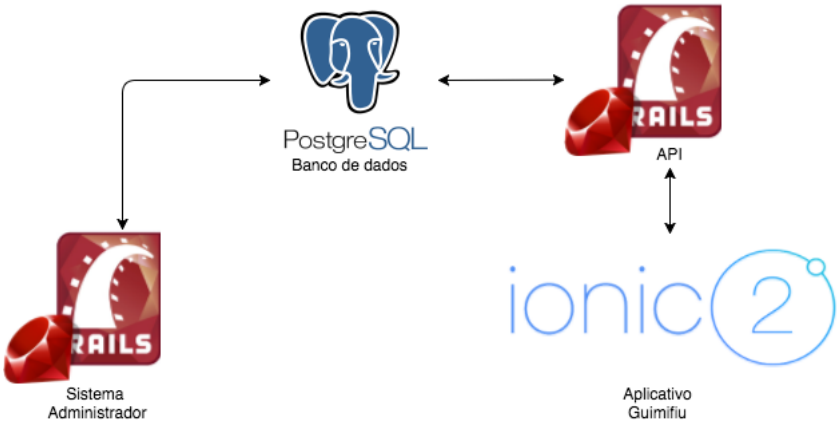

Figura 14: Comunicação entre os sistemas desenvolvidos. Fonte: autores

Os dados armazenados em um banco de dados PostgreSQL são transacionados pela interface administradora e pela API. Por sua vez, aplicativo se comunica com a API para acessar, criar e alterar os dados.

O Apêndice B apresenta algumas outras telas da aplicação desenvolvida.

4.1.1 Método de decisão de postos boicotados

Toda semana, serão selecionados postos onde os usuários serão incentivados a não abastecer. Para decidir quais postos serão boicotados, o Guimifiu tem duas estratégias principais: a **Lista Comum** e o **Boicote por Bandeiras**. A lista comum escolhe os postos com qualidade geral menor que 1 e que não tenham sido boicotados recentemente, criando uma lista inicial. A qualidade geral de um posto é determinada pela fórmula a seguir, onde são considerados o número de estrelas do posto e o preço dos combustíveis:

$$QG = \frac{5 * QE}{2 * (PG + PA + PD)} \tag{4.1}$$

Sendo que QG é a Qualidade Geral do posto, PG, PA e PD são os preços de Gasolina, Álcool e Diesel, respectivamente e, por fim, QE é a Quantidade de Estrelas. A quantidade de estrelas é definida pela média das avaliações dos usuários. O objetivo é que a qualidade do posto indicado pelos usuários seja proporcional à qualidade geral dele e que os preços dos combustíveis sejam inversamente proporcionais, ou seja, quanto maior a quantidade de estrelas e menor o preço dos combustíveis, maior a qualidade geral do posto, diminuindo suas chances do mesmo ser boicotado. Essa fórmula será balanceada de acordo com os dados dos postos e das avaliações dos usuários com o tempo. A tabela 4

contém um exemplo de postos fictícios com preços, quantidade de estrelas e a qualidade geral.

Tabela 4: Tabela de postos fictícios

Posto	**PG**	**PA**	**PD**	**QE**	**QG**
1	4,00	3,00	4,00	2	0,45
2	3,90	3,20	4,50	5	1,07
3	4,00	3,50	3,70	4,5	1,00
4	4,00	3,80	3,90	3,5	0,74
5	4,00	3,80	3,90	4,5	0,96

Na tabela 4, os postos 1, 4 e 5 seriam os colocados na lista inicial, devido ao baixo QG. Os postos 2 e 3 estariam logo acima do limite mínimo e, portanto, estariam de fora da lista.

A lista inicial é montada e então a aplicação verifica se não existem muitos postos próximos sendo boicotados, para não deixar o usuário com poucas opções de locais de abastecimento. Essa verificação é feita através de grafos. Os nós dos grafos são os postos de combustíveis e as arestas são as distâncias entre os todos postos que não sejam maiores que 10 quilômetros. Com este grafo, em cada nó da lista inicial é verificado se os nós adjacentes são todos postos que estão na lista inicial. Em caso positivo, o nó é removido da lista; caso contrário, o nó é mantido. Os nós restantes compõem a lista final de postos boicotados, denominada **Lista Comum**. Na Figura 15, considerando que os pontos vermelhos são nós da lista inicial e os pontos azuis são os demais postos, após a verificação, os pontos vermelhos virariam azuis, como mostrado na Figura 16.

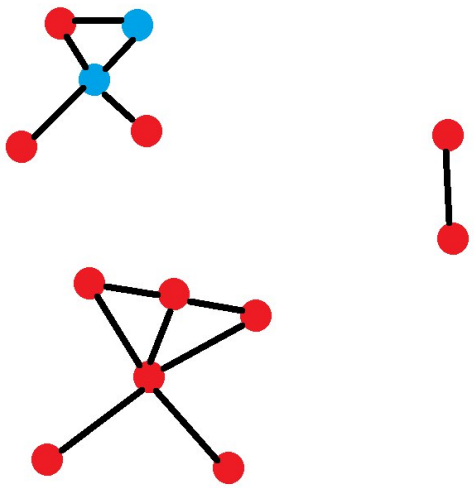

Figura 15: Grafo com Lista Inicial de Postos Boicotados. Fonte: autores

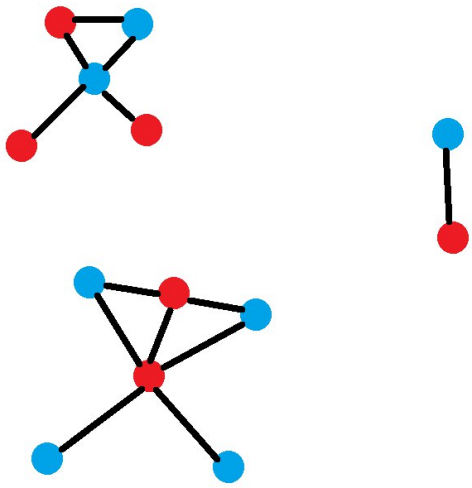

Figura 16: Grafo com Lista Comum de Postos Boicotados. Fonte: autores

Baseado no algoritmo, foi desenvolvido um script na linguagem Ruby que é executado semanalmente no servidor hospedado no Heroku, através do *Heroku Scheduler*, uma ferramenta criada para executar tarefas no servidor.

O **Boicote por Bandeiras** acontece quando todos os postos de uma bandeira previamente selecionada são colocados na lista de boicotados, independentemente da distância entre eles, colocando o prejuízo do boicote menos nos postos e mais na bandeira. Os dois tipos de boicote são exclusivos, ou seja, não ocorrem no mesmo período. A estratégia de boicote é escolhida pela equipe de desenvolvimento. O **Boicote por Bandeiras** é iniciado manualmente no módulo de administração, como mostra a Figura 17

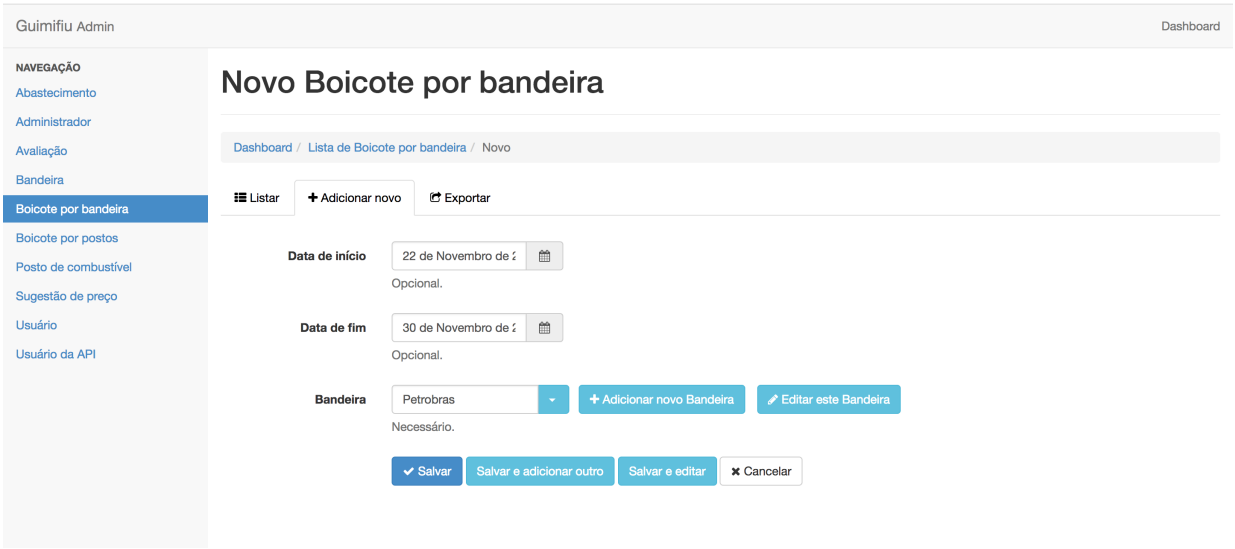

Figura 17: Página de criação de Boicotes do módulo de administração. Fonte: autores

4.1.2 Testes Automatizados

Foram feitos testes de integração e unitários para garantir o funcionamento dos métodos do código-fonte e da integração da aplicação *mobile* com a API.

Encontrou-se uma certa dificuldade pra realizar os testes unitários no Ionic, uma vez que não há muitos exemplos que de fato funcionam e uma documentação do próprio *framework* sobre o assunto.

O código de testes pode ser encontrado no repositório da aplicação no GitHub (https://github.com/Guimifiu/guimifiu-app). Um exemplo de teste unitário e de teste de integração está disponível no Apêndice C.

4.1.3 Métricas de Código-fonte

Como citado no Capítulo 3, foram utilizadas as ferramentas CodeClimate e Rubocop. Além de medir a complexidade no Rubocop, foi feita na integração contínua um relatório de métricas pela ferramenta que impede que a *build* passe caso a complexidade seja maior do que 6, que é o padrão de complexidade ciclomática máxima aceitável.

A Figura 18 mostra o resultado da análise feita pelo Rubocop. O arquivo de configuração utilizado para a análise pode ser encontrado no endereço do GitHub do Guimifiu (2017a).

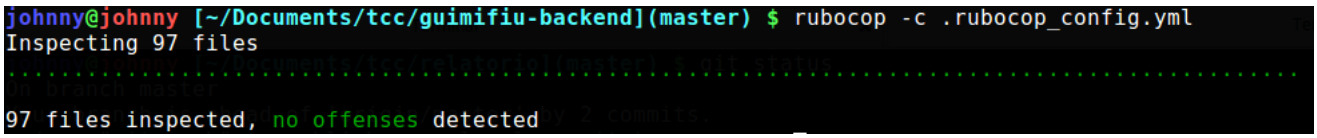

Figura 18: Relatório de análise estática do Rubocop. Fonte: autores

A Figura 19 mostra o resultado das análises realizado pelo CodeClimate.

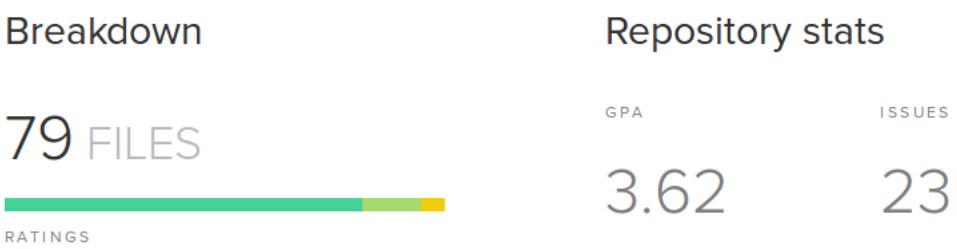

Figura 19: Relatório de análise estática do CodeClimate. Fonte: autores

As *issues* encontradas no CodeClimate infelizmente não puderam ser tratadas a tempo da entrega do projeto devido aos problemas encontrados durante o desenvolvimento, como descritos no Capítulo 5.

4.1.4 Integração e Entrega Contínua

Há três tipos de ambientes no desenvolvimento desta aplicação: *development*, *staging* e *production*. O ambiente de *development*, ou desenvolvimento, é o ambiente local de cada desenvolvedo; *staging* ou pré-produção é um ambiente para serem realizados testes beta e *production*, ou produção, é o ambiente final onde a aplicação será de fato utilizada pelos usuários.

Na API, a integração contínua é feita com a ferramenta TravisCI, onde, quando um *pull request* é aceito na *branch staging*, o Travis executa toda a suíte de testes para identificar se algum deles pode ter resultado em falha. Caso todos passem, ele verifica se todas as métricas definidas pelo Rubocop estão de acordo com o padrão definido. Caso todas as métricas passem, a *build* de pré produção é criada, o Travis envia as informações de cobertura de testes para o Coveralls, as informações de métricas para o Code Climate e em seguida realiza o *deploy* no ambiente de pré produção no Heroku.

Se um *pull request* for aceito na *branch master* todo o processo será o mesmo, exceto pelo *deploy*, que ocorrerá no ambiente de produção no Heroku.

A Figura 20 ilustra o processo de integração e *deploy* contínuo feito na API.

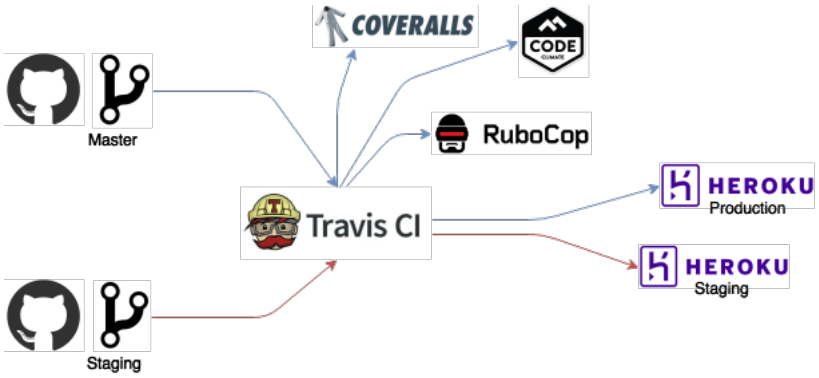

Figura 20: Integração e *deploy* contínuo API. Fonte: autores

No aplicativo planejou-se a integração e o *deploy* contínuo como mostra a Figura 21.

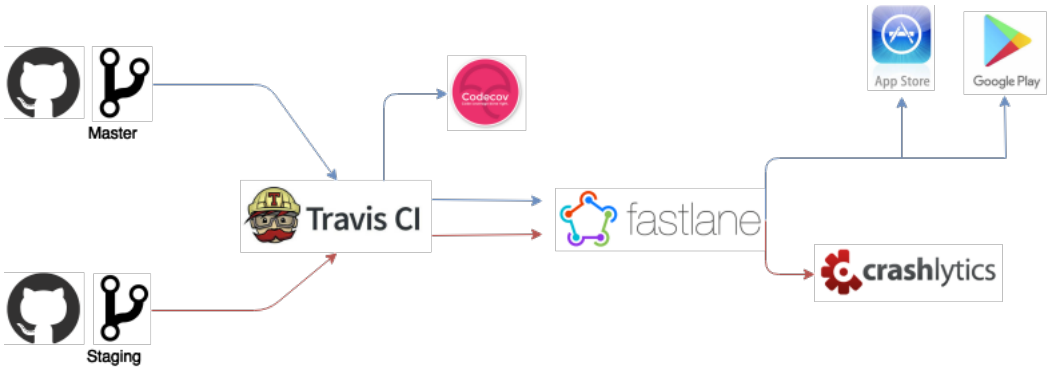

Figura 21: Integração e *deploy* contínuo planejado para o aplicativo. Fonte: autores

Se um *pull request* for aceito na *branch master*, o Travis realizará toda a suíte de testes; caso todos passem, ele enviará as informações de cobertura para o Codecov e através do Fastlane realizará o *deploy* do aplicativo, tanto na Play Store quanto na App Store. Se um *pull request* for aceito na *branch staging*, todo o processo será repetido, exceto pelo fato que o deploy do aplicativo não é realizado nas lojas e sim no Crashlytics, que irá criar a versão beta do aplicativo e o enviará para os e-mails previamente configurados.

Para a primeira entrega do trabalho, foi planejado só até o *deploy* beta, como mostrado na Figura 22:

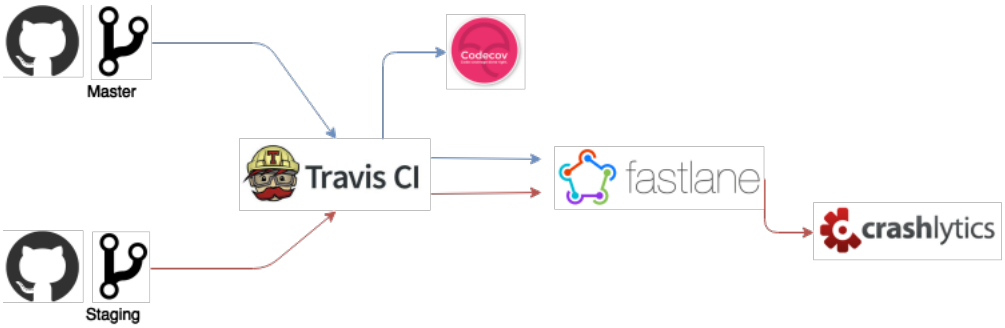

Figura 22: Integração e *deploy* contínuo planejado para a primeira entrega. Fonte: autores

Entretanto, não foi possível fazer com que o *deploy* contínuo acontecesse logo após a integração contínua, um desafio causado pela escolha de desenvolvimento híbridos de aplicativo. Uma vez que o código versionado do aplicativo é o mesmo código para a plataforma Android e iOS, e após realizar a *build* desse código são gerados códigos em cada plataforma, não versionados. A ferramenta de integração contínua não consegue ter acesso aos códigos de cada plataforma para realizar os devidos *deploys*. Sendo assim, o *deploy* contínuo estava acontecendo manualmente através do comando:

```
$ fastlane beta
```

A Figura 23 ilustra como estava funcionando a integração e o *deploy* contínuo do aplicativo até a primeira entrega.

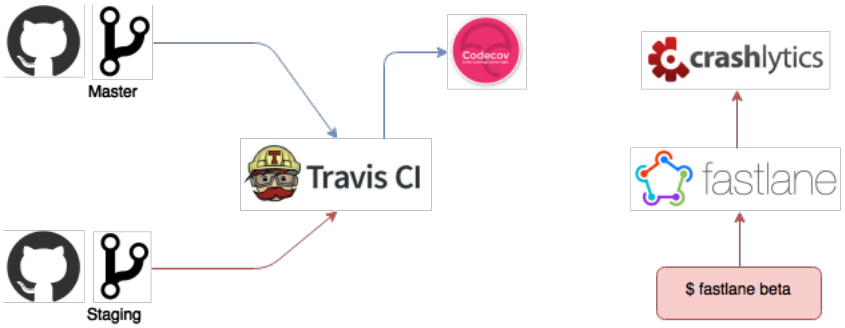

Figura 23: Integração e *deploy* contínuo até a primeira entrega. Fonte: autores

Na segunda parte do trabalho foram avaliadas formas de melhorar essa automatização. Entretanto, com o lançamento da versão 3 do Ionic, muitas dependências pararam de funcionar e a automatização, que deveria economizar tempo, rodando os testes e integrando as novas funcionalidades ao aplicativo e realizando o *deploy* nas lojas, começou a ocasionar perda de tempo de desenvolvimento na tentativa de integrar ferramentas para as quais o Ionic não oferece suporte. Portanto, decidiu-se remover a integração e entrega contínua do aplicativo e realizar essas tarefas manualmente, deixando assim essa melhoria como proposta para trabalhos futuros.

5 Cronograma

A Figura 24 mostra o cronograma das atividades realizadas durante a primeira parte do projeto.

	Name	Duration	Start	Finish
1	Definir escopo do projeto	11d?	03/01/2017	03/15/2017
2	Elicitar requisitos	11d?	03/15/2017	03/29/2017
3	Criar protótipos	6d?	03/22/2017	03/29/2017
4	Implementar login	11d?	03/29/2017	04/12/2017
5	Implementar testes unitários e de integração	16d?	04/12/2017	05/03/2017
6	Implementar integração e deploy contínuo	11d?	05/03/2017	05/17/2017
7	Criar modelo de dados	6d?	05/17/2017	05/24/2017
8	Implementar mapa com postos e rotas	11d?	05/17/2017	05/31/2017
9	Escrever relatório	13d?	05/31/2017	06/16/2017

Figura 24: Cronograma de atividades da primeira entrega. Fonte: autores

A Figura 25 apresenta o cronograma inicial da segunda parte do projeto.

Name	Duration	Start	Finish
Sprint 1	11d?	**08/30/2017**	**09/13/2017**
Eu, como motorista, gostaria de saber quando eu estou em um posto de gasolina para interagir com o mesmo	11d?	08/30/2017	09/13/2017
Sprint 2	11d?	**09/13/2017**	**09/27/2017**
Eu, como motorista, gostaria de informar se estou ou não abastecendo no posto para contribuir com os dados coletados pelo aplicativo	3d?	09/13/2017	09/15/2017
Eu, como motorista, gostaria de ver as informações do posto onde eu estou, para saber se quero abastecer nele ou não	4d?	09/15/2017	09/20/2017
Eu, como motorista, gostaria de registrar o preço de gasolina, álcool ou diesel do posto onde eu me encontro para ele sempre ter o preço certo	6d?	09/20/2017	09/27/2017
Sprint 3	11d?	**09/27/2017**	**10/11/2017**
Eu, como motorista, gostaria de ver uma lista com todos os postos boicotados para me programar com antecedência onde abastecer	11d?	09/27/2017	10/11/2017
Sprint 4	11d?	**10/11/2017**	**10/25/2017**
Eu, como motorista, gostaria de ver uma lista de postos com maior custo/benefício para abastecer no melhor posto para mim	11d?	10/11/2017	10/25/2017
Sprint 5	11d?	**10/25/2017**	**11/08/2017**
Eu, como motorista, gostaria de avaliar os preços já existentes para colocar preços verdadeiros em cima e falsos em baixo	11d?	10/25/2017	11/08/2017
Sprint 6	11d?	**11/08/2017**	**11/22/2017**
Eu, como motorista, gostaria de avaliar o posto onde eu abasteci para definir uma qualidade ao mesmo	11d?	11/08/2017	11/22/2017

Figura 25: Cronograma inicial de atividades da segunda entrega. Fonte: autores

Porém, devido a certas adversidades e melhor compreensão do projeto, foram feitas certas mudanças no cronograma, como mostra a Figura 26.

Name	Duration	Start	Finish
Sprint 1	11d?	08/30/2017	09/13/2017
Problemas com ambiente de desenvolvimento	11d?	08/30/2017	09/13/2017
Sprint 2	11d?	**09/13/2017**	**09/27/2017**
Eu, como motorista, gostaria de saber quando eu estou em um posto de gasolina para interagir com o mesmo	4d?	09/13/2017	09/18/2017
Eu, como motorista, gostaria de ver as informações do posto onde eu estou, para saber se quero abastecer nele ou não	6d?	09/18/2017	09/25/2017
Eu, como motorista, gostaria de informar se estou ou não abastecendo no posto para contribuir com os dados coletados pelo aplicativo	3d?	09/25/2017	09/27/2017
Sprint 3	11d?	**09/27/2017**	**10/11/2017**
Eu, como motorista, gostaria de ter um histórico com todos os meus abastecimentos para manter registro dos meus gastos	4d?	09/27/2017	10/02/2017
Eu, como motorista, gostaria de registrar o preço de gasolina, álcool ou diesel do posto onde eu me encontro para ele sempre ter o preço certo	4d?	10/02/2017	10/05/2017
Eu, como motorista, gostaria de avaliar o posto onde eu abasteci para definir uma qualidade ao mesmo	5d?	10/05/2017	10/11/2017
Sprint 4	11d?	**10/11/2017**	**10/25/2017**
Eu, como desenvolvedor, gostaria de criar um algoritmo para definir quais serão os postos boicotados	4d?	10/11/2017	10/16/2017
Eu, como administrador, gostaria de ter uma interface web administradora para monitorar e controlar os dados do aplicativo	4d?	10/16/2017	10/19/2017
Eu, como motorista, gostaria de visualizar os meus gastos nos últimos meses para controlar meus gastos com combustíveis	2d?	10/19/2017	10/20/2017
Eu, como administrador, desejo poder criar um evento boicote por bandeira de postos de combustíveis para que o boicote tenha efeito direto sobre a bandeira	3d?	10/23/2017	10/25/2017
Sprint 5	11d?	**10/25/2017**	**11/08/2017**
Eu, como motorista, gostaria de ver quando um posto é boicotado para programar onde vou abastecer	4d?	10/25/2017	10/30/2017
Eu, como motorista, desejo abrir a rota de um posto de combustível com outros software de navegação para conseguir chegar ao destino	5d?	10/31/2017	11/06/2017
Realizar deploy	3d?	11/06/2017	11/08/2017
Sprint 6	5d?	**11/13/2017**	**11/17/2017**
Eu, como desenvolvedor, desejo refatorar o front-end do aplicativo para melhorar a usabilidade e o estilo	3d?	11/13/2017	11/15/2017
Melhorar deploy contínuo para evoluções	3d?	11/15/2017	11/17/2017

Figura 26: Cronograma final de atividades da segunda entrega. Fonte: autores

Principalmente na primeira *sprint*, alguns problemas foram encontrados com o ambiente de desenvolvimento do Ionic devido à da atualização do Ionic 2 para o Ionic 3. A maioria dos problemas estavam relacionados a atualizações de dependências.

O *plugin* do *Geofence*, utilizado para definir cercas geográficas nos postos de combustível para indicar ao usuário quando ele entra em um posto, foi implementado para o iOS na linguagem de programação Swift. A Apple atualizou o Swift da versão 2 para a versão 3 durante o desenvolvimento, mas o *plugin* ainda não foi atualizado, o que gerou incompatibilidade do serviço oferecido pelo *plugin* com a plataforma iOS. Mesmo depois de muitas tentativas de fazê-lo funcionar nesta plataforma, não foi obtido êxito. Portanto, a equipe de desenvolvimento, juntamente com o professor orientador, tomaram a decisão de pausar o desenvolvimento para a plataforma iOS e focar apenas para a plataforma Android, evitando assim que todo o trabalho fosse prejudicado pela atualização de uma dependência que ainda não ocorreu.

Além do problema da plataforma da Apple, apesar do *plugin Geofence* ter suporte para notificações *push* (mensagens de alerta enviadas para o usuário notificando na tela do aparelho) locais, uma opção para facilitar a escalabilidade do aplicativo, caso seja necessário enviar notificações *push* em outros momentos, é a utilização de algum serviço externo para enviar as notificações. Essa decisão causou novos atrasos na implementação da funcionalidade, pois foi necessário realizar pesquisas sobre novas alternativas. Por fim, foi utilizado o serviço de notificações *push* do Google, o Firebase.

Durante a sprint 4, percebeu-se que seria necessário uma tela de administração para poder definir os ícones dos postos, bem como corrigir quaisquer informações incorretas oriundas das fontes externas, o que justifica sua adição ao cronograma. Durante a mesma sprint, viu-se necessário melhorar a interface do aplicativo pois o foco até então havia sido exclusivamente em desenvolver as funcionalidades. Porém, essa melhoria foi alocada para a sprint 6 porque a sprint 4 e a sprint 5 já possuíam muitas tarefas.

6 Conclusões

6.1 Considerações Finais

Com o fechamento do trabalho tem-se um aplicativo móvel com impacto social, com intuito de melhorar o controle sobre postos de combustíveis por parte da sociedade. Conceitos de Engenharia de Software como, métricas de código, desenvolvimento ágil, testes de software automatizados, prototipação, integração e entrega contínua foram utilizadas para chegar ao resultado final.

Fazer a aplicação utilizando o Ionic 2 foi um grande risco. Infelizmente, devido aos problemas nas atualizações do Ionic e os plugins utilizados, a maior vantagem de se fazer uma aplicação em Ionic ao invés de nativo não foi utilizada, que é ter aplicativos para as duas principais plataformas com um só código. Aplicar conceitos de engenharia de software em um *framework* relativamente novo ainda crescendo uma comunidade em torno dele foi uma experiência de grande valia, porém a conclusão que se teve foi que para projetos de grande porte o desenvolvimento nativo ainda é mais seguro.

O Ionic e seus criadores têm demostrado que a plataforma ainda irá crescer muito e espera-se que em suas versões subsequentes à versão 2, problemas com dependências e com o ambiente de desenvolvimento sejam resolvidos fazendo com que a plataforma seja mais estável. Para o desenvolvimento da API, não houve grandes problemas. O Ruby é uma linguagem muito estável e todos os pequenos problemas encontrados foram facilmente resolvidos devido a extensa documentação e comunidade.

O módulo de administração se mostrou muito útil inclusive nas fases de desenvolvimento, garantindo que os dados fossem efetivamente salvos e facilitando a criação de dados fictícios para testes. Na continuação deste projeto, certamente serão mantidos o módulo de administração assim como a API, e estudada a possibilidade de migração para a versão mais recente do Ionic ou para desenvolvimento nativo.

6.2 Trabalhos Futuros

Para a continuidade do projeto, tem-se as seguintes atividades:

- **Integração e *Deploy* Contínuo:** novas políticas de qualidade de código serão adicionadas à integração com o intuito de evitar o aceite de código de baixa qualidade no repositório. Implementar soluções para os problemas encontrados com a integração e entrega contínua do aplicativo.

- **Testes de usabilidade:** serão realizados testes de usabilidade com um grupo controlado de usuários, para garantir que o processo do aplicativo esteja funcional;
- **Protótipos de alta fidelidade:** também para a realização de testes de usabilidade, porém com o intuito de validar se o processo do aplicativo está fácil de ser realizado e intuitivo;
- **Definir estratégias de marketing:** é de suma importância que o aplicativo tenha visibilidade para poder garantir a integridade dos dados que serão atualizados pelos próprios usuários;
- **Testes de desempenho:** para estudar melhor a escalabilidade do aplicativo, serão feitos testes de desempenho como testes de carga, testes de *stress* e testes de volume.

Referências

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISO/IEC 9126-1*: Engenharia de software - qualidade de produto. Rio de Janeiro, 2003. Citado na página 19.

BASS, J. M. Scrum master activities: Process tailoring in large enterprise projects. In: *2014 IEEE 9th International Conference*. [S.l.: s.n.], 2014. p. 6–15. Citado na página 18.

BATSOV, B. *About RuboCop*. 2017. Acessado em 03-06-2017. Disponível em: <http://batsov.com/rubocop/>. Citado na página 25.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change*. 2nd. ed. [S.l.]: Addison-Wesley, 2004. Citado na página 20.

BECK, K. et al. *Agile Manifesto*. 2001. Acessado em 25-05-2017. Disponível em: <http://agilemanifesto.org/iso/ptbr/principles.html>. Citado na página 20.

BRAZILIENSE, C. *Descoberta de cartel não reduz o preço da gasolina*. 2015. Acessado em 04-06-2017. Disponível em: <http://www.correiobraziliense.com.br/app/noticia/economia/2015/12/30/internas_economia,512351/descoberta-de-cartel-nao-reduz-o-preco-da-gasolina.shtml>. Citado na página 14.

BRAZILIENSE, C. *Brasilienses preparam boicote aos postos de combustíveis para sexta-feira*. 2016. Acessado em 04-06-2017. Disponível em: <http://www.correiobraziliense.com.br/app/noticia/cidades/2016/01/21/interna_cidadesdf,514755/brasilienses-preparam-boicote-aos-postos-de-combustiveis-para-sexta-fe.shtml>. Citado na página 14.

CODECLIMATE. *About CodeClimate*. 2017. Acessado em 03-06-2017. Disponível em: <https://codeclimate.com/>. Citado na página 25.

CRUZ, F. *Scrum e PMBOK unidos no Gerenciamento de Projetos*. [S.l.]: Brasport, 2013. Citado na página 19.

DALMASSO, I. et al. Survey, comparison and evaluation of cross platform mobile application development tools. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. Sardinia, Italy: [s.n.], 2013. Citado na página 18.

DAVIS, R. About flog. 2017. Acesso em: 2017-06-24. Disponível em: <http://ruby.sadi.st/Flog.html>. Citado na página 19.

DRIFTYCO. *About Ionic*. 2017. Acessado em 09-06-2017. Disponível em: <https://ionicframework.com/about>. Citado na página 24.

DWLLO, R. et al. Software testing and evaluation. In: *BanjaminKummings Publishing Company, Inc.* California: [s.n.], 1987. Citado na página 21.

FASTLANE. *About Fastlane*. 2017. Acessado em 10-06-2017. Disponível em: <https://fastlane.tools/>. Citado na página 26.

FENTON, N.; BIEMAN, J. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. [S.l.]: Taylor and Francis Group, 2015. Citado na página 19.

FOWLER, M. Continuous integration. In: . [S.l.: s.n.], 2006. Citado na página 20.

GANNON, M. An agile implementation of serum. In: . [S.l.: s.n.], 2013. Citado na página 18.

GARTNER. *Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016*. 2016. Acesso em: 2017-05-21. Disponível em: <http://www.gartner.com/newsroom/id/3415117>. Citado na página 17.

GLOBO, O. *Veja os 10 países com gasolina mais cara*. 2013. Acessado em 04-06-2017. Disponível em: <http://infograficos.oglobo.globo.com/economia/veja-os-10-paises-com-gasolina-mais-cara.html>. Citado na página 14.

GOOGLE. About android. 2017. Acesso em: 2017-05-21. Disponível em: <https://developer.android.com/about/android.html>. Citado na página 17.

GOOGLE. About google maps. 2017. Acesso em: 2017-05-24. Disponível em: <https://www.google.com/intl/pt-BR/maps/about/>. Citado na página 17.

GUIMIFIU. *Arquivo de configuração de métricas*. 2017. Acessado em 07-06-2017. Disponível em: <https://github.com/Guimifiu/guimifiu-backend/blob/master/.rubocop_config.yml.> Citado na página 40.

GUIMIFIU. *Kanban do projeto*. 2017. Disponível em: <https://waffle.io/Guimifiu/guimifiu-app>. Citado na página 28.

HANSSON, D. H. *About Ruby on Rails*. 2017. Acessado em 09-06-2017. Disponível em: <http://rubyonrails.org/>. Citado na página 24.

HAYATA; TOMOHIRO; HAN, J. A hybrid model for it project with scrum. In: *2011 IEEE International Conference*. [S.l.: s.n.], 2011. p. 285–290. Citado na página 18.

HEROKU. *What is Heroku?* 2017. Acessado em 10-06-2017. Disponível em: <https://www.heroku.com/what>. Citado na página 26.

HOFMANN-WELLENHOF, B.; LICHTENEGGER, H.; COLLINS, J. *Global Positioning System. Theory and Practice*. [S.l.]: Springer, 1993. Citado na página 17.

HUMBLE, J.; FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. [S.l.]: Addison-Wesley Professional, 2010. Citado na página 20.

INDUSTRIES, L. H. *About Coveralls*. 2017. Acessado em 04-06-2017. Disponível em: <https://coveralls.io/>. Citado na página 25.

INTERNATIONAL ORGANISATION FOR STANDARDISATION. *ISO 9241-11*: Ergonomic requirements for office work with visual display terminals (vdts). part 11 — guidelines for specifying and measuring usability. [S.l.], 1997. Citado na página 21.

KARMA. *About Karma*. 2017. Acessado em 03-06-2017. Disponível em: <https://karma-runner.github.io/1.0/index.html>. Citado na página 25.

KNAPP, J. *The product design sprint: diverge (day 2)*. 2012. Acessado em 11-06-2017. Disponível em: <https://library.gv.com/the-product-design-sprint-diverge-day-2-c7a5df8e7cd0>. Citado na página 22.

KOPITZ, D.; MARKS, B. *RDS: The Radio Data System*. [S.l.]: Artech House Books, 1998. Citado na página 17.

LABS, P. *About Jasmine*. 2017. Acessado em 03-06-2017. Disponível em: <https://jasmine.github.io/>. Citado na página 25.

LIM, S. Experimental comparison of hybrid and native applications for mobile systems. In: *International Journal of Multimedia and Ubiquitous Engineering*. Hankuk University of Foreign Studies: [s.n.], 2015. v. 1. Citado 2 vezes nas páginas 17 e 18.

LUO, G.; PROBERT, R. L.; URAL, H. Approach to constructing software unit testing tools. In: *Software Engineering Journal*. [S.l.: s.n.], 1995. Citado na página 21.

MANTYLA, M. V.; VANHANEN, J. . software deployment activities and challenges - a case study of four software product companies. In: *15th European Conference on Software Maintenance and Reengineering*. [S.l.: s.n.], 2011. Citado na página 20.

MARVEL. *Marvel App Prototyping*. 2017. Acessado em 11-06-2017. Disponível em: <https://marvelapp.com/prototyping/>. Citado na página 26.

MEIRELLES, P. R. M. Monitoramento de métricas de código-fonte em projetos de software livre. In: . USP - São Paulo: [s.n.], 2013. Citado na página 19.

MICROSOFT. *About TypeScript*. 2017. Acessado em 04-06-2017. Disponível em: <https://www.typescriptlang.org/>. Citado na página 24.

MILLS, E. E. Software metrics. In: *Relatório técnico, Software Engineering Institute*. SEI - Carnegie Mellon University.: [s.n.], 1988. Citado na página 19.

MYERS, G. J. *The Art Of Software Testing*. 2nd. ed. [S.l.]: John Wiley & Sons, Inc., 2004. Citado na página 21.

NIELSEN, J. *UX Prototypes: Low Fidelity vs. High Fidelity*. 2017. <https://www.nngroup.com/articles/ux-prototype-hi-lo-fidelity/>. Acesso em: 2017-06-24. Citado 2 vezes nas páginas 21 e 22.

NINJAMOCK. *About NinjaMock*. 2017. Acessado em 11-06-2017. Disponível em: <https://ninjamock.com/>. Citado na página 26.

PALANTIR. *About TSLint*. 2017. Acessado em 03-06-2017. Disponível em: <https://palantir.github.io/tslint/>. Citado na página 25.

PAULO, F. de S. *Retrospectiva: Manifestações não foram pelos 20 centavos*. 2013. Acessado em 06-06-2017. Disponível em: <http://www1.folha.uol.com.br/poder/2013/12/1390207-manifestacoes-nao-foram-pelos-20-centavos.shtml>. Citado na página 14.

PRICES, G. P. *Preços da gasolina, litro*. 2017. Acessado em 04-06-2017. Disponível em: <http://pt.globalpetrolprices.com/gasoline_prices/>. Citado na página 14.

RADJENOVIć, D. et al. Software fault prediction metrics: A systematic literature review. In: *Information and Software Technology*. University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, SI-2000 Maribor, Slovenia: [s.n.], 2013. Citado na página 19.

RSPEC. *About RSpec*. 2017. Acessado em 04-06-2017. Disponível em: <http://rspec.info/about/>. Citado na página 25.

RUBIN, K. S. *Essential scrum*. [S.l.]: Addison-Wesley, 2012. Citado 2 vezes nas páginas 18 e 19.

RUBY. *About Ruby*. 2017. Acessado em 09-06-2017. Disponível em: <https://www.ruby-lang.org/pt/about/>. Citado na página 24.

RUNESON, P. A survey of unit testing practices. In: . [S.l.: s.n.], 2006. p. 22–29. Citado na página 21.

RUTHERFORD, K. About code smells. 2017. Acesso em: 2017-06-24. Disponível em: <https://github.com/troessner/reek/blob/master/docs/Code-Smells.md>. Citado na página 19.

RUTHERFORD, K. Reek quickstart. 2017. Acesso em: 2017-06-24. Disponível em: <https://github.com/troessner/reek#quickstart>. Citado na página 19.

SÁNCHEZ, D. V. et al. Student role functionalities towards learning management systems as open platforms through mobile devices. In: *2014 International Conference on Electronics, Communications and Computers*. Cholula, Mexico: [s.n.], 2014. Citado na página 17.

SERRANO, N.; HERNANTES, J.; GALLARDO, G. Mobile web apps. In: *IEEE Software*. [S.l.: s.n.], 2013. Citado na página 18.

SLACK. *About Slack*. 2017. Acessado em 09-06-2017. Disponível em: <https://slack.com/is>. Citado na página 27.

SOMMERVILLE, I. *Software Engineering*. 8th. ed. [S.l.]: Pearson Education Limited, 1982. Citado na página 21.

SPILLNER, A. Test criteria and coverage measures for software integration testing. In: *Software Quality Journal*. [S.l.: s.n.], 1995. p. 275–286. Citado na página 21.

SUTHERLAND, J. et al. Distributed scrum: Agile project management with outsourced development teams. In: *Proceedings o f the 40th Annual Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2007. p. 274. Citado na página 20.

TECHNOLOGIES, C. Waffle.io. 2017. Acesso em: 2017-11-19. Disponível em: <https://waffle.io>. Citado na página 28.

TRAVISCI. *About TravisCI*. 2017. Acessado em 09-06-2017. Disponível em: <https://travis-ci.org/>. Citado na página 26.

WAZE. About waze. 2017. Acesso em: 2017-05-22. Disponível em: <https://www.waze.com/pt-BR/about>. Citado na página 17.

WILLIAMS, L.; MAXIMILIEN, E. M.; VOUK, M. Test-driven development as a defect-reduction practice. In: *Proceedings of the 14th International Symposium on Software Reliability Engineering*. Denver, CO, U.S.A.: [s.n.], 2003. p. 34–45. Citado na página 21.

WORLD, N. *5 things you need to know about Continuous Delivery software development*. 2013. Acessado em 25-05-2017. Disponível em: <go.galegroup.com/ps/i.do?p=AONE&sw=w&u=capes&v=2.1&id=GALE%7CA341527985&it=r&asid=327e5e304e953b607815c37d26ce48df>. Citado na página 20.

Apêndices

APÊNDICE A – Dicionário de Dados

A Tabela 5 mostra o dicionário de dados da entidade *Usuario*, que armazena todas as informações dos usuários que se cadastrarem no aplicativo.

Tabela 5: Dicionário de dados da entidade *Usuario*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador do usuário
Nome	Texto	Nome do usuário
Sobrenome	Texto	Sobrenome do usuário
Email	Texto	E-mail do usuário
Senha	Texto	Senha do usuário. Deve possuir um mínimo de 8 caracteres

A Tabela 6 mostra o dicionário de dados da entidade *Avaliacao*, que armazena as notas dadas pelos usuários para os postos de combustíveis.

Tabela 6: Dicionário de dados da entidade *Avaliacao*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador da avaliação
Estrelas	Numérico	Classificação de um posto de gasolina. Pode ir de 0 a 5
Usuário Id	Numérico	Chave estrangeira da tabela *Usuario*
Posto de Combustível Id	Numérico	Chave estrangeira da tabela *PostoDeCombustivel*

A Tabela 7 mostra o dicionário de dados da entidade *PostoDeCombustivel*, que irá armazenar todas as informações dos postos de combustíveis importados da API do Google Maps além de informações adicionadas pelos usuários do Guimifiu.

Tabela 7: Dicionário de dados da entidade *PostoDeCombustivel*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador do posto de combus†ível
Latitude	Texto	Latitude do posto de combustível. Pode ir de 0 a 5
Longitude	Texto	Longitude do posto de combustível
Vizinhanca	Texto	Endereço da vizinhança onde o posto se localiza
Google Maps Id	Númerico	Chave da API do Google Maps
Nome	Texto	Nome do posto de combustível
Bandeira Id	Numérico	Chave estrangeira da tabela *Bandeira*

A Tabela 8 mostra o dicionário de dados da entidade *SugestaoDePreco*, que armazena informações sobre a sugestão de preço dos combustíveis em um posto, por usuários que já estiveram nele.

Tabela 8: Dicionário de dados da entidade *SugestaoDePreco*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador da sugestão de preço
Tipo	Numérico	Define o tipo de combustível (Gasolina, Álcool ou Diesel)
Valor	Numérico	Valor do combustível
Usuário Id	Numérico	Chave estrangeira da tabela *Usuario*
Posto de Combustível Id	Numérico	Chave estrangeira da tabela *PostoDeCombustivel*

A Tabela 9 mostra o dicionário de dados da entidade *Abastecimento*, que armazena as informações sobre abastecimentos de usuários em posto de combustíveis.

Tabela 9: Dicionário de dados da entidade *Abastecimento*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador do abastecimento
Data	Data	Data que o usuário abasteceu no posto
Abastecido	Boleano	Define se o usuário abasteceu ou não no posto em que parou
Boicotado	Boleano	Define se o posto em que o usuário parou estava na lista de postos boicotados
Valor	Numérico	Valor de combustível abastecido
Usuário Id	Numérico	Chave estrangeira da tabela *Usuario*
Posto de Combustível Id	Numérico	Chave estrangeira da tabela *PostoDeCombustivel*

A Tabela 10 mostra o dicionário de dados da entidade *Boicote*, que armazena os dados dos eventos chamados de boicotes por lista comum a postos de combustíveis, que aconteceram com uma periodicidade definida.

Tabela 10: Dicionário de dados da entidade *Boicote*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador do boicote
Data de Início	Data	Data de início da lista de boicote
Data de Término	Data	Data final da lista de boicote

Apresentado na Tabela 11, o dicionário de dados da entidade *Bandeira*, que armazena os dados das bandeiras dos postos de combustíveis, auxiliando tanto na identificação quanto no boicote por bandeira.

Tabela 11: Dicionário de dados da entidade *Bandeira*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador da bandeira
Nome	Texto	Nome da bandeira
Caminho da Imagem	Texto	Caminho da imagem da bandeira

Por fim, a Tabela 12 mostra o dicionário de dados da entidade *BoicoteBandeira* que armazena os dados dos boicotes por bandeira.

Tabela 12: Dicionário de dados da entidade *BoicoteBandeira*

Atributo	**Domínio**	**Descrição**
Id	Numérico	Número identificador do boicote por bandeira
Data de Início	Data	Data de início da lista de boicote
Data de Término	Data	Data final da lista de boicote
Bandeira Id	Numérico	Chave estrangeira da tabela *Bandeira*

APÊNDICE B – Telas da Aplicação

Para acessar a aplicação, além de utilizar contas no Facebook ou Google, é possível criar uma conta diretamente na aplicação, ilustrado na Figura 27

Figura 27: Tela de criação de conta. Fonte: autores

A aplicação pode mostrar os detalhes de um posto de combusível específico, assim como iniciar uma rota para chegar nele, como mostra a Figura 28

Figura 28: Tela de detalhes do posto de combustível na rota. Fonte: autores

É possível também inserir um endereço para o aplicativo planejar uma rota e mostrar todos os postos de combustíveis perto daquela rota, como ilustra a Figura 29.

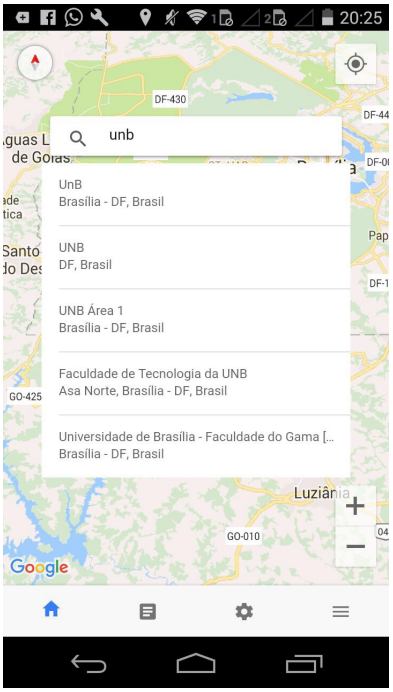

Figura 29: Buscando localizações. Fonte: autores

APÊNDICE C – Testes no Ionic 2

O teste unitário relizado em serviços precisa de um *mock*, que é a simulação de algum objeto para testes, para o *backend*, uma vez que o objetivo é testar a unidade separadamente. O código abaixo é um exemplo de teste unitário de um serviço simulando a comunicação com a API, para garantir que o serviço funciona por si só. Entre as linhas 31 e 43, é criado um usuário fictício e quando o serviço é chamado para procurar um usuário pelo email, na linha 51, o usuário retornado tem que ter o nome e e sobrenome definido no *mock*, como requisitado nas linhas 54 e 55.

```
1  import { TestBed, inject, async } from '@angular/core/testing';
2  import { Http, HttpModule, BaseRequestOptions, Response, ResponseOptions
       } from '@angular/http';
3  import { MockBackend } from '@angular/http/testing';
4
5  import { User } from '../../../models/user';
6  import { UserService } from '../../../providers/user-service';
7
8  describe('Unit test: Providers: UserService', () => {
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [],
12       providers: [
13         UserService,
14         MockBackend,
15         BaseRequestOptions,
16         {
17           provide: Http,
18           useFactory: (mockBackend, options) => {
19             return new Http(mockBackend, options);
20           },
21           deps: [MockBackend, BaseRequestOptions]
22         }
23       ],
24     imports: [ HttpModule ]
25     }).compileComponents();
26   }));
27
28   describe('getUser()', () => {
29     it('should get an user', inject([UserService, MockBackend], (
         userService, mockBackend) => {
30           let user = new User();
31           const mockResponse = {
32             "id": 1,
```

```
33              "email": "apptest@apptest.com",
34              "created_at": "2017-04-19T01:06:15.145Z",
35              "updated_at": "2017-04-19T01:06:15.145Z",
36              "name": "App",
37              "surname": "Teste",
38              "document_number": null,
39              "phone": null,
40              "uid": null,
41              "oauth_token": null,
42              "provider": null
43            }
44
45            mockBackend.connections.subscribe((connection) => {
46                connection.mockRespond(new Response(new ResponseOptions({
47                    body: mockResponse
48                })));
49            });
50
51            userService.getUser('apptest@apptest.com')
52            .then(response => {
53              user = response;
54              expect(user.name).toBe('App');
55              expect(user.surname).toBe('Teste');
56            })
57        }));
58    });
59  });
```

Foi realizado o mesmo teste no serviço de usuários porém agora integrado com a API sem a criação de um *mock* para ela. O código abaixo mostra que antes de cada teste ser relizado um usuário é criado na API, como mostrado nas linhas 19 à 28, e depois de cada teste ele é deletado nas linhas 30 à 32. Ao pedir para a API retornar o usuário pelo email, na linha 37, o nome e o sobrenome tem que ser os mesmos do usuário criado antes do teste acontecer, como requisitado nas linhas 40 e 41.

```
1  import {HttpModule} from '@angular/http';
2  import {async, TestBed,getTestBed } from '@angular/core/testing';
3
4  import { User } from '../../../models/user';
5  import { UserService } from '../../../providers/user-service';
6
7  let userService: UserService;
8  let user : User;
9
10 describe('Integration test: Providers: UserService', () => {
11   beforeEach(async(() => {
12    TestBed.configureTestingModule({
```

```
13      imports: [HttpModule],
14      providers:[UserService]
15    })
16    .compileComponents();
17  }));
18
19  beforeEach(async(() => {
20    user = new User();
21    user.name = "App";
22    user.surname = "Teste";
23    user.email = 'apptest@apptest.com';
24    user.password = '12345678';
25    user.provider = 'email';
26    userService = getTestBed().get(UserService);
27    userService.create(user);
28  }));
29
30  afterEach(async(() => {
31    userService.delete(user);
32  }));
33
34  describe('getUser()', () => {
35    it('should get User', async(() => {
36      let user = new User();
37      userService.getUser('apptest@apptest.com')
38      .then(response => {
39        user = response;
40        expect(user.name).toBe('App');
41        expect(user.surname).toBe('Teste');
42      })
43    }));
44  });
45});
```

APÊNDICE D – Protótipos

A Figura 30 mostra o protótipo de papel da tela de quando um usuário chegar à um posto de combustíveis, onde terá informações sobre o mesmo.

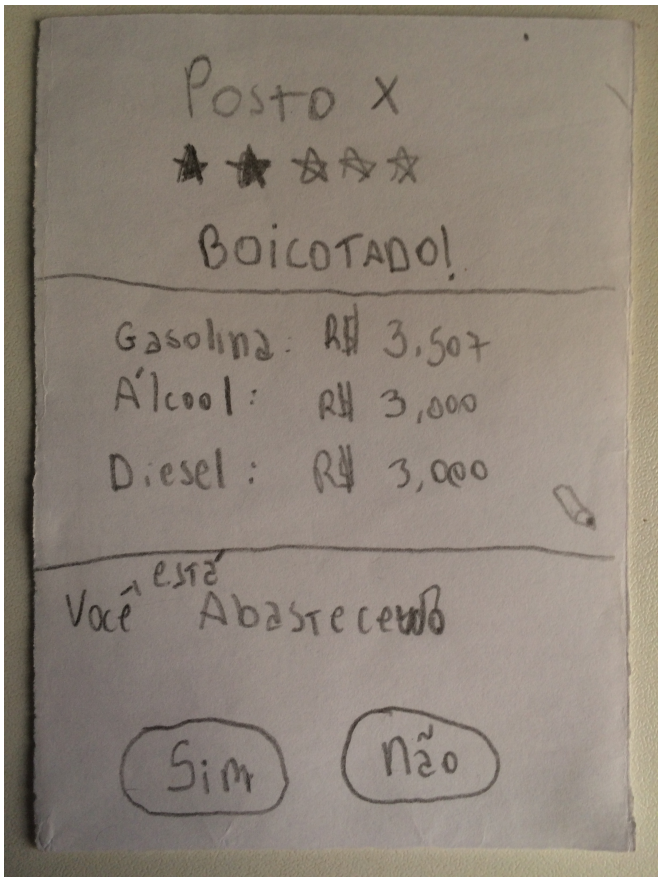

Figura 30: Protótipo de papel da tela de informações do posto. Fonte: autores

A Figura 31 mostra o protótipo de papel de como será feito a avaliação dos preços dos combustíveis do posto que o usuário está localizado.

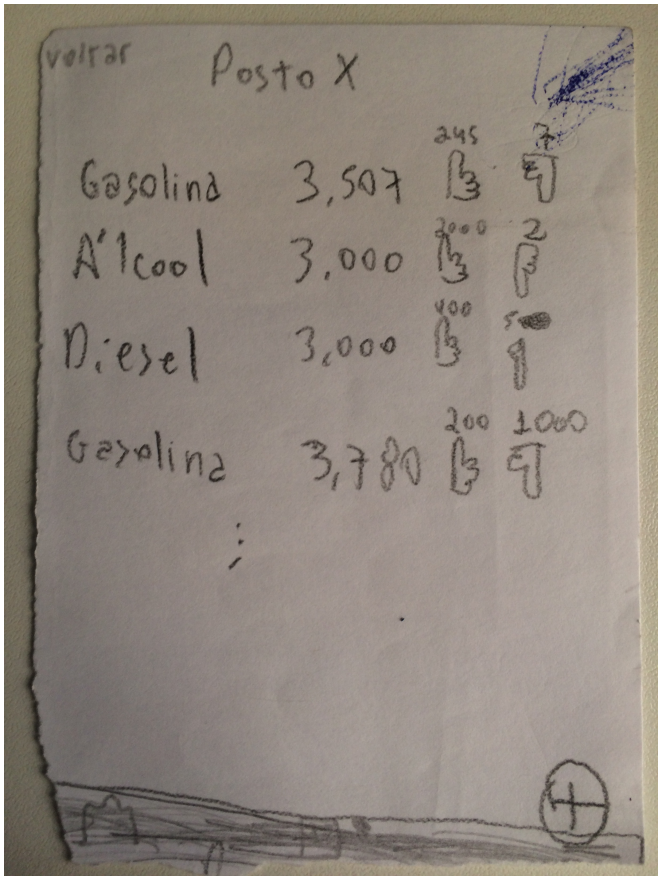

Figura 31: Protótipo de papel da tela de avaliar preço dos combustíveis. Fonte: autores

A Figura 32 mostra o protótipo de papel da tela onde o usuário pode avaliar, após abastecer, um posto de combustíveis de 0 a 5 estrelas.

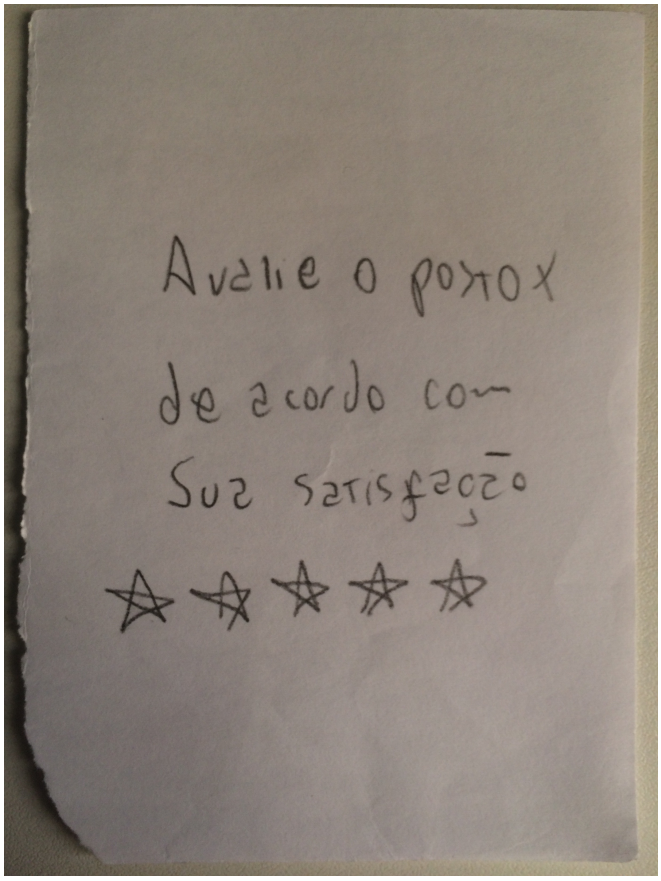

Figura 32: Protótipo de papel da tela de avaliar um posto de combustíveis. Fonte: autores

A Figura 33 mostra o protótipo de papel da tela com uma lista de postos boicotados.

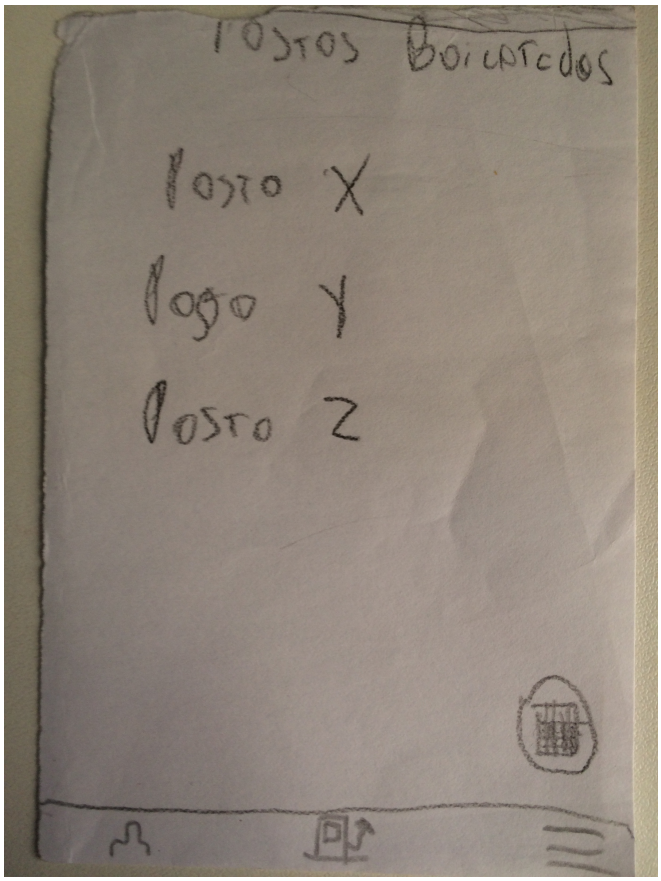

Figura 33: Protótipo de papel da tela de postos boicotados. Fonte: autores

A Figura 34 mostra o protótipo de papel da lista de postos de combustíveis. Que a priori pode ser ordenada por preço e distância.

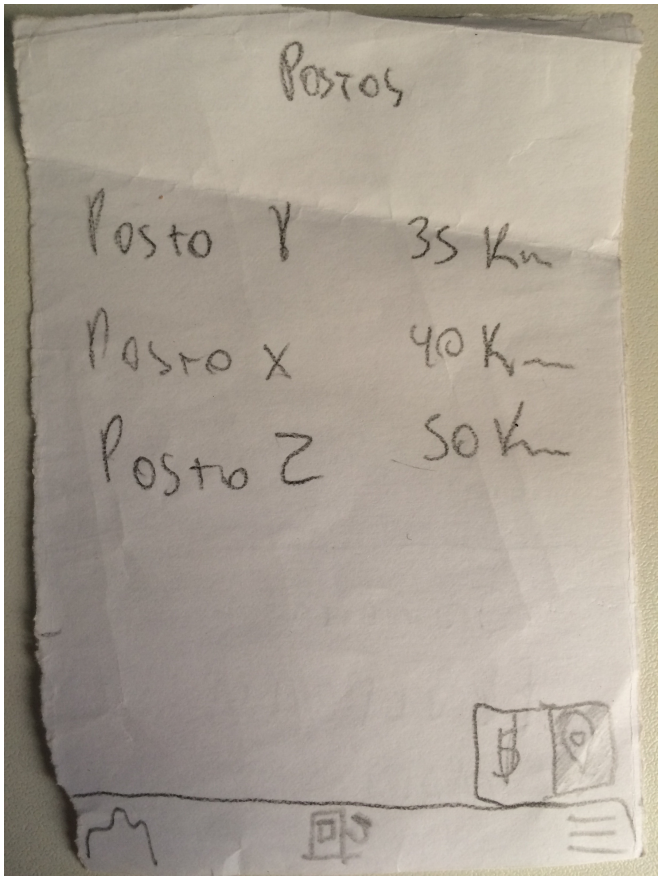

Figura 34: Protótipo de papel da tela de postos de combustíveis. Fonte: autores

A Figura 35 mostra as 8 telas sendo utilizadas no aplicativo Marvel, para a simular a interação entre telas.

Figura 35: Protótipos de papel utilizados no aplicativo Marvel. Fonte: autores

A Figura 36 mostra o protótipo da tela de login da aplicação. O login pode ser feito via Facebook, Google ou criando uma conta local.

Figura 36: Protótipo da tela de login. Fonte: autores

A Figura 37 mostra o protótipo da tela de cadastro da aplicação.

Figura 37: Protótipo da tela de cadastro. Fonte: autores

A Figura 38 mostra o protótipo da tela inicial da aplicação, que mostra o mapa para usuário navegar e visualisar postos de combustíveis.

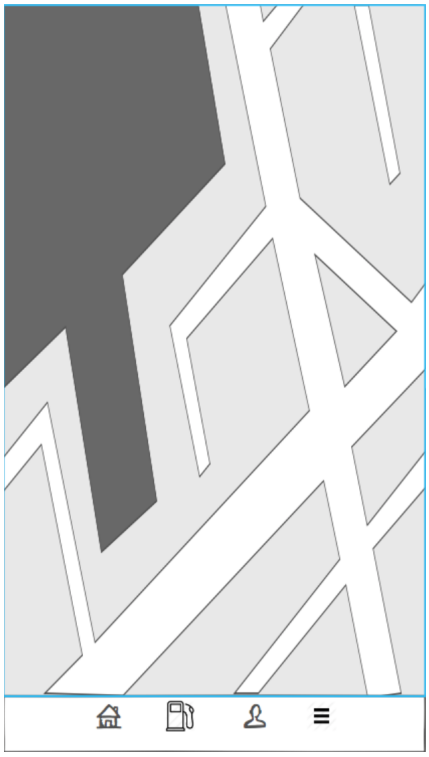

Figura 38: Protótipo da tela inicial. Fonte: autores