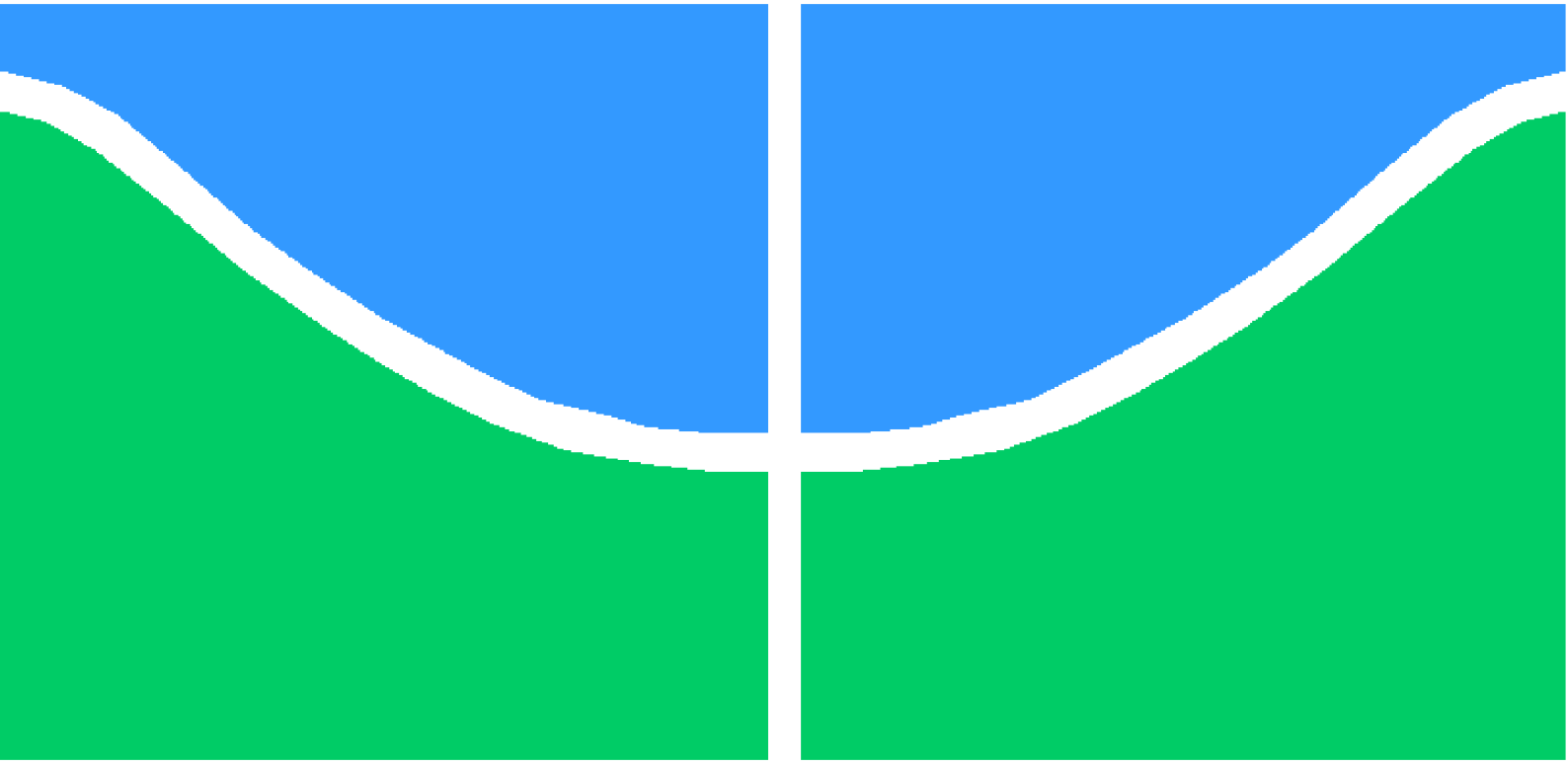

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

NotifiCar: Aplicativo para Monitoramento da Situação dos Automóveis de uma Cidade

Autor: Jonathan Nogueira Rufino Batista Paiva
e Luis Henrique Nunes Guimarães

Orientador: Profº. Dr.André Luiz Peron Martins Lanna

Brasília, DF

2018

Jonathan Nogueira Rufino Batista Paiva
e Luis Henrique Nunes Guimarães

NotifiCar: Aplicativo para Monitoramento da Situação dos Automóveis de uma Cidade

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: Profº. Dr.André Luiz Peron Martins Lanna

Brasília, DF

2018

Jonathan Nogueira Rufino Batista Paiva
e Luis Henrique Nunes Guimarães
NotifiCar: Aplicativo para Monitoramento da Situação dos Automóveis de uma Cidade/ Jonathan Nogueira Rufino Batista Paiva
e Luis Henrique Nunes Guimarães. – Brasília, DF, 2018-
64 p.

Orientador: Profº. Dr.André Luiz Peron Martins Lanna

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2018.

1. Desenvolvimento Mobile 2. React Native. 3. Android 4. iOS 5. Profº. Dr.André Luiz Peron Martins Lanna. 6. Universidade de Brasília. 7. Faculdade UnB Gama. 8. NotifiCar. 9. NotifiCar: Aplicativo para Monitoramento da Situação dos Automóveis de uma Cidade.

Jonathan Nogueira Rufino Batista Paiva
e Luis Henrique Nunes Guimarães

NotifiCar: Aplicativo para Monitoramento da Situação dos Automóveis de uma Cidade

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, 06 de Julho de 2018

Profº. Dr.André Luiz Peron Martins Lanna
Orientador

Profª. Msc.Cristiane Soares Ramos
Convidada

Brasília, DF
2018

Sem esforços e sem sacrifícios, não há vitórias. À nossa orientadora somos gratos por ter almejado o objetivo que tanto sonhamos conquistar. Foi uma montanha a escalar, um deserto atravessar, mas cheguamos no topo com ajuda de todos e daqui ninguém irá nos tirar.

Agradecimentos

Eu, Luis Henrique Nunes Guimarães, agradeço a minha mãe Janice, meu Pai Cícero por não medirem esforços para que eu pudesse levar meus estudos adiante, agradeço meus familiares por sempre me apoiarem e ajudarem em todas as minhas conquistas. Agradeço aos amigos que fiz durante a jornada na faculdade pela motivação e incentivos. Agradeço aos professores pelo ensino recebido durante a faculdade e em especial aos professores, Edna Dias, George Marsicano e Sérgio Freitas pelas orientações e oportunidades que foram dadas a mim. Por fim, agradeço a Universidade de Brasília por ter me dado a oportunidade de realizar este curso.

Eu, Jonathan Nogueira Rufino Batista Paiva, agradeço primeiramente aos meus pais por sempre me apoiarem e não me deixarem desistir nos momentos difíceis dessa longa jornada que é a universidade. Agradeço também a minha namorada por sempre estar ao meu lado, por apoiar as minhas decisões, e por ser compreensiva nos momentos difíceis. Agradeço aos meus colegas de curso e projetos pelas experiências que compartilhamos, seja nas disciplinas que cursamos juntos, como em nosso projeto na Fábrica de Software. Por fim, agradeço aos meus professores George Marsicano, Sérgio Freitas e Edna Dias pelas oportunidades que me foram dadas, os ensinamentos e orientações, os quais me permitiram evoluir pessoalmente e profissionalmente.

“Todo progresso acontece fora da zona de conforto.“
(Michael John Bobak)

Resumo

Monitorar a situação de veículos em uma cidade pode ser uma tarefa árdua dada a necessidade de utilização de sensores ou processamento de imagens por câmeras. No entanto, pode-se substituir a utilização de sensores digitais/eletrônicos por indivíduos que fornecem as informações necessárias. Portanto, este trabalho de conclusão de curso tem como objetivo desenvolver uma aplicação mobile multiplataforma, utilizando o framework React Native, este framework utiliza uma única linguagem de programação para geração de um aplicativo para os sistemas operacionais iOS e Android. Essa aplicação, denominada NotifiCar, tem como objetivo principal notificar os donos dos veículos que se encontram em alguma situação adversa por meio de outras pessoas que identificaram essa adversidade.

Palavras-chaves: Desenvolvimento Mobile; React Native; Android; iOS; NotifiCar.

Abstract

Monitoring the situation of vehicles in a city can be an arduous task given the need to use sensors or image processing by cameras. However, it is possible to replace the use of digital/electronic sensors with individuals who provide the necessary information. Therefore, this course completion work aims to develop a multiplatform mobile application, using the framework React Native, this framework uses a single programming language to generate an application for the iOS and Android operating systems. This application, called NotifiCar, has as main objective to notify the owners of the vehicles that are in some adverse situation through other people who have identified this adversity.

Keywords: Mobile Development; React Native; Android; iOS; NotifiCar.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

TCC	Trabalho de Conclusão de Curso
UnB-FGA	Universidade de Brasília - Campus Gama
PO	Product Owner
PB	Product Backlog
IC	Integração Contínua
CI	*Continuous Integration*
CD	*Continuous Delivery*
GCS	Gerência de Configuração de Software
TICs	Tecnologias da Informação e Comunicação
IDE	Ambiente de Desenvolvimento Integrado

Sumário

1 Introdução

No Brasil cerca de 43,9% (IBGE, 2016) da população utiliza veículo para locomoção, seja por motivos profissionais como ir ao trabalho ou à uma reunião de negócios, por necessidade pessoal como ir ao mercado comprar alimentos ou também à lazer, por exemplo, para ir à uma festa.

Chegando ao seu destino, o motorista costuma realizar algumas verificações antes de deixar o veículo. As verificações mais comuns costumam ser o fechamento das janelas, o desligamento dos faróis e o trancamento do carro. Contudo, alguns fatores podem colaborar para que o motorista não realize essas ações ou até mesmo que algo passe despercebido, podendo deixar o veículo vulnerável à assaltos ou causar danos ao próprio veículo, como por exemplo, manter o farol aceso o que pode fazer com que a bateria descarregue.

Além disso, veículos estacionados em locais públicos podem atrapalhar outros motoristas, pedestres e estabelecimentos. Mesmo que estacionado de maneira correta e em local adequado, um veículo muito grande pode precisar de mais espaço para trafegar, ficando assim impossibilitado.

Atualmente, não existe uma solução efetiva para encontrar ou entrar em contato com os responsáveis pelos veículos que nos deparamos na rua, onde seria necessário procurar no local pelo possível dono. No entanto, essa tarefa pode ser inviável, devido ao local ou circunstâncias em questão.

Utilizar uma aplicação que possibilite o registro da situação percebida utilizando apenas a identificação da placa, sem a necessidade de se comunicar com outras pessoas, pode se mostrar uma solução adequada para este problema.

Assim, percebe-se a necessidade em comunicar ao responsável pelo veículo que se encontra em uma situação adversa, para que dessa forma, a situação possa ser resolvida o mais rápido possível, além de evitar a geração de prejuízo para outras pessoas ou entidades.

Este trabalho propõe o desenvolvimento de uma aplicação *mobile*, que permita a comunicação de forma fácil e rápida de qualquer eventualidade associada à um veículo identificada por outros indivíduos. Fazendo uso da tecnologia *React Native*, uma tecnologia emergente para desenvolvimento de aplicações *mobile*, a qual usa uma única linguagem de programação, *JavaScript*, e possibilita a criação de aplicativos para as plataformas *Android* e *iOS*, permitindo dessa forma o maior compartilhamento de código fonte.

1.1 Justificativa

Atualmente, a população faz o uso de redes sociais como *Facebook* para encontrar as pessoas que são donas dos veículos, o que torna a busca pouco eficiente, se pararmos para pensar, muitas pessoas possuem carros parecidos, fazendo com que as pessoas nessas redes sociais especulem o possível dono, além disso, algumas pessoas não sabem a placa do seu próprio veículo.

Ainda mais, existem situações em que uma pessoa pode deixar seu veículo estacionado em um local e trabalhar ou estudar em outro, a menos que o veículo seja conhecido pelas pessoas locais, se torna ainda mais difícil encontrar o dono para informá-lo da situação.

Dentre as mais variadas situações adversas que um desses veículos pode se encontrar, algumas podem acarretar prejuízo à terceiros ou ao próprio dono, como por exemplo veículos com:

- Farol aceso ou Alarme disparado (descarregamento da bateria);
- Vidro aberto (furtos);
- Impedindo a locomoção de outros veículos ou pedestres (multas ou algum tipo de retaliação);
- Pneus furados (danos ao próprio pneu).

Devido a essas dificuldades apontadas, o **NotifiCar**, aplicativo produto deste trabalho, vem com o propósito de facilitar a comunicação entre o dono de um veículo e a pessoa que deseja alertá-lo de algum evento que esteja ocorrendo com o veículo em questão.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é desenvolver uma aplicação *mobile* que facilite a comunicação pelas pessoas à respeito de veículos com algum tipo de problema estacionados em locais públicos, ou mesmo privados.

1.2.2 Objetivos Específicos

- Investigar as práticas recomendadas relacionadas à *User Experience - UX*, para implementar uma boa usabilidade e boa visualização dos dados apresentados;

- Utilizar um *framework* de desenvolvimento híbrido que possibilita a geração de aplicativos nativos para as plataformas *iOS* e *Android*;
- Documentar a solução proposta, gerando os artefatos de acordo com as boas práticas da Engenharia de *Software* em diferentes níveis de abstração: requisitos, projeto, codificação e teste;
- Implementar um processo de entrega contínua para a solução;
- Implementar a solução proposta;
- Implantar e testar a solução proposta em um ambiente real.

1.3 Metodologia de Pesquisa

A metodologia de pesquisa deste trabalho é exploratória. Estas pesquisas têm como objetivo proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a constituir hipóteses. Pode-se dizer que estas pesquisas têm como objetivo principal o aprimoramento de ideias ou a descoberta de intuições (GIL, 2002). Na maioria dos casos, essas pesquisas envolvem: (a) levantamento bibliográfico; (b) entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; e (c) análise de exemplos que "estimulem a compreensão"(GIL, 2002). Embora a pesquisa exploratória seja bastante flexível, na maioria dos casos assume a forma de pesquisa bibliográfica ou de estudo de caso (GIL, 2002).

1.4 Fases do Trabalho

Como o TCC, no curso de Engenharia de Software da UnB-FGA é dividido em duas etapas, este trabalho foi separado de acordo com essas fases. A primeira fase, que é representada pelo TCC 1, consiste na definição e análise do problema, na definição do escopo, na análise de referencial teórico sobre os conceitos que fundamentam este trabalho, na revisão bibliográfica, na escolha da tecnologia para desenvolver a solução, e ainda, na definição dos requisitos do *software* e a metodologia que será utilizada para o desenvolvimento do mesmo. Já a segunda fase, TCC 2, é a fase de desenvolvimento da solução validada. Nessa fase será desenvolvido o *software* idealizado, utilizando as práticas definidas, como também seguindo o planejamento já realizado. A Figura 1 apresenta o processo descrito anteriormente.

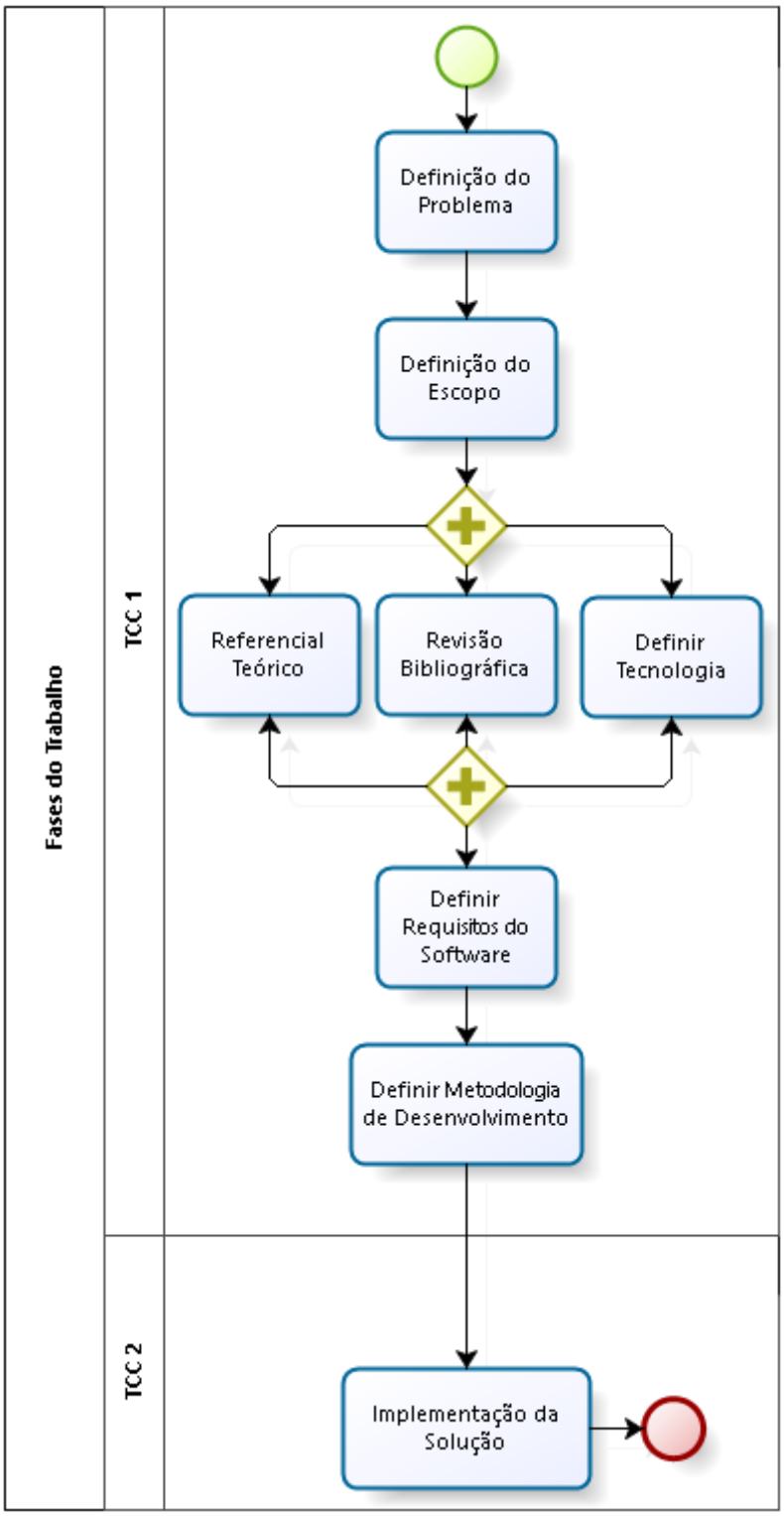

Figura 1 – Fases do Trabalho. Fonte: Autores

1.5 Organização do Trabalho

O presente trabalho está organizado em cinco capítulos, sendo este o capítulo introdutório, que discorre sobre o contexto, a descrição do problema, os objetivos do trabalho e sua organização.

- **Capítulo 2 - Referencial Teórico:** Neste capítulo são apresentados os conceitos teóricos que fundamentam este trabalho, explicitando as informações obtidas por meio da pesquisa bibliográfica realizada.
- **Capítulo 3 - Metodologia:** Neste capítulo são descritos os procedimentos e práticas adotados para a realização deste trabalho.
- **Capítulo 4 - Desenvolvimento:** Neste capítulo são apresentadas as ferramentas que darão suporte as atividades de desenvolvimento de *software*, gestão da equipe e documentação da aplicação.
- **Capítulo 5 - Considerações Finais:** Neste capítulo são apresentadas as considerações sobre o estudo apresentado e os trabalhos futuros a serem realizados.

2 Referencial Teórico

Neste Capítulo, é apresentado o referencial teórico que fundamenta os conceitos, práticas e técnicas utilizadas no desenvolvimento do presente trabalho.

O Capítulo está organizado em Seções. A Seção 2.1 apresenta os conceitos fundamentais que constituem a Engenharia de *Software*. Na Seção 2.2 são apresentadas as abordagens e conceitos referentes às aplicações *mobile*. A Seção 2.3 aborda a ideia dos princípios ágeis. Na Seção 2.4 são apresentados os aspectos relacionados à garantia da qualidade de *software*. Por fim, a Seção 2.5 trata dos conceitos envolvidos no processo de entrega contínua de *software*.

2.1 Engenharia de *Software*

Engenharia de *Software* é o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um *software* que seja confiável e que funcione eficientemente em máquinas reais (BAUER, 1972), com a aplicação sistemática de conhecimentos científicos, tecnológicos, métodos e experiência para o projeto, implementação, teste e documentação de *software* (ISO/IEC/IEEE, 2010).

Segundo (PRESSMAN, 2011), a base da engenharia de software é a camada de processos. O processo de engenharia de *software* é a liga que mantém as camadas tecnológicas coesas e possibilita o desenvolvimento de *software* de forma racional e dentro do prazo. O processo define a metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de *software*. O processo de *software* constitui a base para o controle do gerenciamento de projetos de *software* e estabelece o contexto no qual são aplicados métodos técnicos, a qualidade é garantida e mudanças são geridas de forma apropriada.

2.2 Aplicações *Mobile*

Com os *smartphones* sendo o principal dispositivo de mão para mais de três bilhões de usuários (LTD, 2017), os aplicativos móveis já se tornaram uma necessidade tanto nos campos técnico quanto comercial.

No entanto, existem muitas diferenças no desenvolvimento de aplicações *mobile* em relação ao desenvolvimento de aplicações tradicionais *desktop* ou *web*, devido as diversas limitações de *hardware* (capacidade de processamento, armazenamento, ambiente de utilização, tamanho da tela etc) e também características específicas como sensores,

câmera, bateria etc (RAHIMIAN; RAMSIN, 2008).

Por outro lado, existem várias abordagens para o desenvolvimento de aplicativos móveis, mas dada a rápida velocidade da evolução do *software* móvel é crucial entender as tecnologias básicas (SERRANO; HERNANTES; GALLARDO, 2013). Na Tabela 1 são apresentados os tipos de Sistemas Operacionais - SO móvel e as suas respectivas linguagens para desenvolvimento de aplicações (CHARLAND; LEROUX, 2011).

Tabela 1 – SO *mobile* e suas linguagens para desenvolvimento de aplicações. Adaptado de (SERRANO; HERNANTES; GALLARDO, 2013)

Tipo de SO	**Linguagem**
Apple iOS	C, Objective C e Swift
Google Android	Java (Harmony flavored, Dalvik VM), Kotlin
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Window 7 Phone	.NET
HP Palm web	OS HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

Como pode ser visto na Tabela 1 há uma grande diversidades de sistemas móveis juntamente com diferentes linguagens para desenvolver as respectivas aplicações. Com isso, percebe-se a necessidade de uma vasta gama de conhecimento tecnológico para que seja possível o desenvolvimento para tais plataformas.

Apesar dos diversos sistemas operacionais *mobile* existentes, atualmente apenas três deles detêm a grande parcela do consumo mundial conforme apresentado na Figura 2.

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Figura 2 – Distribuição da porcentagem de sistemas *mobile* dentre as principais plataformas. Fonte: (IDC, 2017)

Como mostra a Figura 2, o sistema *Android* detêm uma parcela de mais de 80% desde o primeiro quadrimestre de 2016 e que vem se mantendo até o primeiro quadrimestre de 2017. Já a plataforma *iOS* vem em segundo lugar com uma média de 15%. Com uma parcela bem pequena temos o *Windows Phone* que vem decrescendo constantemente.

De acordo um levantamento publicado por (APPANNIE, 2016) a *Play Store* teve em uma média de downloads 100% maior que a *App Store*, apesar disso a receita para o mesmo período foi maior em 100% na loja da plataforma *iOS*. Com isso, pode-se perceber porque apesar da baixa parcela em número de dispositivos as plicações desenvolvidas para *iOS* ainda são altamente viáveis devido à sua rentabilidade.

2.2.1 Desenvolvimento Nativo

Aplicações nativas são específicas de sistema operacional, segundo (SERRANO; HERNANTES; GALLARDO, 2013), os sistemas operacionais móveis desejam aplicativos específicos para que seus próprios ambientes possam aproveitar ao máximo seus recursos particulares.

O desenvolvimento de aplicações *iOS* é realizando por meio da *SDK iOS*, uma *api* que fornece acesso aos recursos dos dispositivos. Além disso, o *XCode* é a *IDE* oficial fornecida pela *Apple* para manipulação de código fonte, que tem como linguagens suportadas o *ObjectiveC* e *Swift* (APPLE INC., 2017c). Vale ressaltar que aplicativos *iOS* só podem ser desenvolvidos em máquinas com o sistema operacional *MacOS*. Os aplicativos são disponibilizados na loja virtual *App Store*, e requer uma licença do *Apple Developer Program* para publicação dos apps. A licença pode ser obtida por valor anual de US$99 (APPLE INC., 2017a).

Já as aplicações *Android* são desenvolvidos utilizando a *Android SDK*, que semelhante à *SDK iOS* fornece acesso aos recursos dos dispositivos. A *IDE* oficial de desenvolvimento é o *Android Studio* e as linguagens suportadas são *Java* e *Kotlin* (GOOGLE INC., 2017c). Os aplicativos são disponibilizados oficialmente na *Play Store*. Para a publicação dos *apps* é necessário uma conta de desenvolver que pode ser obtida pelo valor único de US$25 (GOOGLE INC., 2017a).

2.2.2 Desenvolvimento Híbrido

Aplicativos híbridos são desenvolvidos utilizando ferramentas de desenvolvimento *web* em conjunto com elementos nativos (SERRANO; HERNANTES; GALLARDO, 2013). Esses aplicativos são criados com o uso de *HTML*, *CSS* e *JavaScript* e ainda podem utilizar características e sensores nativos do aparelho pela *JS API* (CHARLAND; LEROUX, 2011).

Todo esse código roda dentro de algo chamado *webview*. *Webview* é o nome dado

a um tipo especial de *browser* que começa a rodar assim que a app híbrida é aberta pelo usuário. É dentro desse browser que o *app* é executado. A *webview* contém apenas o necessário para que o *html*, *css* e *javascript* funcionem (LUIS VASCONCELLOS, 2017).

Ela se comporta como a *engine* de renderização do *app*. Ela também consegue acessar recursos nativos do dispositivo como a câmera, microfone, acelerômetro, etc. Isso é possível, graças a uma interface *javascript* que torna a *webview* apta a executar código nativo nos dispositivos. Ou seja, no desenvolvimento híbrido é possível usar apenas *javascript* para acessar os recursos nativos do dispostivo, coisa que nenhum *browser* comum instalado no aparelho seria capaz de acessar (LUIS VASCONCELLOS, 2017).

Logo, a principal vantagem de um *app* híbrido é sem dúvida ter uma única base de código capaz de rodar em diversos sistemas diferentes. Ao invés de dar manuntenção para *Android*, *iOS* ou *Windows Phone* por exemplo, em um *app* híbrido só é necessário dar manutenção em única base de código que, por sua vez, poderá ser empacotada e distribuída para todas as plataformas disponíveis (LUIS VASCONCELLOS, 2017).

2.3 Desenvolvimento Ágil de *Software*

O desenvolvimento ágil de *software* é uma abordagem que surgiu formalmente após a publicação do Manifesto Ágil por um grupo de especialistas da área. Os métodos ágeis focam os esforços na entrega contínua de *software* em curtos períodos de tempo, removendo a necessidade de processos pesados e vasta documentação. Além disso, tem como uma das principais características a alta capacidade de lidar com mudanças devido ao seu princípio de "responder à mudanças em vez de seguir um plano estritamente". Como esta capacidade pode ser ativada através de um desenvolvimento iterativo e incremental. Emergiram novas práticas ágeis como *Extreme Programming - XP*, *Scrum* e *KanBan*. Esses métodos ágeis abrangem práticas para aumentar a capacidade de entrega no desenvolvimento de *software* (JENTSCH, 2017).

2.3.1 *Scrum*

O *Scrum* é uma metodologia ágil para gestão e planejamento de projetos de *software*, sejam eles, *web* ou *mobile*. Ele basicamente abrange cinco eventos: *sprint*, *sprint planning*, *daily scrum*, *sprint review* e retrospectiva; três artefatos: *product backlog*, *sprint backlog*, *increment*; e três papéis: *product owner*, *development team*, *scrum master* (JENTSCH, 2017). A Figura 3 detalha o processo do *Scrum*.

O funcionamento do *Scrum* acontece da seguinte forma: Como metodologias ágeis são iterativas, o projeto é dividido em *Sprints*, que representa o conjunto de atividades que deve ser feitas durante um período de tempo. Todas as atividades que devem ser desenvolvidas são mantidas no *PB*. No ínicio de cada *Sprint* é feito uma reunião denomi-

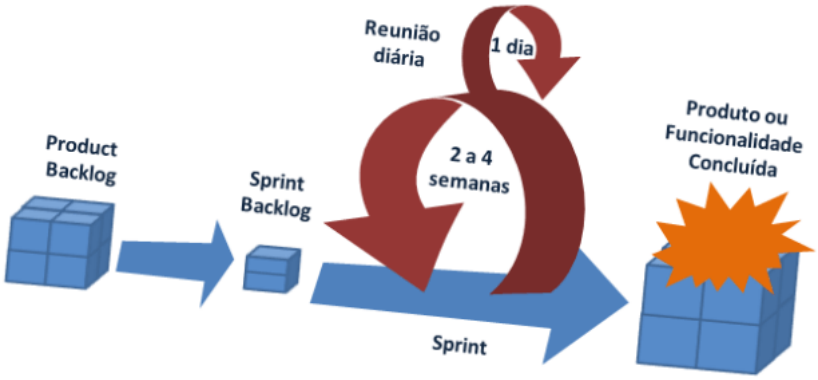

Figura 3 – Processo Scrum. Fonte: (MINDMASTER, 2014)

nada de *Sprint Planning Meeting*, ao qual o *PO* determina quais os itens que ele deseja priorizar do *PB* e toda a equipe seleciona as atividades que irá realizar durante a *Sprint* para satisfazer tais itens. Os itens selecionados vão para o *Sprint Backlog*. A cada dia de trabalho, a equipe faz uma breve reunião chamada de *Daily Scrum*, onde o principal objetivo é dispersar o conhecimento do que foi realizado no dia anterior, o que será feito e verificar se há algo que estar impedindo a continuação do trabalho. Ao fim da *Sprint*, a equipe mostra o que foi desenvolvido na reunião denominada *Sprint Review Metting*, e logo após é feito a última reunião da *Sprint*, chamada *Sprint Retrospective*, onde a equipe fala pontos positivos, negativos, melhorias e planeja a próxima *Sprint*, onde todo o ciclo reinicia-se (SCRUM..., 2014).

A descrição detalhada dos termos apresentados acima estão na Tabela 2.

2.4 Qualidade de *Software*

Como descrito por (KAN, 2014) a qualidade de *software* está associada à falta de *bugs* presentes no produto. Do mesmo modo, também é caracterizado como qualidade a conformidade entre os requisitos e as funcionalidades presentes na aplicação, visto que um *software* que contém muitas falhas funcionais não cumpre o requisito básico de prover as funcionalidades desejadas.

Além disso, a qualidade pode estar relacionada as diversas áreas existentes em um projeto de *software* como Qualidade de Requisitos; Qualidade de Implementação; Qualidade de Design entre outros (KAN, 2014).

Para que se possa garantir a qualidade esperada de um produto, é necessário definir anteriormente que tipo de qualidade se pretende alcançar, para que assim possam ser definidos os processos, práticas, técnicas e ferramentas que vão auxiliar na obtenção do objetivo esperado.

Tabela 2 – Práticas do Scrum

Práticas do *Scrum*	**Descrição**
Sprint	*Sprint* representa o tempo que um conjunto de atividades deve ser executado.
Sprint planning	Reunião de planejamento na qual o *product owner* prioriza os itens do *product backlog* e a equipe seleciona as atividades que ela será capaz de implementar durante o *sprint* que se inicia.
Daily scrum	Breve reunião que a equipe faz a cada dia da *sprint*.
Sprint review	Reunião que acontece no final de uma *sprint*, onde a equipe apresenta as funcionalidades implementadas.
Retrospectiva	Ocorre ao final depois da *Sprint review* e serve para identificar o que funcionou bem, o que pode ser melhorado e que ações serão tomadas para melhorar as próximas *sprints*.
Product backlog	Lista contendo todas as funcionalidades que serão implementadas durante o projeto.
Sprint backlog	Lista de funcionalidades do *product backlog* que serão implementadas durante a *Sprint*.
Increment	É a soma de todas as funcionalidades do *product backlog* que foram concluídas durante as *sprints*.
Product owner	É o papel que uma pessoa se torna o responsável por definir os itens que compõem o *Product Backlog* e os priorizar nas *sprint Planning Meetings*.
Development team	Equipe de desenvolvimento.
Scrum master	É o papel que uma pessoa se torna o responsável pela *sprint*, logo a pessoa deve verificar o *development team*, retirar empecilhos e manter a ordem durante a *sprint* garantindo a entrega das funcionalidades listadas no *sprint backlog*.

Tendo em vista que o presente trabalho busca garantir a qualidade voltada à implementação, as subseções seguintes são necessários para o entendimento dos conceitos abordados.

2.4.1 Testes de *Software*

Segundo (SOMMERVILLE, 2011) teste de *software* é destinado a mostrar que um programa faz o que é proposto a fazer, e para descobrir os defeitos do programa antes do uso. Quando se testa o *software*, o programa é executado usando dados fictícios. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa.

Existem ferramentas que auxiliam na execução e automatizam os testes de maneira rápida e com maior qualidade. Porém, para cada tipo teste aplicado se encontra uma ferramenta diferente. Nas subseções 2.4.1.1 e 2.4.1.2 são apresentados dois tipos de testes,

dentre os existentes.

2.4.1.1 Teste Unitário

Segundo (LING; XU, 2009), teste de unidade é permitir a alta qualidade no desenvolvimento de *software*, nos quais os desenvolvedores projetam, desenvolvem, executam e mantêm o código de teste da unidade para verificar e validar a funcionalidade de cada unidade. Logo, esse teste serve para garantir que cada unidade de seu sistema está realizando o que deveria fazer corretamente, sem respostas indesejadas. As ferramentas que podem se utilizadas aqui são: *NUnit* e *JUnit* (para aplicações *Java*).

2.4.1.2 Teste Funcional

O teste funcional é o processo de avaliação das funções em um sistema para assegurar que atinjam os requisitos especificados para afirmar a qualidade, com métodos que podem ser aplicados de forma econômica e efetiva em sistemas de grande escala e em pequena escala. Essencialmente, o teste funcional pode ser visto como a análise ou execução controlada de um programa para verificar a presença predeterminada de algumas propriedades desejadas. Quando realizado sistematicamente, o teste funcional verifica a ausência de funções errôneas indesejadas (KOBROSLY; VASSILIADIS, 1988). Um exemplo de ferramenta que pode ser utilizada aqui é: *UIautomator* (para aplicações *Android*).

2.4.2 Métricas de *Software*

Métricas de *software* são indicadores pré-estabelecidos que ajudam a monitorar e controlar a qualidade de um *software*. Além disso, contribuem para guiar o planejamento de atividades inerentes ao desenvolvimento (SINGH, 2013).

De acordo com (GAFFNEY JR., 1981), as métricas de *software* auxiliam na análise de dois aspectos, estático e dinâmico, do software. O aspecto estático está voltado ao código desenvolvido, levando em conta as características de manutenibilidade, organização, estruturas utilizadas etc. Já o aspecto dinâmico diz respeito às características funcionais do *software*, como facilidade de uso, e integração com outros sistemas por exemplo.

Para implementar um processo de coleta e análise de métricas é necessário realizar as seguintes atividades (BAKER, 1991):

1. Definir o objetivo de medição;
2. Identificar os atributos que serão medidos;
3. Determinar o objetivo do resultado das medições;
4. Coletar os dados com base nos passos anteriores;

5. Ajustar o processo de medição com base nos resultados caso necessário.

As métricas também são utilizadas para estimar um cronograma e custos de desenvolvimento do *software* além de possibilitar a medição da produtividade e qualidade do produto (MIRANDA, 2013). De acordo com (MILLS, 1988), as boas métricas devem facilitar o desenvolvimento, capazes de relatar os parâmetros do processo ou do produto. Assim, métricas ideais devem ser simples, objetivas, facilmente obtidas, válidas e robustas.

2.5 Entrega Contínua

Dentro do ciclo de desenvolvimento de um produto de *software*, existe a fase de entrega que consiste em disponibilizar a aplicação para uso, seja para usuários ou outros sistemas. Esta fase se resume ao conjunto de práticas que possibilitam que novos incrementos de *software* possam ser integrados ao código já existente (HUMBLE; FARLEY, 2013).

2.5.1 Integração Contínua

A IC é realizada através de compilações de *software* de vários tipos. As compilações de *software* típicas incluem a compilação do código-fonte para obter o programa ou componentes executáveis, compilar o código de teste para obter testes unitários executáveis, executar os testes unitários e testes de aceitação para obter relatórios de teste, realizar análise estática para encontrar problemas no código-fonte e produzir pacote instalável de componentes pré-construídos (CHIN-YUN; CHIEN-TSUN, 2015).

Logo, a IC garante que a cada mudança isolada do serviço, seja ele, código, teste, ambiente de produção, todos serão rapidamente testados e verificados até o ponto que eles serão incluídos no código principal do sistema, sendo assim, o principal objetivo da IC é a rápida identificação de defeitos, correção e relatório (LAVRIV; BUHYL; KLYMASH; GRYNKEVYCH, 2017). Um exemplo de ferramenta de IC para mobile é o *CircleCI*.

2.5.2 *Deploy* Contínuo

Deploy Contínuo é uma abordagem de engenharia de *software* em que as equipes continuam produzindo *software* valioso em curtos período de tempo e garantem que o *software* possa ser liberado de forma confiável a qualquer momento (CHEN, 2015).

Desta forma, o *deploy* contínuo é a questão da automatização da publicação da aplicação, no ambiente de produção no caso de aplicações *web* ou na loja referente ao sistema móvel, no caso de aplicações *mobile*. Porém, para que esta publicação ocorra é necessário que todas as etapas da IC seja satisfeita com sucesso. Assim, como na seção

3.6.1.2 a ferramenta *CircleCI* é um bom exemplo para esta seção, já que ela comporta a funcionalidade de *deploy* contínuo.

A Figura 4 representa um processo simples de *deploy* contínuo de um *software*. Após o desenvolvimento de uma funcionalidade ou alteração de alguma parte do código fonte, é feita a submissão para a ferramenta de controle de versão, que integrada com a ferramenta de *deploy* contínuo realiza a geração da *build*, para garantir que o código continua íntegro, em seguida são executados os testes unitários e funcionais para garantir que o comportamento do *software* não foi alterado. Após as etapas anteriores serem executadas com sucesso o novo incremento de *software* é então disponibilizado em produção (KAUKKANEN et al., 2016).

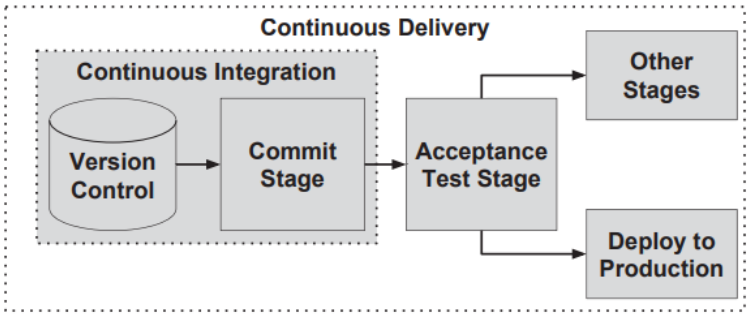

Figura 4 – Processo de Deploy Contínuo. Fonte: (KAUKKANEN et al., 2016)

3 Desenvolvimento

Este Capítulo tem como objetivo descrever os aspectos técnicos e gerenciais referentes ao desenvolvimento do aplicativo NotifiCar.

3.1 Gerenciamento do projeto

Para o desenvolvimento da solução apresentada neste trabalho foi utilizada uma abordagem ágil de desenvolvimento de *software*, combinando o *Scrum* e o *Kanban*. Tendo em vista que a equipe é composta por apenas dois desenvolvedores, alguns ajustes foram necessários para melhor atender às necessidades existentes.

As reuniões diárias ou *daily meetings* foram realizadas de forma remota, já que os autores não se encontram fisicamente no mesmo local e horário. Já as *sprints* tiveram sua duração definida para duas semanas pelo fato deste trabalho ser desenvolvido com uma tecnologia que não é dominada pelos autores, com isso jugou-se necessário um prazo maior para que possa ser realizado o estudo, absorção dos conceitos e a devida implementação das funcionalidades planejadas. Posteriormente, as *sprints* foram replanejadas para uma semana de duração, visto a maior capacidade técnica desenvolvida pelos alunos.

Para auxiliar o acompanhamento do trabalho, o *Kanban* foi utilizado de forma que fosse possível ter uma visão geral da situação do projeto de forma simplificada. Para isso, foram definidos 4 possíveis estados, cada uma representando uma situação diferente para as atividades planejadas. Esses estados são apresentados e detalhados a seguir.

- ***Backlog*** - É o *backlog* do projeto, aqui estão todas as histórias ou outras atividades planejadas para o projeto;
- ***To do*** - contém o *backlog* da *sprint*, são as atividades alocadas para a *sprint* atual;
- ***Doing*** - são as atividades que estão em execução dentro da *sprint* atual;
- ***Checking*** - contém as atividades que foram finalizadas, mas que ainda precisam ser avaliadas e aprovadas por outro desenvolvedor;
- ***Closed*** - por fim, são as atividades que já foram revisadas, concluídas e integradas a *branch* de desenvolvimento, no caso de histórias de usuário ou histórias técnicas.

Como a equipe de desenvolvimento não possuía um local fixo para trabalho, se viu a necessidade de uma ferramenta que pudesse substituir o *kanban* físico. Para isso foi utilizada uma extensão para os navegadores *Google Chrome* e *Mozila Firefox* chamada

Zenhub. O *zenhub* é uma ferramenta de gerenciamento de projetos que fornece uma série de recursos utilizando as informações do repositório no qual se encontra o código. O kanban, por exemplo, é construído com as issues existentes no repositório. Outros recursos existentes são o cálculo de velocity, rastreabilidade de dependências, definição de épicos e features.

3.1.1 *Sprints*

As *sprints* deste projeto serão realizadas com duração de duas semanas cada. Onde cada desenvolvedor será responsável em desenvolver pelo menos uma história por *sprint*, caso o desenvolvedor termine uma história antes da *sprint* terminar, poderá iniciar o desenvolvimento de outra história. Em algum momento do desenvolvimento, alguma história poderá precisar de grande esforço para soluciona-lá, e então os desenvolvedores utilizarão a estratégia do *pair programming* na *sprint*.

As atividades que serão utilizadas em cada sprint deste projeto serão:

- *Sprint planning*
- Retrospectiva
- Reunião semanal
- *Sprint Review*

A explicação de cada atividade pode ser encontrada na Tabela 2.

3.2 Requisitos

Os requisitos do sistema foram documentados em forma de histórias de usuário, como proposto pela metodologia ágil. As Histórias de Usuário, também conhecidas por *User Stories* ou *US*, são artefatos desenvolvidos segundo metodologias ágeis, de forma a definir os requisitos de um sistema.

Elas devem ser curtas, simples e de fácil entendimento. São utilizadas principalmente para fracionar os requisitos, averiguar o esforço para implementar uma determinada funcionalidade e para auxiliar na priorização de quais funcionalidade deve ser implementadas primeiro. Geralmente são escritas em conjunto com os clientes do projeto para negociar a solução, criar critérios de aceitação, retirar dúvidas e tornar o entendimento da solução mais amplo para todos os envolvidos.

Podemos construir uma história especificando um ator, a ação e a funcionalidade desejada, com a seguinte frase:

Como [Ator], desejo [Ação] para [Objetivo]

- **Ator** - Identifica o usuário que realizará a ação no sistema.
- **Ação** – Indica a interação que o usuário irá realizar com o sistema.
- **Objetivo** – Representa o valor agregado que a funcionalidade irá trazer para o usuário.

As histórias de usuários definidas para o desenvolvimento da solução são apresentadas na Tabela 3:

Tabela 3 – Histórias de Usuário

Identificador	**História de Usuário**
US01	Como usuário, desejo cadastrar veículos para receber notificações a seu respeito
US02	Como usuário, desejo registrar ocorrências para alertar o responsável pelo veículo
US03	Como usuário, desejo visualizar o ranking de ocorrências para saber as ocorrências mais frequentes
US04	Como usuário, desejo avaliar a notificação recebida para dar credibilidade ao usuário que a registrou
US05	Como usuário, desejo visualizar o perfil do usuário que me notificou para ver seu nível de confiabilidade
US06	Como usuário, desejo possuir um nível a partir da reputação para determinar a minha confiabilidade
US07	Como usuário, desejo receber medalhas para mostrar aos outros usuários a minha participação no app
US08	Como usuário, desejo visualizar as ocorrências registradas para os meus veículos para saber se eles se encontram em uma situação adversa
US09	Como usuário, desejo verificar se o veículo é roubado para alertar as autoridades
US10	Como usuário, desejo identificar a placa do veículo pela câmera para facilitar o registro da ocorrência
US11	Como usuário, desejo desconectar do aplicativo para deixar de receber notificações
US12	Como usuário, desejo acessar o sistema para utilizar seus recursos

3.3 Ferramentas

Nesta Subseção será explicado o uso de cada ferramenta que foi utilizada no desenvolvimento da aplicação.

- **React Native:** É o *framework* utilizado para o construir a nossa aplicação nativa usando o *React*, que utiliza apenas *JavaScript*. Versão: 0.54.2;
- **API Android:** É a versão mínima requerida para que possa executar o projeto em um aparelho com o sistema operacional *Android*, para que este projeto seja executado, o usuário tem que ter um aparelho com a versão 4.4 ou superior do Android. Versão: 19;
- **API iOS:** É a versão mínima requerida para que possa executar o projeto em um aparelho com o sistema operacional iOS, para este projeto seja executado, o usuário precisa ter um aparelho com a versão 8 ou superior do iOS. Versão: 8.0;
- **VSCode:** Editor de código-fonte utilizado para a visualização e edição dos arquivos de código-fonte da aplicação. Versão: 1.23;
- **Android Studio:** *IDE* utilizada para a aplicação manual das bibliotecas necessárias para o projeto Android, e assim como o *Google Chorme*, também foi utilizado para *Debug*. Versão: 3.1.2;
- **XCode:** *IDE* utilizada para a aplicação manual das bibliotecas necessárias para o projeto iOS e envio da aplicação para *AppStore*. Versão: 9.2;
- **Google Chorme:** Navegador utilizado para efetuar o *Debug* da aplicação, quando necessário. Versão: 65.0.3325.181;
- **Detox:** Ferramenta utilizada para a implementação dos testes de integração do projeto, ele roda em um aparelho real ou emulador, simulando o uso de um usuário real. Versão: 7.3.3;
- **Jest:** Ferramenta utilizada para a implementação dos testes unitários de *Javascript* do projeto. Versão: 22.4.3;
- **GitHub:** Plataforma utilizada para o controle de versão do código-fonte do projeto usando o *Git*. Versão: 2.13;
- **CircleCI:** Plataforma utilizada para a integração e entrega contínua da aplicação. Versão: 2.0;
- **Firebase:** Plataforma utilizada para alguns recursos do projeto como: banco de dados, funções de interação com os dados do banco de dados e armazenamento de imagens das ocorrências. Versão: 4.12.0.

3.4 Aplicativo NotifiCar

Nesta Seção é apresentada a história (Subseção 3.4.1) da aplicação e os principais recursos (Subseção 3.4.2) que ela propõem.

3.4.1 História do Aplicativo NotifiCar

O aplicativo NotifiCar surgiu inicialmente quando os autores deste trabalho perceberam em um grupo de discussões da FGA/UnB criado no *Facebook* que haviam muitos registros de pessoas publicando mensagens avisando sobre carros que estavam, principalmente, com vidros abertos e faróis acesos.

Como o intuito do grupo é discutir assuntos relacionados à universidade, essas publicações acabam poluindo o grupo e desvirtuando o seu objetivo. Como a intenção dessas publicações são de ajudar os companheiros de estudo e/ou trabalho, é interessante manter apenas o conteúdo associado ao contexto do grupo para que assim as informações alcancem todos os integrantes do grupo. Do mesmo modo, as publicações relacionadas ao veículos nem sempre alcançam as pessoas corretas, ou até mesmo demoram para que o façam, visto que em geral o que costuma acontecer, é que as pessoas que conhecem alguém com o veículo semelhante ao informado o avisam sobre o ocorrido, mas muitas vezes o carro não pertencia à ele.

Com isso, surgiu a ideia de criar um aplicativo que pudesse ajudar tanto o grupo a não ter esse tipo de publicação como também que possibilitasse alcançar os responsáveis pelos veículos mais rapidamente e de forma simplificada.

Uma versão do aplicativo foi desenvolvida pelos autores na disciplina de Desenvolvimento de *Software* como trabalho final. No entanto, apesar de funcional, os desenvolvedores julgaram que ainda era necessário realizar diversas melhorias para que o produto estivesse estável e pudesse ser divulgado para utilização.

Durante esse período de análise surgiu a ideia de desenvolver e aprimorar o aplicativo no TCC, mas com um visão mais gananciosa que consistia em desenvolver o *app* para as plataformas *iOS* e *Android*, já que em sua primeira versão o *app* foi desenvolvido apenas para *Android*. Mas, não só transformá-lo em multiplataforma era um dos objetivos, mas também adicionar novos recursos e torná-lo de contexto geral, já que em sua concepção o objetivo era atender o contexto específico da FGA/UnB.

3.4.2 Recursos

O principal recurso do *app* NotifiCar é facilitar a comunicação aos donos de veículos que seu carro se encontra em uma situação adversa, situação esta, que pode ser identificada por qualquer usuário do sistema. O aviso ocorre por meio de uma notificação *push*, recurso

presente nas plataformas *iOS* e *Android* que emite uma notificação no dispositivo do usuário, o que possibilita que o usuário receba a informação instantaneamente. Esse fluxo descrito anteriormente é apresentado na Figura 5.

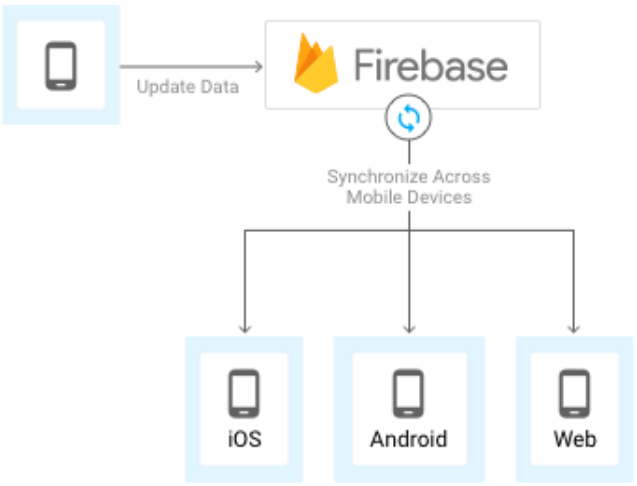

Figura 5 – Fluxo de envio de notificação do Firebase. Fonte: (GOOGLE INC., 2017d)

Além disso, pretende-se utilizar o conceito de cidades inteligentes, onde será possivel possibilitar que os dados registrados no *app* possam ser compartilhados para os cidadãos de maneira que seja possível que os mesmos tenham visão das ocorrências que estão ocorrendo nas cidades.

3.5 Suporte Tecnológico

Disponibilizar um sistema para o usuário final costuma ser uma tarefa árdua e repetitiva devido a diversos fatores como publicar o código no servidor de produção, alterar urls de teste para produção, instalar dependências no servidor, iniciar serviços de banco de dados entre outras tarefas. Aplicativos mobile de forma geral possuem o mesmo processo de desenvolvimento de aplicações web por exemplo, se diferenciando apenas na forma de distribuição do aplicativo, que é realizada através de lojas virtuais como a App Store e Google Play Store, para iOS e Android respectivamente.

Desta forma, é essencial estabelecer um processo que auxilie nessa etapa do desenvolvimento de software, conhecida como implantação. Para o presente trabalho foi implementado um fluxo de integração contínua para garantir o funcionamento do aplicativo, realizando a execução dos testes implementados e também um fluxo de entrega contínua, que disponibiliza novas versões do sistema para o usuário final.

Para implementação da Integração Contínua foi utilizar o serviço na nuvem CircleCI, que disponibiliza máquinas virtuais com sistemas Linux ou MacOS para a execução de aplicações. Essas máquinas virtuais executam containers docker com imagens de sistemas específicos de acordo com a necessidade do usuário. Por exemplo, caso o usuário esteja desenvolvendo uma aplicação Android, será utilizada uma imagem docker contendo todas as dependências necessárias para a compilação e execução de um aplicativo Android. Além disso podem ser utilizadas imagens customizadas, ou instalar dependências adicionais caso seja necessário.

O CircleCI é utilizado em conjunto com o GitHub, outro serviço na nuvem para hospedagem de código fonte. Quando uma alteração é submetida ao GitHub uma requisição HTTP POST é enviada para o servidor do CircleCI indicando que há uma nova alteração, e com isso é disparada a execução da integração contínua, que realizará um série de etapas pré configuradas pelo usuário por meio de arquivo de configuração definido como config.yml.

O arquivo config.yml define os passos que serão realizados no CircleCI por meio da linguagem YAML, aqui são definidos comandos semelhantes aos executados em um ambiente de desenvolvimento, no entanto com o foco voltado para a verificação da integridade da aplicação e adicionalmente as etapas associadas a distribuição da nova versão para os usuários. O Anexo A apresenta o arquivo de configuração utilizado neste projeto.

A configuração apresentada define três "jobs", cada job é uma coleção de passos que são independentes entre sim, mas que em geral são utilizados de forma complementar. O primeiro job chamado de "build"é responsável pela instalação das dependências do projeto e execução dos testes unitários e de interface, além disso após a execução dos testes, o relatório resultante é enviado para outra ferramenta (Coveralls) que gera um relatório que pode ser acessado de forma online e fornece uma estimativa da quantidade de código coberto pelos testes. O seguindo job, android, é responsável pela geração do aplicativo Android, nesse job são gerados os arquivos de assinatura do aplicativo, e o envio do aplicativo gerado para a Play Store. O terceiro job, iOS, de forma similar ao anterior, é responsável por compilar e gerar o aplicativo iOS.

Em seguida são definidos os chamados "workflows", que funcionam como linhas de produção, aqui é definida como os jobs devem ser executados, podem ser definidos vários workflows e os jobs podem ser executados de forma paralela ou sequencial. Como definido no arquivo, primeiramente é exeutado o job build para que todas as dependências sejam configuradas, ao término deste job, são iniciados os jobs android e ios de forma paralela, visto que ambos são independentes. Caso os jobs sejam executados com sucessos uma nova versão do aplicativo estará disponível nas lojas, é importante destacar também que caso um dos jobs falhe, ios ou android, o outro não será afetado.

Devido ao serviço do CircleCI ser limitado para planos gratuitos, em especial a

utilização de máquina com MacOS, a integração contínua foi configurada para executar os testes nas branches master e staging, e a entrega contínua é executada apenas quando há alterações na branch master.

Como já citado anteriormente, para realizar o envio dos aplicativos para as respetivas lojas, é utiliza a ferramenta fastlane, que automatiza o processo de publicação dos apps. Para a plataforma android o fastlane realiza apenas a compilação do aplicativo em modo Release e o envia para a loja. Já para o iOS o fastlane trata também da assinatura do código, acessando os certificados e arquivos de distribuição gerador pela Apple.

Esse processo inclui a instalação dos certificados na máquina que gera o arquivo final a ser enviado para a loja. No entanto todos os certificados possuem um período para expiração, casos os certificados expirem o próprio fastlane gera novos certificados e ignora os inválidos. Todo esse processo é feito por meio da ferramenta "match"fornecida pelo fastlane. Esse processo também é realizado por meio de um arquivo de configuração chamado Matchfile, este arquivo possui as informações do repositório com os certificados e as devidas permissões de acesso.

3.5.1 Linguagens e *Frameworks*

O aplicativo Notificar será desenvolvido com o *framework React Native*, que utiliza a linguagem de programação *JavaScript* e possibilita o desenvolvimento de aplicações nativas para *iOS* e *Android*. Atualmente, este *framework* se encontra na versão 0.49.

O *React Native* foi criado em 2015 pelo *Facebook* e tem como *slogan* a frase *"Learn once, write anywhere."*, que representa a capacidade de aprender apenas uma linguagem de programação e possibilitar o desenvolvimento de aplicações nativas *iOS* e *Android*, com a ressalva de que alguns ajustes de código podem ser necessários devido as especificidades de cada plataforma (FACEBOOK INC., 2017c).

Já o *JavaScript* é uma linguagem de programação interpretada e orientada a objetos. É uma das linguagens mais utilizadas no mundo, e apesar de seu uso ter sido durante muito tempo predominantemente usado no desenvolvimento *Web*, com o surgimento dos *frameworks* de desenvolvimento *mobile* multiplataforma sua utilização tem aumentado cada vez mais nessa área (CASS, 2017).

3.5.2 Testes

Para a implementação dos testes unitários será utilizado o *framework open source Jest*, também desenvolvida pelo *Facebook* e que já vem integrada ao *React Native* (FACEBOOK INC., 2017b). O *Jest* é um *framework* que não requer configuração, sendo necessário apenas adicioná-lo ao projeto, apesar de já vir por padrão com o *React Native*. Adicionalmente, também permite realizar o *mock* dos componentes do *React Native*, o

que consiste em simular o comportamento de um objeto real, para que assim seja possível testar componentes que dependem de outros, mas que não necessariamente se tem acesso a eles em determinados momentos.

Apesar dos testes unitários garantirem que o comportamento de um trecho de código implementado não esteja com seu comportamento alterado ou que ele funcione de acordo com o esperado, ainda assim pode acontecer que uma determinada funcionalidade não apresente o comportamento adequadamente devido a outros fatores relativos a implementação. Um exemplo para essa situação pode ser representada por um método que retorne a temperatura de um sensor, e de acordo com a temperatura, seja necessário alterar a cor de um elemento na tela para dar destaque a essa informação, no entanto apesar do método retornar o valor correto o elemento na tela não tem sua cor alterada, logo é necessário outro recurso para capturar esse tipo de erro.

Os testes funcionais são utilizados para resolver o problema citado acima, além de garantir que a integração de novas funcionalidades ao sistema não quebrem o funcionamento das já existentes. Para este projeto será utilizada a biblioteca *Detox* (FACEBOOK INC., 2017a), que permite a execução dos testes tanto em emuladores como em dispositivos reais, além de ser integrável aos sistemas de integração contínua.

3.6 Testes Unitários

Para o desenvolvimento dos testes unitários foi utilizado o *Jest* que discutimos na Subseção 3.5.2, como estamos utilizando o *Framework Redux*, os testes unitários foram feitos pensado na estratégia de verificação dos dispatchs, que são os retornos padrões dos métodos das *Actions*, onde esses retorno possuem dois valores o *type* e o *payload*, *type* é o que o *Reducer* utiliza para entrar em um case e realizar alguma alterar ou não as variáveis globais da aplicação, seja pelo atributo *payload* ou não.

Logo, a verificação das respostas dos métodos foi a partir do retorno correto para o *Reducer*, sendo o método a ser testado síncrono ou assíncrono. Dois exemplos de testes são apresentados nas Figuras 6 e 7.

3.6.1 Gerência de Configuração de *Software*

Mudanças durante o desenvolvimento de *software* acontecem a todo momento, os requisitos, o ambiente do projeto, o entendimento da equipe de desenvolvimento e do cliente mudam. GCS é um conjunto de atividades, que permitem de forma ordenada controlar as mudanças no desenvolvimento de *software*, mantendo a integridade e a estabilidade do *software*.

```
describe('Account Actions Sync Methods', () => {
    it('should create an action to write a vehicle', () => {
        const vehicle = 'ABC-1234';
        const expectedAction = {
          type: Types.WRITE_VEHICLE,
          payload: vehicle
        };

        expect(actions.writeVehicle(vehicle)).toEqual(expectedAction);
    });
```

Figura 6 – Exemplo de teste unitário síncrono implementado na aplicação. Fonte: Autores

```
describe('Account Actions Async Methods', () => {
    it('should execute addVehicle', () => {
        const store = mockStore({});
        return store.dispatch(
            dispatch =>
                firebaseApp.database().ref('/users/test/vehicles')
                    .child('ABC-1234')
                    .set(true)
                    .then(() => dispatch(
                        {
                            type: Types.ADD_VEHICLE_SUCCESS
                        })
                    ))
                .then(() => {
                    const actions = store.getActions();

            expect(actions[0]).toEqual({
                type: Types.ADD_VEHICLE_SUCCESS
            });
        });
    });
```

Figura 7 – Exemplo de teste unitário assíncrono implementado na aplicação. Fonte: Autores

3.6.1.1 Política de *Branch*

Em um processo de GCS é essencial estabelecer o fluxo de trabalho por meio de *branches* para que se possa garantir a organização e controle do respositório. Além disso, deve-se garantir a integridade do código que será utilizado em produção. A Figura 8 detalha a política proposta para este trabalho.

A branch *master* contém o código utilizado em produção, ou seja, a versão final

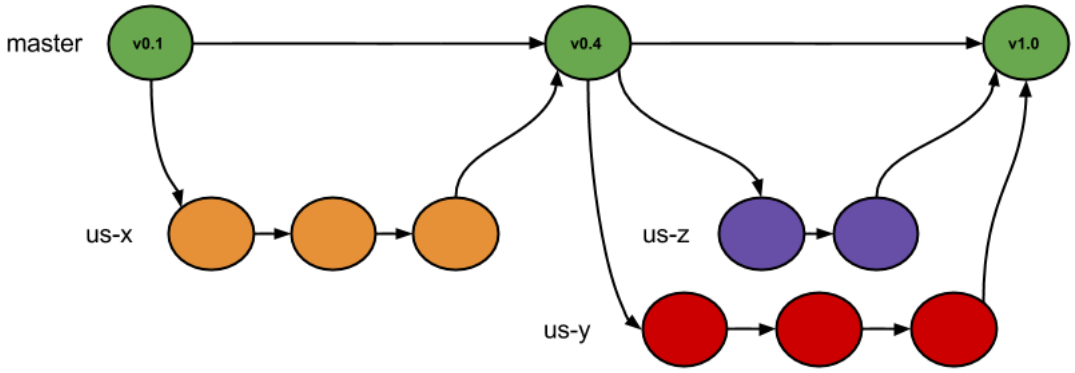

Figura 8 – Política de *branches* do repositório. Fonte: Autores

do *app* que é disponibilizado nas lojas virtuais. Todo código antes de ser integrado à esta *branch* deve ser analisado e testado.

As *branches* auxiliares são utilizadas para o desenvolvimento de novas funcionalidades ou correção de *bugs*, essas *branches* devem sempre ser criadas a partir da *master*. Ao término do desenvolvimento, deve ser realizado um *pull request* para a *master*, de forma que outro desenvolvedor poderá avaliar o código desenvolvido e integrá-lo à versão de produção. Quando o *pull request* é aceito a *branch* em questão deve ser excluída, afim de manter o repositório limpo e com menor tamanho devido a quantidade de arquivos.

3.6.1.2 Integração Contínua

O *Travis CI* é uma ferramenta gratuita de integração contínua que em conjunto com ferramentas de controle de versão como o *GitHub* realizam a execução de testes unitários, teste funcionais e geração de *build* do código fonte (TRAVISCI, 2017). Este processo é realizado por meio de um arquivo de configuração chamado *.yml*, este arquivo contém as instruções que definem o processo de execução dos testes, geração de *builds* e relatórios de cobertura de testes e análises de código.

3.6.1.3 *Deploy* Contínuo

O *FastLane* é uma solução de *deploy* contínuo para aplicações *mobile* e de código fonte aberto (FASTLANE, 2017). Possibilita a distribuição de aplicações *mobile* nas lojas *App Store* e *Google Play* de forma automatizada, visto que o processo de distribuição é fixo, essa tarefa pode ser realizada de forma sistemática pela ferramenta. De forma similar ao *Travis CI* o processo de *deploy* também é realizado por meio de um arquivo de configuração chamado *FastFile* que define as instruções e configurações necessárias para que cada um dos *apps* possam ser distribuídos nas suas respectivas plataformas.

O processo de *deploy* é iniciado ao realizar o *merge* em uma *branch*, no nosso caso é na *branch master*. Após o *merge*, o *script* de integração contínua é disparado e executado,

este por sua vez executa os testes unitários e funcionais e a gera uma *build*. Caso a etapa anterior seja concluída com êxito, o código é então enviado para o *FastLane*, que realiza as etapas necessárias para geração e disponibilização do *app* na sua respectiva loja. Esse processo é apresentado detalhadamente na Figura 9.

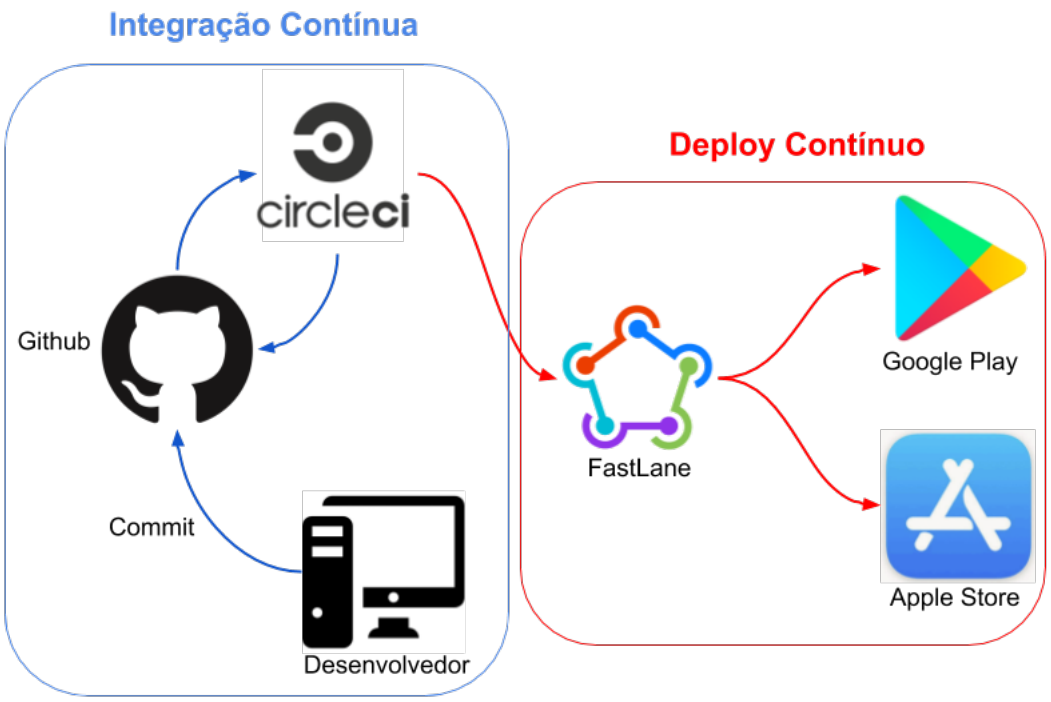

Figura 9 – Processo de Integração Contínua e *Deploy* Contínuo. Fonte: Autores

Essa integração entre ambos serviços, *GitHub* e *CircleCI*, é realizada por *Webhooks*, que são ações disparadas quando um determinado evento ocorre no repositório. Para que isso ocorra o *CircleCI* é adicionado como um *Webhook* ao repositório em questão, e sempre que ocorrer um *push* no repositório, é disparada uma requisição *http* do tipo *POST* para a *API* do *CI*, que será executado.

Para definir as ações que serão realizadas pelo *CircleCI* é necessário criar um arquivo chamado *config.yml* que contém as configurações e as tarefas a serem executadas. O arquivo de configuração utilizado neste trabalho está disponível no Anexo A e é detalhado a seguir.

A primeira linha do arquivo especifica a versão do *CircleCI* utilizada, atualmente em sua segunda versão.

Em seguida, temos a definição dos *jobs*, que são blocos de execução que podem ser executados de forma paralela, ou sequencial. Aqui temos a definição de três *jobs*: *build*, *android* e *ios*. O *build* é responsável por fazer a configuração inicial do projeto, fazendo o download do código fonte, instalando as dependências e executando os testes de *Javascript*.

Logo abaixo é definido o job *android*, este é responsável pela geração do aplicativo android, a assinatura do .apk(*Android Package*), captura das telas do aplicativo e por fim o envio das novas informações geradas para a loja.

De forma similar ao demais, o job *ios* gera o app ios, arquivo .ipa (*iOS App Store Package*), realizar a assinatura do app, captura as telas do aplicativo e o envia para a App Store.

Abaixo são definidos os *workflows*, que podem ser associados a uma linha de produção, para este trabalho foi definido apenas um workflow chamado node-android-ios. Este workflow é responsável por executar os jobs definidos anteriormente, no entanto, esses jobs são executados sob algumas condições. Todos os jobs são executados apenas quando ocorre um push na branch master do repositório.

Já os jobs android e ios possuem o job anterior como dependência, sendo assim, ambos só serão executados após a execução com sucesso do build. Estes dois jobs são executados de forma paralela e independente, sendo assim caso um dos dois falhes a execução do outro não é afetada. Ao término de suas execuções um novo aplicativo é enviado para as respectivas lojas de cada plataforma, caso algum dos passos definidos falhe o restante é abortado e a integração é dada como quebrada, ou executada com falha.

A configuração utilizada com apenas uma branch apesar de não ser ideal, por não ser possível testar a integração ao se desenvolver em diversas branchs, se fez necessária por utilizarmos uma conta gratuita no CircleCI a qual possui recursos limitados. Logo, para que não seja atingida a cota máxima o uso do CI ficou restrito à branch master, diminuindo assim o número de execuções e tempo de uso da máquina virtual disponibilizada para contas gratuitas.

3.6.1.4 Entrega Contínua

Dentro do ciclo de desenvolvimento de um produto de *software*, existe a fase de entrega que consiste em disponibilizar a aplicação para uso, seja para usuários ou outros sistemas. Esta fase se resume ao conjunto de práticas que possibilitam que novos incrementos de *software* possam ser integrados ao código já existente (HUMBLE; FARLEY, 2013).

De forma geral, antes das aplicações serem disponibilizadas para uso, são reali-

zadas atividades de verificação de qualidade, corretude de funcionamento, otimização de recursos como imagens, e por fim a aplicação está pronta para ser utilizada pelo usuário final, ou quase isso. Apesar de se ter um "produto"pronto para uso, ainda é necessário realizar a disponibilização desse software, seja em um servidor (aplicações web), ou em uma loja digital (aplicativos mobile), ou um arquivo executável para instalação (aplicações desktop).

Mesmos tipos de aplicações podem exigir um processo diferente para a geração do produto final, como é o caso de aplicações web que utilizam linguagens interpretadas e compiladas. Quando se utiliza uma linguagem compilada é necessário realizar todo o processo de compilação após uma alteração para que por fim seja obtido o código pronto para uso. Já no caso de linguagens interpretadas não há a necessidade recompilar o código, apenas realizar as alterações necessárias, e no pior dos casos reexecutar a aplicação.

Para os aplicativos mobile o processo é similar, se diferenciando apenas pelo arquivo final gerado.

Na plataforma Android é gerado um arquivo .apk, que é distribuído para instalação oficial por meio da *Google Play Store*, este mesmo arquivo pode ser utilizado para instalar o app no dispositivo diretamente, sem o intermédio da loja, no entanto algumas questões de segurança estão associadas a esse método. Esse arquivo contém o código fonte compilado, otimizado e ofuscado, além de todos os recursos gráficos como ícones, imagens, sons, etc.

Para realizar o processo de entrega continua dos aplicativos foi utilizado outra aplicação fornecida como serviço chamada *fastlane*. O *fastlane* permite a captura das telas do aplicativo de forma automatizada, liberação de versões beta e final para as lojas e assinatura dos aplicativos.

De forma similar ao circleci o fastlane utiliza arquivos de configuração para ser executado, no entanto sendo executado de forma local e não em um servidor. Dois arquivos de configuração são utilizados, são eles Appfile e Fastfile.

O arquivo Appfile possui dois atributos que são o caminho para um arquivo .json que contém uma chave de acesso que permite a publicação dos aplicativos nas lojas e o nome do pacote da aplicação.

O arquivo Fastfile possui as configurações das ferramentas disponibilizadas pelo fastlane. Neste arquivo são especificadas as chamadas *lanes*, similares aos jobs especificados no CI, cada *lane* é responsável por uma tarefa e podem ser definidas quantas *lanes* forem necessárias.

A lane test é responsável apenas pela execução dos testes da plataforma específica (android/ios). A *lane playstore* gera o aplicativo de *release* do app e por fim o envia para loja virtual.

O arquivo *Appfile* contém a definição de apenas 2 parâmetros, o json_key_file é a chave de acesso que o fastlane utiliza para enviar arquivos para a *Goole Play* e *App Store* como apkś e ipaś, imagens, e descrições do app. Já o package_name é o identificador do pacote da aplicação, que é única para cada aplicativo.

```
1 json_key_file("fastlane/Google_Play_Android_Developer.json")
2 package_name("com.jldevs.notificar")
```

O arquivo *Fastfile* contém a definição de rotinas que poderão ser executadas pelo *fastlane*. De forma similar a um método, aqui são definidas "*lanes*"que especificam um conjunto de passos a serem executados de forma automatizada. Aqui foram definidas 2 *lanes*, uma *lane* de "*test*"para realizar a execução dos testes específicos da plataforma, android ou ios, e a *lane* "*playstore*", essa é a *lane* responsável por fazer o upload do app gerado para a respectiva loja, para que isso ocorra são definidas duas ações, uma para gerar o aplicativo em modo *release*, e em seguida o envio do apk para a loja.

```
1  default_platform(:android)
2
3  platform :android do
4    desc "Runs all the tests"
5    lane :test do
6      gradle(task: "test")
7    end
8
9    desc "Deploy a new version to the Google Play"
10   lane :playstore do
11     gradle(
12       task: 'assemble',
13       build_type: 'Release'
14     )
15     upload_to_play_store
16   end
17 end
```

3.7 Licença

Ao decorrer do projeto pensamos em deixá-lo fechado, porém ao decorrer do desenvolvimento vimos que é vantajoso deixá-lo livre, visto que com ele teríamos um bom projeto para portfólio, poderíamos utilizar gratuitamente ferramentas que ajudam no processo de engenharia de software que definimos nos primeiros tópicos deste trabalho e que após a finalização do trabalho de conclusão de curso, haveria a possibilidade de termos

a ajuda da comunidade. Logo, foi necessário escolher qual a licença de software livre escolheríamos dentre as três categorias existentes:

- Licenças Permissivas
- Licenças Recíprocas Totais
- Licenças Recíprocas Parciais

Observando os conceitos das três categorias, foi escolhida a licença GPL, que está na categoria de recíprocas totais, como pensamos em deixar esse projeto sempre livre e gratuito, está é a licença ideal já que toda a cópia, execução, modificação e redistribuição do nosso projeto deve continuar livre.

3.8 Banco de Dados

O *Firebase* é uma plataforma que disponibiliza infraestrutura e *software* como serviço, foi criado pela *Firebase Inc.* em 2011 e adquirida pelo *Google Inc.* em 2014. O *Firebase* fornece a maior parte de seus recursos de forma gratuita, porém limitada, para maior proveito dos recursos disponíveis podem ser contratados plano com franquia de uso ou com o modo 'pague a medida que usa' (FIREBASE, 2017).

Para este trabalho serão utilizados os serviços de autenticação, banco de dados e armazenamento de arquivos no plano gratuito, visando a utilização de recursos já consolidados no mercado mas que podem ser migrados para outras plataformas a qualquer momento, caso necessário.

O banco de dados fornecido pelo *Firebase* além de possibilitar o armazenamento e sincronização de dados em tempo real, utiliza o conceito *NoSQL* baseado em documentos *JSON*, o que permite alta escalabilidade e flexibilidade na estruturação do armazenamento de dados.

Ao decorrer do desenvolvimento verificou-se que poderíamos explorar mais os serviços gratuitos que o *Firebase* nos proporciona, e então passamos a utilizar o serviço de funções, onde podemos fazer alguns *scripts* para observar as alteração de algum nó e realizar alguma operação no banco de dados e o serviço de armazenamento, que permite que a gente faça o *upload* de arquivos no *Firebase*. Com a evolução da aplicação *NotifiCar*, por conta da adição de novas funcionalidades, o banco de dados se alterou várias vezes, entretanto, o modelo *noSql* do banco de dados da aplicação na sua versão atual (2.0.0) é apresentado na Figura 10.

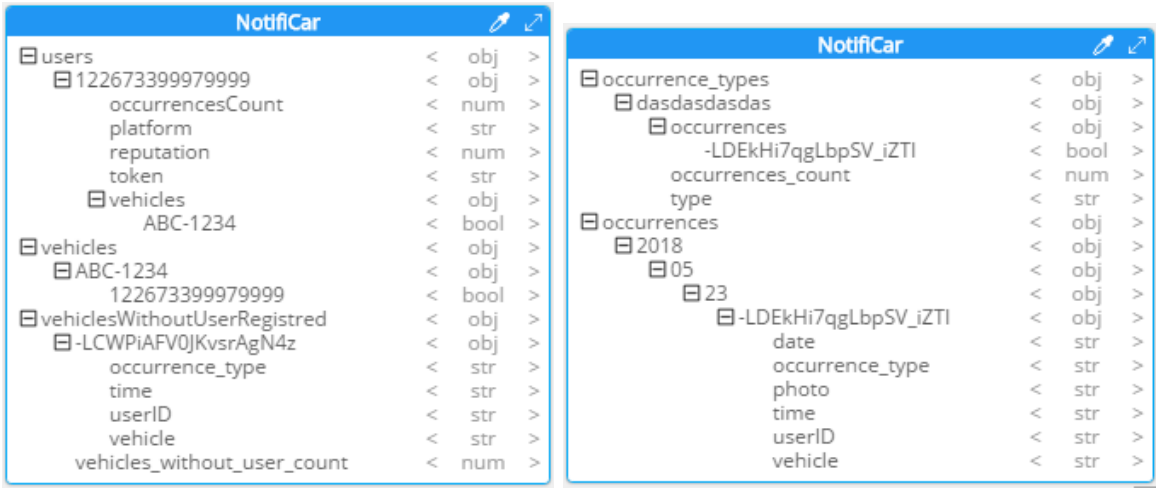

Figura 10 – Exemplo do banco de dados atual da aplicação NotifiCar v2.0.0. Fonte: Autores

3.8.1 Análise Estática de Código

As ferramentas de análise estática de código tem por objetivo aumentar a qualidade de código por meio de processos que avaliam o código fonte e buscam falhas de implementação sem que este seja executado. Essas falhas podem estar relacionadas com variáveis não inicializadas, falhas de segurança, carregamento de recursos entre outros. A análise estática também pode ser utilizada para avaliar os padrões de implementação do código para determinadas linguagens (DELEY; GJORGJEVIKJ, 2017).

O *ESLint* é uma ferramenta *open source* para avaliação de estilo de código fonte e identificação de más práticas associadas ao padrões do *javascript*. O *ESLint* já vem por padrão pré-configurado, no entanto pode-se especificar novas regras a serem obedecidas e o *ESLint* fará a avaliação se o código desenvolvido está dentro dos padrões estabelecidos. Esta técnica ajuda a manter a padronização e organização do código, principalmente em uma equipe com vários desenvolvedores onde cada um possui o seu estilo próprio.

Já a ferramenta *Flow* é utilizada para realizar a analise estática de escrita para a linguagem *javascript*. Dessa forma, pode-se antecipar a identificação de erros que seriam percebidos apenas durante a execução do código fonte, visto que por não ser uma linguagem compilada, não é possível detectar erros sem a execução do código ou a nível de compilação.

3.8.2 Repositório

O *GitHub* é uma ferramenta gratuita para hospedagem de código fonte e utiliza o *Git* como ferramenta de controle de versão, além disso possui integração com diversas ferramentas e plugins que auxiliam no desenvolvimento de aplicações (GITHUB INC., 2017b).

Atualmente é uma das plataformas mais populares para compartilhamento de código, com mais de 69 milhões de projetos hospedados (GITHUB INC., 2017a). Outro aspecto importante do *GitHub* é que muitas empresas hoje já o utilizam para procurar novos desenvolvedores para ocupar suas vagas de desenvolvedores.

3.9 *Design*

Aplicações *mobile* tem características de interface e interação com o usuário diferentes de aplicações *web* ou *desktop*, uma vez que são executadas em dispositivos móveis. No entanto, cada uma das plataformas para as quais serão desenvolvidas os aplicativos já definem suas próprias heurísticas de usabilidade e de construção de interface.

As aplicações desenvolvidas para *Android* devem ser criadas com base nos guias do *Material Design*. O *Material Design* é um guia completo para construção de interfaces de usuários, e que possui diretrizes tanto para os aspectos visuais como para os aspectos referentes a experiência do usuário (GOOGLE INC., 2017b).

Já as aplicações desenvolvidas para *iOS* seguem as diretrizes apresentadas no *Human Interface Guidelines*. Este também é um conjunto de guias e boas práticas para a criação das interfaces de usuário, recursos de acessibilidade e experiência de usuário na plataforma *iOS* (APPLE INC., 2017b).

Vale ressaltar que as diretrizes de cada uma das plataformas se aplicam também às aplicações desenvolvidas para *desktop*, *smart watches* e *smart* TV's.

4 Resultados

4.1 O Aplicativo

Nesta seção será demonstrada as funcionalidades do aplicativo **NotifiCar** na versão mais recente (v2.0.0), o repositório da aplicativo se encontra no seguinte endereço: <https://github.com/NotifiCar/>.

4.1.1 *Login/Logout* e Políticas de Privacidade

Para efetuar o *login* é necessário que o usuário tenha uma conta no *Facebook* e toque no botão da opção 1 como é mostrado na Figura 11, ao efetuar o login o usuário poderá acessar todas as funcionalidades da aplicação. Caso o usuário deseje verificar as políticas de privacidade da aplicação, ele deve abrir o menu no canto superior direito e escolher a opção "Políticas de Privacidade"como é mostrado na opção 1 da Figura 12. Quando o usuário não desejar receber mais notificações da aplicação é necessário que o mesmo efetue o *Logout*/Sair da aplicação escolhendo a opção 2 da Figura 12.

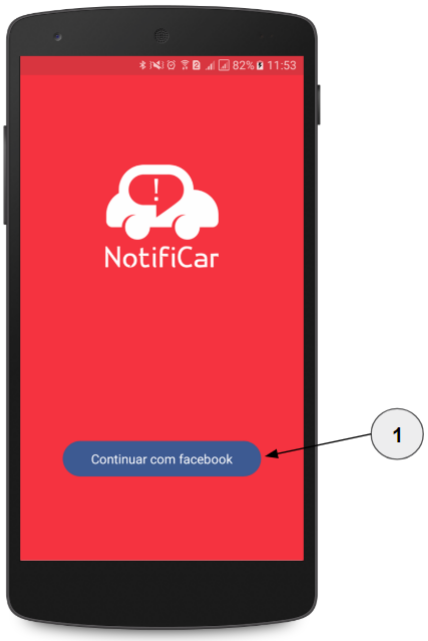

Figura 11 – Tela de *login*. Fonte: Autores

4.1.2 Lista de Ocorrências

A lista de ocorrências(opção 1) pode ser vista na Figura 13 na aba FEED, nesta lista está as ocorrências sobre seus veículos cadastrados na sua conta, essas ocorrências são

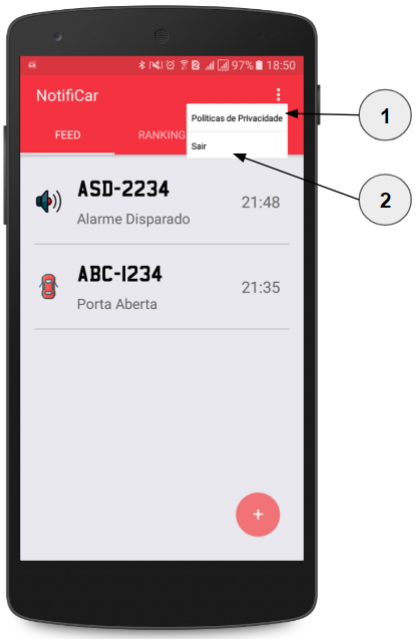

Figura 12 – Tela da lista de ocorrências com o *menu* aberto. Fonte: Autores

referentes ao dia atual. Os itens dessa lista(opção 2) possuem alguns detalhes, onde pode ser visto o tipo da ocorrência pela imagem que representa ela(opção 3) ou pelo texto(opção 5), o horário que a ocorrência foi registrada(opção 4), a placa do veículo que foi feita à ocorrência(opção 6), e caso algum veículo seu tenha recebido uma ocorrência, seu aparelho receberá uma notificação(opcão 7), ao tocar no botão da opção 8 será aberta uma janela onde pode ser registrado uma ocorrência para um veículo, mostrada na Subseção 4.1.4 e a opção 9 é referente ao menu que foi citado na Subseção 4.1.1. Pode-se também tocar em um item da lista(opção 2) e será aberta uma janela(Figura 14), onde haverá detalhes da pessoa que fez o registro da ocorrência(SubSeção 4.1.3).

4.1.3 Perfil do Usuário que Realizou a Ocorrência

Esta janela(Figura 14) possue os detalhes da pessoa que fez o registro da ocorrência para o seu veículo, aqui é possível visualizar a foto do Facebook da pessoa(opção 1), o nome da mesma(opção 2), a quantidade de ocorrências que essa pessoa já cadastrou(opção 3), a reputação dela(opção 4), um botão que você pode tocar e aumentar a reputação da pessoa que cadastrou a ocorrência, afirmando que está ocorrência foi útil(opção 5) e diminuindo a reputação dela caso toque no botão da opção 6 dizendo que a ocorrência realizada por ele foi falsa.

4.1.4 Cadastrar Ocorrência

Esta janela(Figura 15) é aberta ao tocar no botão da opção 8 da Figura 13 para cadastrar uma ocorrência, onde é possível selecionar o tipo de ocorrência que o veículo

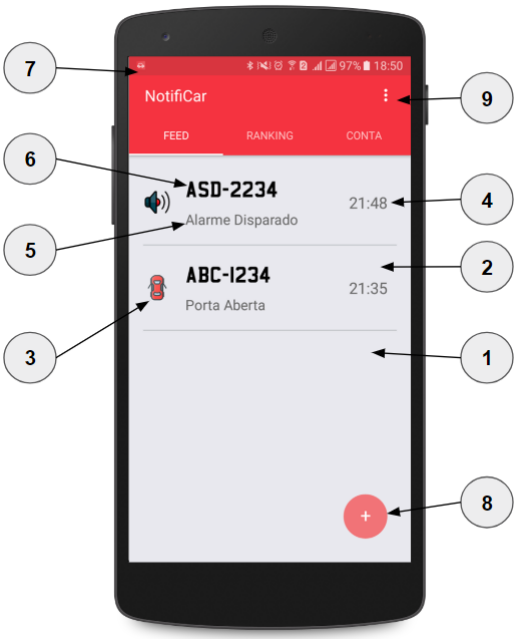

Figura 13 – Tela da lista de ocorrências. Fonte: Autores

Figura 14 – Tela da janela de reputação. Fonte: Autores

se encontra(opção 1), digitar a placa do veículo(opção 2), tirar uma foto ou selecionar da biblioteca(opção 3), e fazer o registro(opção 4). A opção 3 é opcional, porém ao fazer o registro de uma foto trás maior confiança ao dono do veículo que aquila ocorrência é verdadeira.

Figura 15 – Janela de cadastro de uma nova ocorrência. Fonte: Autores

4.1.5 Lista de Placas

A lista de placas é possível ser encontrada na aba CONTA(Figura 21), nessa lista é possível ver as placas que estão associadas a sua conta(opção 1) e cada item dessa lista contém o texto da placa(opção 2). Caso o usuário deseje registrar uma nova placa é necessário tocar no botão da opção 3 para abrir a janela(Figura 17) e para remover qualquer placa da lista é necessário que o usuário deslize o item da lista para esquerda para que o botão deletar apareça(opção 4) e ao tocar nele a placa é removida.

4.1.6 Cadastrar Placa

Esta janela(Figura 17) possue apenas 1 campo para preencher(opcao 1), que é o local onde se põem a placa do veículo que deseja ficar observando. Ao tocar no botão da opção 2 a placa escrita será cadastrar na sua conta.

4.1.7 *Ranking* dos Tipos de Ocorrência

O *ranking* de tipos de ocorrências é possível ser encontrado na aba *RANKING*(Figura 18), este *ranking* possui a representação de um pódium(opção 1), onde a ocorrência que está no topo(opção 2) é a que mais ocorreu, ao lado esquerdo o tipo de ocorrência que está em segundo e ao lado direito o que está em terceiro, abaixo há a lista dos tipos de ocorrência que não foram para o pódium(opção 3) e a opção 4 representa o total de ocorrências registradas na aplicação. A quantidade de vezes que ocorreu cada tipo de ocorrência é representado por um número verde, que está debaixo de cada item no pódium e no lado

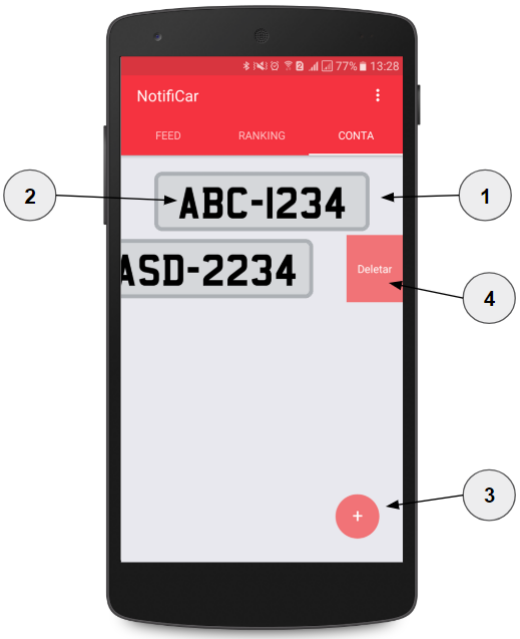

Figura 16 – Tela da lista de placas com opção deletar aberta. Fonte: Autores

Figura 17 – Janela de cadastro de uma nova placa. Fonte: Autores

direito de cada item da lista.

4.2 Relatório de Testes

Os testes unitários foram implementados apenas para o pacote app, o qual contém a implementação da lógica do aplicativo, excluindo arquivos de configuração, imagens e arquivos de definição de constantes comuns. Também foram implementados testes funci-

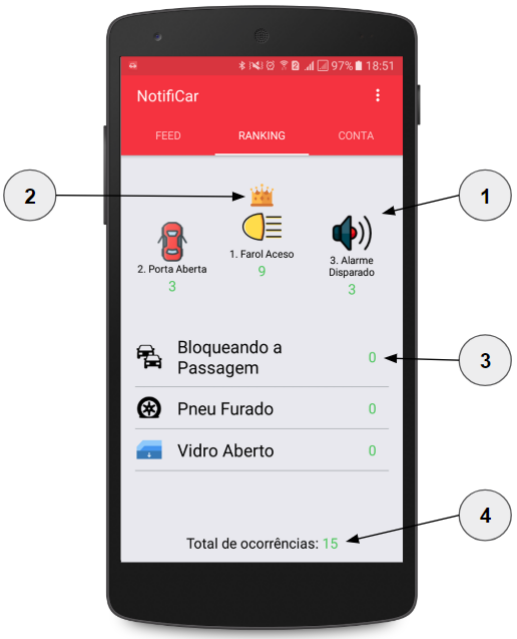

Figura 18 – Tela do *ranking* de tipos de ocorrência. Fonte: Autores

onais.

Além disso, os testes foram subdivididos em duas categorias que compreendem testes comportamentais e testes de renderização. Os testes unitários tem como objetivo garantir o comportamento dos métodos implementados. Os testes de renderização foram criados devido a característica de componentização existente no React, que permite a definição de um componente que pode ser reutilizado em diversas telas, estes componentes são independentes entre si.

Como os componentes são reaproveitados é importante verificar se sua renderização está sendo realizada de forma correta, para isso são utilizados os testes de snapshot. O teste consiste na geração de um snapshot, que é o componente convertido para código html/jsx. Em seguida é realizada uma nova renderização do componente e comparada com o snapshot original. Caso algum atributo do componente tenha sido alterado o teste irá falhar, "em geral" esse teste é utilizado para evitar alterações indevidas em um determinado componente, já que após a alteração do mesmo o teste irá falhar, logo caso o componente realmente tenha sofrido uma atualização deverá ser gerado um novo snapshot que será usado para comparação nos próximos testes.

Os testes automatizados servem para simular a utilização do aplicativo por um usuário real, com passos preestabelecidos que executam as funcionalidades do sistema. Apesar desse tipo de teste não simular todas as possibilidades de interação com o sistema, ainda é uma boa prática para simular os casos de uso mais prováveis.

No entanto foi encontrado um problema para a utilização dos testes no serviço de Integração Contínua devido ao login com o facebook. A ferramenta utilizada, detox, que

é específica para javascript, bem como as próprias ferramentas de testes automatizados nativas, Espresso e UITest, não são compatíveis com o componente utilizado pelo facebook para realizar o processo de autenticação. Devido a esse problema não é possível realizar a autenticação no aplicativo já que a única forma de login é com uma conta do facebook. Sendo assim, os testes automatizados não foram integrados no serviço de CI, mas ainda assim foram utilizados localmente no desenvolvimento.

Na primeira coleta da cobertura de código foi obtido um resultado de 46%, no entanto durante uma das refatorações que foram realizadas foi percebido que as pasta "componentes"não estava incluída na cobertura. Após realizar a correção da configuração da ferramente de testes e realizar uma nova coleta dos testes foi obtido um valor de apenas 11% de cobertura de código. Após identificada a baixa taxa de cobertura foram planejadas ações para implementação de mais testes. A Figura 19 detalha a cobertura de código obtida para cada módulo testado, e a Figura 20 detalha a execução das suítes de testes implementadas.

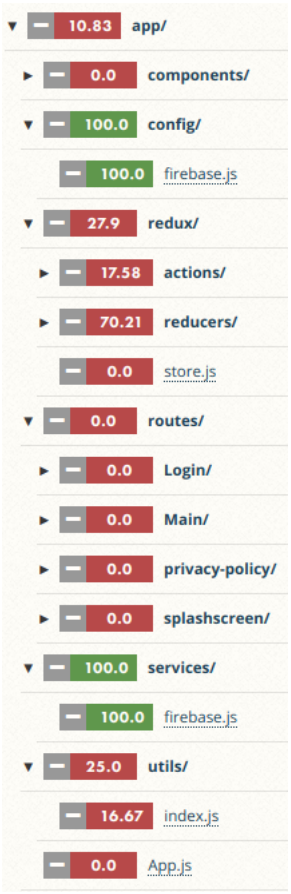

Figura 19 – Cobertura de Código

Com os testes funcionais foi possível testar o comportamento do aplicativo antes da autenticação com o facebook e após a autenticação, não sendo possível testar o processo de autenticação como já citado anteriormente.

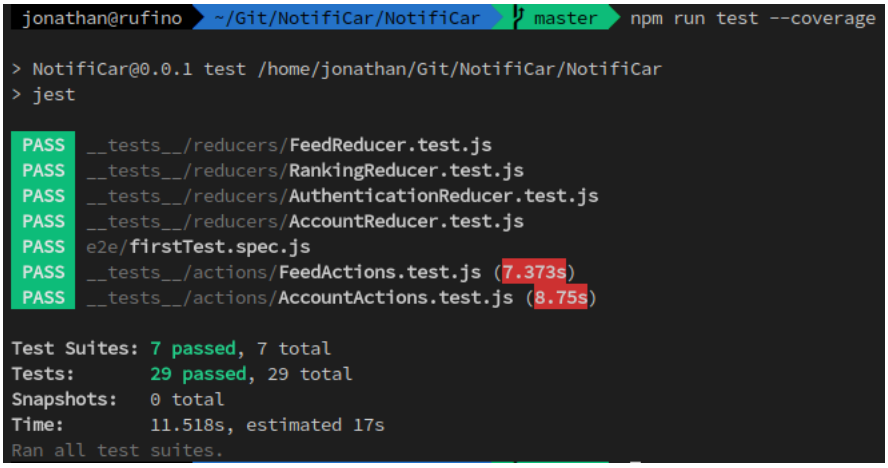

Figura 20 – Execução de Testes

4.3 Estrutura do Projeto

Para garantir uma boa organização do projeto e sua manutenabilidade, foi utilizada uma estrutura de pastas para separar os arquivos do projeto de forma que arquivos relacionados ao mesmo princípio ficassem agrupados.

O React Native não impõe uma estrutura rígida para o código fonte, exigindo apenas que o arquivo de entrada da aplicação index.js esteja presente na raiz do projeto, ficando a cargo dos desenvolvedores definir a estrutura adequada para o projeto. Como o react native é implementando em cima do react.js que é uma biblioteca para desenvolvimento de frontend web, muito do que se tem no mercado em termos de estruturação tem como base as práticas utilizadas na web. Após a análise e diversos códigos fontes e recomendações de outros desenvolvedores que utilizam o react native foi estabelecida a estrutura apresentada a seguir (a estrutura pode ser vista também na Figura 21):

- NotifiCar: é a pasta raiz, a qual contém todo o código fonte do projeto
- ___mocks___: esta pasta contém os arquivos que realizam o mock dos componentes utilizados durante a execução dos testes
- ___tests___: esta pasta agrupa todos os arquivos de testes unitários e de snapshots, além dos snapshots gerados
- .circleci: esta pasta contem os arquivos de configuração do serviço de integração continua
- android: contém o código nativo para a aplicação android
- app: esta pasta contém toda a lógica do aplicativo e é subdividida em outras pastas
  - assets: Contém arquivos estáticos como imagens e fontes,

  - commom: Contém arquivos com variáveis comums utilizadas em todo o aplicativo como cores, textos, valores etc
  - components: Contém os componentes reaproveitáveis do app. Todo componente é constituido por 3 arquivos, são eles o index.js, styles.js e Componente.js. Onde o arquivos index.js é responsável simplesmente pela exportação do componente, o styles.js contém as variáveis de estilização do componente, e o Componente.js contém a lógica do componente propriamente dita
  - config: A pasta config contém os arquivos de configuração para seriços externos, como chaves de acesso a APIS, variáveis de ambiente etc
  - redux: Esta pasta contém as actions e os reducers que são elementos utilizados na implementação do redux
  - routes: A pasta routes contém as rotas ou telas do aplicativo, que também são componentes mas em geral não é possível reaproveitá-las como um componente genérico
  - services: contém os arquivos para acesso a serviços externos como o firebase ou apis
  - utils: conjunto de métodos genéricos reutilizados no código
  - App.js: Este arquivo é responsável pela inicialização do aplicativo.

- e2e: contém a implementação dos testes automatizados
- functions: esta pasta contém os arquivos de implementanção dos gatilhos que são disparados no firebase quando eventos especificados são identificados no banco de dados
- ios: contém o código nativo para a aplicação ios
- node_modules: esta pasta contém as dependencias do projeto

4.4 Comparativo Android x iOS

O desenvolvimento de uma aplicação react native apesar de ter como objetivo a escrita de um único código para a geração de 2 aplicativos em plataformas diferentes, ainda é necessário em alguns momentos a implementação de um código específico para cada plataforma.

As implementações específicas podem se necessárias por diversos motivos, como por exemplo um componente que existe em uma plataforma mas não existi na outra, ou ainda caso exista, possui um comportamento diferenciado como por exemplo a status bar.

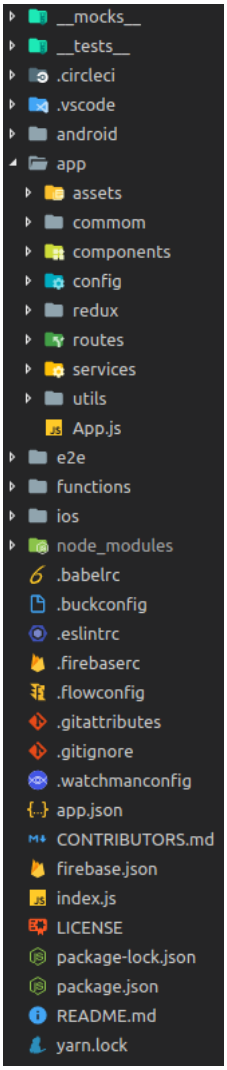

Figura 21 – Estrutura de arquivos do projeto. Fonte: Autores

No android a status bar não faz parte da aplicação, podendo apenas ser removida ou ter sua cor de fundo alterada. Já no iOS a status bar faz parte do aplicativo e precisa ser levada em consideração no design do aplicativo como é mostrado na Figura 22.

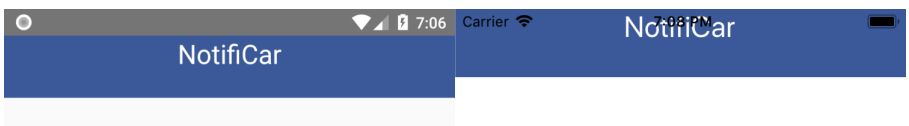

Figura 22 – Conteúdo do aplicativo sobreposto pela barra de status. Fonte: Autores

O react native fornece uma API que permite identificar as plataformas e implementar um código diferente para cada uma, este recurso está disponivel por meio do módulo ***Platform***, o método ***Platform.OS*** retorna o sistema em execução, e podemos diferenciar os códigos a partir daqui, um exemplo para resolver o caso da status bar citado acima está descrito abaixo.

```
1  import { Platform } from 'react-native'
2
3  const styles = {
4    statusbar = {
5      marginTop: Platform.OS === 'ios' ? 20 : 0
6    }
7  }
```

Aqui estamos criando um estilo chamado *statusBar* que contém o atributo paddingTop, e passamos como valor a margem interna que deve ser aplicada, para isso fazemos uso do componente Platform, e especificamos que caso a plataforma em execução seja ios, será aplicado um padding de 20 pixels, caso seja android aplica-se 0 de padding.

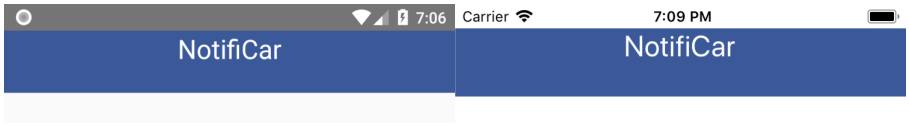

Figura 23 – Barra de status corrigida. Fonte: Autores

Este recurso pode ser utilizados não apenas para diferenciar estilos mas também para especificar comportamentos dentro do aplicativo com base na plataforma.

Outra grande diferença entre Android e iOS está no gerenciamento de dependências com implementação nativa. Diversas bibliotecas disponíveis para react native possuem implementação apenas em javascript e não é necessário realizar alterações na parte nativa do aplicativo, no entanto para algumas bibliotecas é necessários realizar a configuração da biblioteca na código nativo.

As IDES utilizadas, Xcode e Android Studio gerenciam de formas diferentes as dependencias de seus projetos, o XCode utiliza um sistema de frameworks e o Android Studio utiliza o gradle. Em geral, as bibliotecas disponibilizam de forma detalhada os passos para configuração da mesma. No entanto para uma biblioteca a configuração pode ser simples no iOS e complicada no Android e vice e versa, a depender dos recursos nativos utilizados.

Durante a realização deste projeto não foram encontrados grandes problemas para gerencias tais dependências, mas cabe citar que em geral a parte referente ao Android é mais fácil de ser realizada apesar do XCode permitir que praticamente tudo serja realizado de forma gráfica, o que foi considerado um ponto positivo pelos autores.

5 Considerações Finais

A partir do desenvolvimento deste trabalho, observamos diversas dificuldades, devido à definição do escopo de um projeto apenas pela sua ideia central. Alem disso, a opção pelo desenvolvimento, com uma tecnologia jamais utilizada pelos autores, ocasionou em relevante esforço de pesquisa para assim, garantir que os recursos necessários à aplicação fossem suportados. Adicionalmente, foram testados diversas ferramentas compatíveis com o *React Native* para garantir a aplicabilidade das boas práticas de engenharia de *software* a nível de desenvolvimento.

Com o objetivo do trabalho proposto, foi possível refletir a necessidade de desenvolver uma aplicação utilizando um *framework*, que possibilita gerar uma aplicação para diferentes tipos de plataformas, *iOS* e *Android*. Para isso, optou-se pelo *React Native*, por utilizar a linguagem de programação *JavaScript*, o que permite desenvolver aplicações mobile, como também, permite adquirir conhecimento necessário para o desenvolvimento de aplicação web.

Neste projeto aplicamos diversas práticas de engenharia de software, no qual foi observado todo o problema em questão, gerando requisitos para solucionar o problema, foram definidos prazos. Ocorreu a escolha da tecnologia que cumprisse, da melhor forma, os requisitos. Foi gerado a metodologia de desenvolvimento, posteriormente realizou-se o desenvolvimento, criou-se testes, aplicou-se uma gerência de configuração de software e ocorreu a implantação do software na loja.

Apesar de conseguirmos implementar um processo de entrega contínua para ambas as plataformas, o aplicativo para a plataforma *iOS* não foi aceito na *App Store* após passar pela análise dos avaliadores. De acordo com os avaliadores, a possibilidade de reportar informações a respeito de um veículo pertencente a um terceiro infringe a diretriz *Guideline 1.1.1 - Safety - Objectionable Content*, que se refere a conteúdo que pode ser considerado difamatório ou de mau gosto. Mesmo após entrar em contato para justificar o objetivo do aplicativo, a resposta de manteve negativa. Essa eventualidade nos mostrou que entender apenas da tecnologia não é suficiente, mas que também é imprescindível ter conhecimento de outros aspectos associados ao contexto do desenvolvimento de aplicativos, como as diretrizes publicação de um aplicativo.

Apesar das dificuldades, estamos gratos com o resultado alcançado, pois os objetivos dos autores foram alcançados, já que é uma aplicação voltada a sociedade, focada principalmente em ajudar os alunos e funcionários da Universidade da Brasília, como também é de grande relevância para o futuro profissional que certamente trará oportunidades e agregará o portfólio.

Referências

APPANNIE. **App Annie Index Market Q1 2016: China Takes Japan 2nd Spot for iOS Revenue**. 2016. <https://www.appannie.com/en/insights/market-data/app-annie-index-market-q1-2016/>. Acessado em: 03/10/2017. Citado na página 21.

APPLE INC. **Apple Developer**. 2017. <https://developer.apple.com>. Acessado em: 04/10/2017. Citado na página 21.

_____. **Human Interface Guidelines**. 2017. <https://developer.apple.com/design/>. Acessado em: 15/10/2017. Citado na página 45.

_____. **XCode**. 2017. <https://developer.apple.com/xcode/>. Acessado em: 04/10/2017. Citado na página 21.

BAKER, M. D. Implementing an initial software metrics program. In: **Proceedings of the IEEE 1991 National Aerospace and Electronics Conference NAECON 1991**. [S.l.: s.n.], 1991. p. 1289–1294 vol.3. Citado na página 25.

BAUER, F. **Software Engineering**. [S.l.], 1972. Citado na página 19.

CASS, S. **The 2017 Top Programming Languages**. 2017. <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>. Acessado em: 14/10/2017. Citado na página 35.

CHARLAND, A.; LEROUX, B. Mobile application development: Web vs. native. **Queue**, ACM, New York, NY, USA, v. 9, n. 4, p. 20:20–20:28, abr. 2011. ISSN 1542-7730. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/1966989.1968203>. Citado 2 vezes nas páginas 20 e 21.

CHEN, L. Continuous delivery: Huge benefits, but challenges too. **IEEE Software**, v. 32, n. 2, p. 50–54, Mar 2015. ISSN 0740-7459. Citado na página 26.

CHIN-YUN, H.; CHIEN-TSUN, C. **C. Patterns for Continuous Integration Builds in Cross-Platform Agile Software Development**. [S.l.], 2015. Citado na página 26.

DELEY, T.; GJORGJEVIKJ, D. Static analysis of source code written by novice programmers. In: **2017 IEEE Global Engineering Education Conference (EDUCON)**. [S.l.: s.n.], 2017. p. 825–830. Citado na página 44.

FACEBOOK INC. **Detox**. 2017. <https://github.com/wix/detox>. Acessado em: 15/10/2017. Citado na página 36.

_____. **Jest: Delightful JavaScript Testing**. 2017. <http://facebook.github.io/jest/>. Acessado em: 15/10/2017. Citado na página 35.

_____. **React Native**. 2017. <https://facebook.github.io/react-native/>. Acessado em: 14/10/2017. Citado na página 35.

FASTLANE. **Automation done right**. 2017. <https://fastlane.tools>. Acessado em: 15/10/2017. Citado na página 38.

FIREBASE. 2017. <https://firebase.google.com>. Acessado em: 15/10/2017. Citado na página 43.

GAFFNEY JR., J. E. Metrics in software quality assurance. In: **Proceedings of the ACM '81 Conference**. New York, NY, USA: ACM, 1981. (ACM '81), p. 126–130. ISBN 0-89791-049-4. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/800175.809854>. Citado na página 25.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4th. ed. [S.l.]: Atlas, 2002. ISBN 85-224-3169-8. Citado na página 16.

GITHUB INC. **About GitHub**. 2017. <https://github.com/about>. Acessado em: 15/10/2017. Citado na página 45.

_____. **The world's leading software development platform**. 2017. <https://github.com>. Acessado em: 15/10/2017. Citado na página 44.

GOOGLE INC. **Google Play Console**. 2017. <https://developer.android.com/distribute/index.html>. Acessado em: 04/10/2017. Citado na página 21.

_____. **Material Design**. 2017. <https://material.io/guidelines/>. Acessado em: 15/10/2017. Citado na página 45.

_____. **Meet Android Studio**. 2017. <https://developer.android.com/studio/intro/index.html>. Acessado em: 04/10/2017. Citado na página 21.

_____. **Serviços de back-end de aplicativos para dispositivos móveis**. 2017. <https://cloud.google.com/solutions/mobile/mobile-app-backend-services>. Acessado em: 25/10/2017. Citado 2 vezes nas páginas 9 e 33.

HUMBLE, J.; FARLEY, D. **Entrega Contínua - Como Entregar Software de Forma Rápida e Confiável**. 1st. ed. [S.l.]: Bookman, 2013. ISBN 9788582601044. Citado 2 vezes nas páginas 26 e 40.

IBGE. **Frota de veículos**. 2016. <http://cidades.ibge.gov.br/painel/frota.php>. Acessado em: 21/09/2017. Citado na página 14.

IDC. **Smartphone OS Market Share, 2017 Q1**. 2017. <https://www.idc.com/promo/smartphone-market-share/os>. Acessado em: 02/10/2017. Citado 2 vezes nas páginas 9 e 20.

ISO/IEC/IEEE. **Systems and software engineering**. [S.l.], 2010. Citado na página 19.

JENTSCH, C. S. The impact of agile practices on team interaction quality – insights into a longitudinal case study. 2017. Disponível em: <http://aisel.aisnet.org/amcis2017/SystemsAnalysis/Presentations/5/>. Citado na página 22.

KAN, S. H. **Metrics and Models in Software Quality Engineering**. 2nd. ed. [S.l.]: Addison-Wesley Professional, 2014. ISBN 0-201-72915-6. Citado na página 23.

KAUKKANEN, E. et al. Bottom-up adoption of continuous delivery in a stage-gate managed software organization. In: **Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: ACM, 2016. (ESEM '16), p. 45:1–45:10. ISBN 978-1-4503-4427-2. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/2961111.2962608>. Citado 2 vezes nas páginas 9 e 27.

KOBROSLY, W.; VASSILIADIS, S. A survey of software functional testing techniques. p. 127–134, Oct 1988. Citado na página 25.

LAVRIV, O.; BUHYL, B.; KLYMASH, M.; GRYNKEVYCH, G. Services continuous integration based on modern free infrastructure. p. 150–153, July 2017. Citado na página 26.

LING, Y.; XU, B. Role clarification in software unit testing for middle scale database interface components. p. 176–179, May 2009. Citado na página 25.

LTD, W. A. S. **Digital in 2017: Global Overview**. 2017. <https://wearesocial.com/special-reports/digital-in-2017-global-overview>. Acessado em: 03/10/2017. Citado na página 19.

LUIS VASCONCELLOS. **Apps Híbridas com Cordova e Ionic**. 2017. <http://goo.gl/J18H4N>. Acessado em: 04/10/2017. Citado na página 22.

MILLS, E. E. **Software Metrics**. [S.l.], 1988. Citado na página 26.

MINDMASTER. **Scrum: A Metodologia Ágil Explicada de forma Definitiva**. 2014. <http://www.mindmaster.com.br/scrum/>. Acessado em: 09/10/2017. Citado 2 vezes nas páginas 9 e 23.

MIRANDA, P. R. **Monitoramento de métricas de código-fonte em projetos de software livre**. [S.l.], 2013. Tese (Doutorado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013. doi:10.11606/T.45.2013.tde-27082013-090242. Acesso em: 07-09-2017. Citado na página 26.

PRESSMAN, R. S. **Software Engineering: a Practitioner's Approach**. 7th. ed. [S.l.]: The McGraw-Hill Companies, 2011. ISBN 0073375977/9780073375977. Citado na página 19.

RAHIMIAN, V.; RAMSIN, R. Designing an agile methodology for mobile software development: A hybrid method engineering approach. In: **2008 Second International Conference on Research Challenges in Information Science**. [S.l.: s.n.], 2008. p. 337–342. ISSN 2151-1349. Citado na página 20.

SCRUM: metodologia ágil para gestão e planejamento de projetos. 2014. <http://www.desenvolvimentoagil.com.br/scrum/>. Acessado em: 27/09/2017. Citado na página 23.

SERRANO, N.; HERNANTES, J.; GALLARDO, G. Mobile web apps. **IEEE Software**, v. 30, n. 5, p. 22–27, Sept 2013. ISSN 0740-7459. Citado 3 vezes nas páginas 10, 20 e 21.

SINGH, G. Metrics for measuring the quality of object-oriented software. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 38, n. 5, p. 1–5, ago. 2013. ISSN 0163-5948. Disponível em: <http://doi.acm.org.ez54.periodicos.capes.gov.br/10.1145/2507288.2507311>. Citado na página 25.

SOMMERVILLE, I. **Software Engineering**. 9th. ed. [S.l.]: Pearson Education, 2011. ISBN 978-85-7936-108-1. Citado na página 24.

TRAVISCI. **Test and Deploy with Confidence**. 2017. <https://travis-ci.org>. Acessado em: 15/10/2017. Citado na página 38.

Anexos

ANEXO A – Arquivo de configuração do CircleCI

```
version: 2
jobs:
  build:
    working_directory: ~/project
    docker:
      - image: circleci/node:8
    steps:
      - checkout
      - run: npm install
      - run: npm run test
      - persist_to_workspace:
          root: ~/project
          paths:
            - node_modules
      - store_test_results:
          path: ~/project/junit.xml

  android:
    working_directory: ~/project/android
    docker:
      - image: circleci/android:api-27-node8-alpha
    steps:
      - checkout:
          path: ~/project
      - attach_workspace:
          at: ~/project
      - run:
          name: Configure Android Secret Files
          command: bash ../.circleci/android.sh
      - run: bundle install
      - run: bundle exec fastlane test
      - run: bundle exec fastlane playstore
      - store_test_results:
          path: ~/project/android/reports

  ios:
    macos:
      xcode: "9.0"
    working_directory: /Users/distiller/project/ios
    environment:
      FL_OUTPUT_DIR: /Users/distiller/project/output
```

```
42      shell: /bin/bash --login -o pipefail
43      steps:
44        - checkout:
45            path: /Users/distiller/project
46        - run:
47            name: Set Ruby Version
48            command: echo "ruby-2.4" > ~/.ruby-version
49        - run: npm install
50        - run: bundle install
51        - run: bundle exec fastlane test
52        - run:
53            command: cp $FL_OUTPUT_DIR/scan/report.junit $FL_OUTPUT_DIR/scan/results.xml
54            when: always
55        - store_artifacts:
56            path: /Users/distiller/project/output
57        - store_test_results:
58            path: /Users/distiller/project/output/scan
59
60  workflows:
61    version: 2
62    node-android-ios:
63      jobs:
64        - build:
65            filters:
66              branches:
67                only:
68                  - master
69        - android:
70            requires:
71              - build
72        - ios:
73            requires:
74              - build
```