UNIVERSIDADE FEDERAL FLUMINENSE
CAMPUS UNIVERSITÁRIO DE RIO DAS OSTRAS
INSTITUTO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Muniz Lopes
Matheus Lima Marins

Aplicação para o gerenciamento de caronas da UFF

Rio das Ostras – RJ
2018

Aplicação para o gerenciamento de caronas da UFF

Monografia apresentada ao curso de Bacharelado em Ciência da Computação da Universidade Federal Fluminense – Rio das Ostras, como pré-requisito para a obtenção do Grau de Bacharel.

_____

Professor orientador:
CARLOS BAZILIO MARTINS

RIO DAS OSTRAS - RJ
2018

MATHEUS LIMA MARINS
LUCAS MUNIZ LOPES

Aplicação para o gerenciamento de caronas da UFF

Monografia apresentada ao curso de Bacharelado em Ciência da Computação da Universidade Federal Fluminense – Rio das Ostras, como pré-requisito para a obtenção do Grau de Bacharel.

Aprovado em ..... de ..... de 2018

BANCA EXAMINADORA

Prof. DSc. Carlos Bazilio Martins – Orientador
Universidade Federal Fluminense

Prof. MSc. Eduardo Marques - Avaliador
Universidade Federal Fluminense

Prof. DSc. Dalessandro Soares Vianna - Avaliador
Universidade Federal Fluminense

A todos que acreditaram no nosso potencial e nos deram força para nunca desistir.

AGRADECIMENTOS

Agradecemos em primeiro lugar a Deus por ter nos concedido a vida para chegar até esse momento.

Após isso, nossas famílias por todo suporte concedido para concluir mais essa etapa em nossas vidas. Por todas as vezes que pensamos que não seria possível, mas nos mantiveram firmes para caminhar.

Somos gratos também ao nosso orientador Carlos Bazilio, que aceitou nos ajudar e dar o apoio necessário para que tomássemos as melhores decisões na elaboração deste projeto.

Aos amigos Leonardo Muniz, Roney Aguiar, Marcello Câmara, José Morista e muitos outros pelo apoio diário em nossas jornadas, pelos momentos únicos proporcionados, o nosso muito obrigado.

Aos docentes que sempre, de alguma forma, colaboraram para nos tornarmos o que somos hoje.

Agradecer a Universidade Federal Fluminense por nos gerar a oportunidade de crescer profissionalmente e pessoalmente.

A todos os familiares que contribuíram de alguma forma.

SUMÁRIO

RESUMO

ABSTRACT

LISTA DE ABREVIATURAS

LISTA DE TABELAS

LISTA DE FIGURAS

1 INTRODUÇÃO 13

1.1 OBJETIVO 13
1.2 MOTIVAÇÃO 14
1.3 ESTRUTURA 14

2 REFERENCIAL TEÓRICO 15

2.1 PLATAFORMA MÓVEL 15
2.1.1 Android 15
2.1.1.1 Activity 17
2.1.1.2 View 18
2.1.1.3 Fragmentos 19
2.2 DESENVOLVIMENTO MOBILE 20
2.2.1 Aplicativo Nativo 20
2.2.2 Aplicativo Web 21
2.2.3 Aplicativo Híbrido 21
2.3 COMPARAÇÃO DOS DIFERENTES TIPOS DE DESENVOLVIMENTO 22
2.3.1 Recursos do dispositivo 22
2.3.2 Desempenho 23
2.3.3 Custo de desenvolvimento 23
2.3.4 Interface do usuário 24
2.3.5 Independência de plataforma 25

3 TRABALHOS RELACIONADOS 26

3.1 ZUMPY 26
3.2 BLABLACAR 27
3.3 KARONAS 27

3.4 CARONA PHONE 28
4 PROJETO DO CARONASUFF 29
4.1 MODELAGEM DO SISTEMA 29
4.1.1 Requisitos 29
4.1.1.1 Requisitos funcionais 29
4.1.1.2 Requisitos não funcionais 31
4.1.2 Casos de uso 33
4.1.2.1 Descrição dos casos de uso 34
4.1.3 Banco de Dados 41
4.1.3.1 Sobre banco de dados 41
4.1.3.3 Estrutura do banco de dados 42
5 DESENVOLVIMENTO 45
5.1 TECNOLOGIAS 45
5.1.1 Java 45
5.1.2 PHP 46
5.1.2.1 Web service 47
5.1.3 MySQL 48
5.1.4 Firebase 49
5.1.5 Android Studio 52
5.2 IMPLEMENTAÇÃO 54
5.2.1 Configurações 54
5.2.1.1 Android Manifest 54
5.2.1.2 Bibliotecas externas 55
5.2.2 Classes 56
5.3 FLUXO DE TELAS 68
5.3.1 Criar Cadastro 68
5.3.2 Pedir e oferecer carona 70
5.3.3 Perfil do usuário 72
5.4 TESTES 73
6 CONCLUSÃO 78
6.1 TRABALHOS FUTUROS 78
REFERÊNCIAS BIBLIOGRÁFICAS 80

LISTA DE ABREVIATURAS

UFF	Universidade Federal Fluminense
OHA	*Open Handset Alliance*
JVM	*Java Virtual Machine*
ART	*Android Run Time*
IDC	*International Data Corporation*
PWA	*Progressive Web App*
GPS	*Global Positioning System*
SGBD	Sistema Gerenciador de Banco de Dados
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
PHP	*Hypertext Preprocessor*
SQL	*Structured Query Language*
CGI	*Computer-Generated Imagery*
PDO	*PHP Data Objects*
SOAP	*Simple Object Access Protocol*
REST	*Representational State Transfer*
XAMPP	Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P)
API	*Application Programming Interface*
SSL	*Secure Sockets Layer*
CSS	*Cascading Style Sheets*
UIL	*Universal Image Loader*
URL	*Uniform Resource Locator*
MIME	*Multipurpose Internet Mail Extensions*

LISTA DE TABELAS

Tabela 1 - Mercado Mundial de Smartphones por Sistema Operacional 17
Tabela 2 - Comparação entre os aplicativos analisados 28
Tabela 3 - Requisitos funcionais do projeto 30
Tabela 4 - Requisitos não funcionais do projeto.....33
Tabela 5 - Perguntas feitas ao final do teste 74
Tabela 6 - Resposta usuário 1 75
Tabela 7 - Resposta usuário 2 75
Tabela 8 - Resposta usuário 3 76
Tabela 9 - Resposta usuário 4 76

LISTA DE FIGURAS

Figura 1 - Ciclo de vida de uma activity....18
Figura 2 - Utilização de fragmentos....19
Figura 3 - Diagrama de casos de uso....40
Figura 4 - Modelo do banco de dados....44
Figura 5 - Serviços x plataformas firebase....50
Figura 6 - Serviços do firebase....52
Figura 7 - Estrutura do projeto....53
Figura 8 - Permissões de uso Android Manifest....54
Figura 9 - Método loggedin....57
Figura 10 - Código para envio de e-mail....58
Figura 11 - Método para pesquisa....60
Figura 12 - Menu Lateral....61
Figura 13 - ViewPager com TabLayout....62
Figura 14 - TimePicker....64
Figura 15 - DatePicker. 64
Figura 16 - Método onCreateDialog....65
Figura 17 - Método onScrolled....67
Figura 18 - Tela inicial....68
Figura 19 - Tela solicitação código....69
Figura 20 - Tela cadastro usuário....70
Figura 21 - Tela pedir carona....71
Figura 22 - Tela oferecer carona....72
Figura 23 - Tela perfil usuário....73

RESUMO

O Caronas UFF é um aplicativo para dispositivos móveis que permite requisitar, aceitar caronas e avaliar usuários da mesma por parte da comunidade UFF. A necessidade de criar um aplicativo móvel desse gênero se deu pelo fato de que atualmente a área de caronas compartilhadas vem crescendo de forma rápida com o passar do tempo, porém não há nada no âmbito mobile que forneça suporte com opções exclusivas para a comunidade. A aplicação tem como objetivo unir diversos grupos de caronas compartilhadas localizados em diferentes redes sociais em um aplicativo exclusivo para a UFF, diminuindo a insegurança dos usuários no que diz respeito a conhecer com quem está dividindo uma carona. Para tal, é necessário o engajamento da comunidade, de modo que os integrantes envolvidos em uma carona se avaliem depois de realizada a mesma, ou até mesmo em casos onde usuários não cumpram o combinado, gerando assim, uma reputação dos usuários no sistema.

Palavras-chave: caronas, mobile.

ABSTRACT

Caronas UFF is a mobile application that allows you to request, accept rides and evaluate users of it by the UFF community. The need to create a mobile application of this kind was due to the fact that currently the area of shared carpool has been growing rapidly over time, but there is nothing in the mobile scope that provides support with options unique to the community. The application aims to join several groups of shared carpool located in different social networks in a unique application for the UFF, reducing the insecurity of the users with respect to knowing who is sharing a ride. To do this, community engagement is necessary, so that the members involved in a ride are evaluated after performing the same, or even in cases where users do not comply, thus generating a reputation of users in the system.

Keywords: rides, mobile.

1 INTRODUÇÃO

É fato que os dispositivos móveis estão cada vez mais presentes na vida da população. Atualmente compra-se mais smartphones do que computadores, dando força de que tais aparelhos tendem a crescer muito mais englobando sempre um número maior de usuários. Segundo dados da IDC [1], mesmo com queda na venda de smartphones em 2017 e 2018, em 2019 deve-se retomar o crescimento nas vendas. Fator de contribuição para tal retomada se deve a entrada de celulares 5G no segundo semestre de 2019.

Smartphones são computadores de bolso, com sensores e capacidade de processamento alta. Para esse tipo de tecnologia é possível criar ferramentas robustas para oferecer serviços compartilhados como caronas, por exemplo, na qual quem pede e quem oferece caronas são os principais beneficiários deste serviço.

Projetos coletivos no que se diz respeito a consumir bens e serviços estão trazendo uma nova perspectiva para o mercado. “O modelo de consumo colaborativo ganhou novas dimensões após a crise financeira dos EUA em 2008 e que também atingiu outros países pelo mundo” [2]. Ideias colaborativas atingem diferentes áreas, algumas sem a ambição de lucro, com o intuito apenas de dividir gastos como caronas compartilhadas e outras que visam o retorno monetário como *crowdfunding* (financiamento colaborativo) e *coworking* (compartilhamento de espaço de trabalho). A UFF é uma universidade bastante interiorizada, tal fato exige um deslocamento constante de alunos entre cidades. Esse volume de deslocamento gera um alto custo aos universitários, custo esse que é diminuído com as caronas compartilhadas. Diante disso, a fim de oferecer uma opção a mais e exclusiva, sugere-se no presente trabalho o desenvolvimento de um sistema móvel para o gerenciamento de caronas da comunidade UFF.

1.1 OBJETIVO

O presente projeto tem como objetivo trazer uma aplicação Android aos diferentes pólos da Universidade Federal Fluminense. O aplicativo propõe-se a

gerenciar caronas oferecidas e pedidas por usuários, oferecendo também algumas opções que vão além da funcionalidade básica, como, por exemplo, uma funcionalidade de avaliação mutua dos participantes da carona ao final da mesma.

1.2 MOTIVAÇÃO

A motivação para tal projeto se dá pelo fato de que os maiores aplicativos deste segmento oferecem muitas opções que, em alguns casos, não são necessárias para um universitário. Levando em conta o fato de que o usuário pode usar quantos aplicativos de caronas compartilhadas quiser, o presente trabalho oferece uma opção a mais, porém com mais segurança e funcionalidades únicas voltadas para usuários da UFF.

1.3 ESTRUTURA

O trabalho apresenta a seguinte estrutura:

Referencial Teórico: serão apresentados aspectos gerais de uma plataforma móvel, a plataforma Android em específico, os tipos de aplicativos e uma comparação entre eles em diversos quesitos como, por exemplo, desempenho e custo de desenvolvimento.

Trabalhos relacionados: busca fazer uma análise de alguns dos maiores aplicativos deste tipo de mercado, mostrando aspectos considerados positivos e negativos em todos eles.

Projeto do CaronasUFF: neste capítulo é mostrada a modelagem do sistema como um todo. Mostra-se o que é engenharia de requisitos, detalha-se os requisitos funcionais e não-funcionais do sistema. Apresenta também os casos de uso e o banco de dados.

Implementação: como a modelagem é traduzida em código, mostrando bibliotecas utilizadas e comunicação com o banco de dados por exemplo. Aqui também são abordados testes feitos por terceiros na aplicação e o fluxo de telas das principais funcionalidades do sistema.

Conclusão: neste capítulo o foco é mostrar resultados obtidos juntamente com possíveis trabalhos futuros a serem elaborados.

2 REFERENCIAL TEÓRICO

Neste capítulo vamos abordar sobre o que é uma plataforma móvel e sobre uma especificação da mesma que é a plataforma Android. Serão apresentados os principais modos de desenvolvimento e uma comparação entre esses meios, considerando desempenho, custo e interface do usuário.

2.1 PLATAFORMA MÓVEL

A principal função de uma plataforma móvel é garantir o acesso ao dispositivo. Para rodar softwares e serviços em qualquer dispositivo, é necessária uma plataforma, ou uma linguagem de programação básica em que todo o software está escrito [3]. Os dispositivos móveis principalmente os celulares deixaram de ser apenas celulares e se tornaram computadores, com diversos tipos de serviços que podem ser realizados facilmente a qualquer hora e lugar. Sendo assim, será analisada a plataforma Android, que foi a escolhida para a elaboração do aplicativo.

2.1.1 Android

A plataforma Android teve seu início em outubro de 2003, quando Andy Rubin, Rich Miner, Nick Sears e Chris White fundaram a Android Inc. A empresa desenvolvia sistemas operacionais para celulares de forma secreta. Em 2005 a Google anunciou a compra da Android Inc, dando assim, seu primeiro passo na área de dispositivos móveis. No entanto, o sucesso do Android não se deve apenas à força do Google. Por trás do desenvolvimento de toda a plataforma estão gigantes do mercado de mobilidade, como fabricantes de celulares e operadoras. “Esse grupo que ajuda no desenvolvimento da plataforma é chamado de OHA (*Open Handset Alliance*) e conta com nomes de peso como Intel, Samsung, LG, Motorola, Sony Ericsson, HTC, Sprint Nextel, ASUS, Acer, Dell, Garmin etc.” [4].

“O sistema operacional Android é baseado no kernel do Linux, que é responsável por gerenciar a memória, os processos, threads, segurança dos arquivos e pastas, além de redes e drivers” [4]. Por ser uma plataforma de código aberto (open source), o Android possibilita uma série de customizações e personalizações. Essa vantagem também se aplica aos fabricantes de celulares, visto que é possível utilizar o sistema operacional Android em seus aparelhos sem ter de pagar por isso.

A linguagem Java é a utilizada para construir aplicações para Android. A questão é que o sistema operacional não possui uma máquina virtual Java(JVM), mas sim uma máquina virtual chamada Dalvik que é otimizada para execução em dispositivos móveis [4]. A partir do Android 4.4 (Kit Kat), o Dalvik foi deixado de lado para a entrada da máquina virtual ART (Android Runtime), sendo ativado opcionalmente nas configurações. Quando o Android 5.0 (Lollipop) foi lançado, o ART se tornou a máquina virtual padrão, substituindo o Dalvik.

Atualmente o Android é o sistema operacional mais utilizado no Brasil e no mundo. De acordo com uma pesquisa realizada em 2017 pela empresa americana IDC (International Data Corporation), o Android detém quase 85% dos usuários de smartphones e a tendência é de crescimento ao longo dos próximos 4 anos. A tabela 1 ilustra essa pesquisa. Além disso, segundo [5], 96,5% dos usuários *Android* utilizam a versão 4.4 ou maior.

Devido à popularidade do Android, buscando atender uma maior parte de usuários dentro do público alvo (discentes e docentes da UFF), e a disponibilidade dessa tecnologia para o desenvolvimento do projeto, esse foi o sistema operacional utilizado para o projeto.

Tabela 1 - Mercado Mundial de Smartphones por Sistema Operacional

Elementos Analisados	Android	iOS	Others
2016	84.6%	14,7%	0.7%
2017	85.1%	14,7%	0.2%
2018	84,8%	15,1%	0.1%
2019	85,2%	14,8%	0,1%
2020	85,3%	14,6%	0,1%
2021	85,4%	14.6%	0,1%
2022	85,5%	14.5%	0,1%

Fonte: International Data Corporation (IDC)[1].

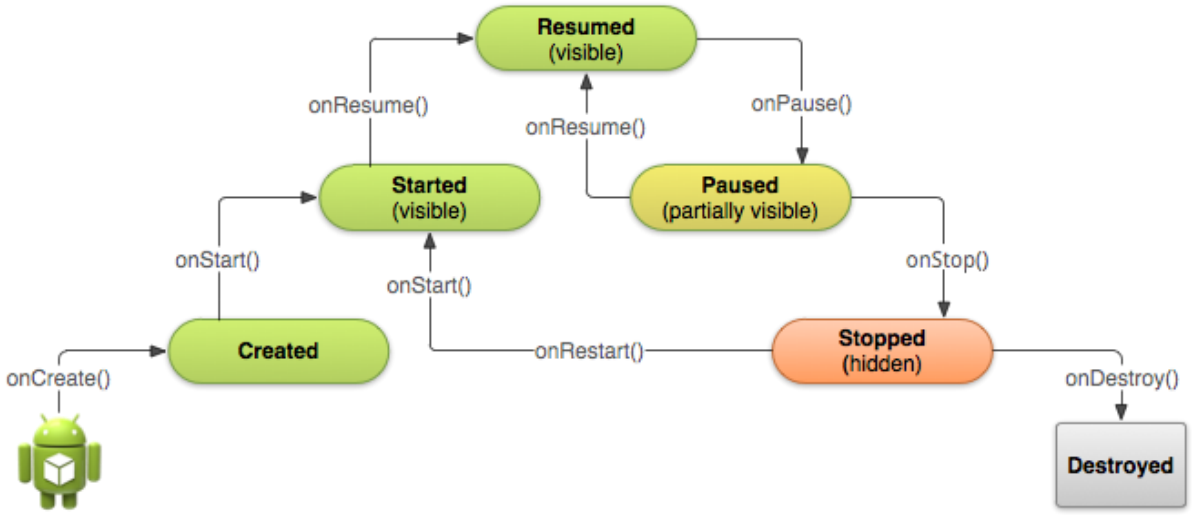
2.1.1.1 Activity

Um conceito fundamental no desenvolvimento Mobile é sobre activity, que é uma classe com um papel fundamental no ciclo de vida de um projeto Android. A activity é responsável por representar uma tela da aplicação, capturar os eventos que ocorrem e qual View que será a responsável por apresentar a parte gráfica ao usuário.

O ciclo de vida de uma Activity é controlado pelo sistema operacional. Porém existem alguns detalhes que devem ser considerados durante o desenvolvimento como mostra a figura 1.

[1] https://www.idc.com/promo/smartphone-market-share/os

Figura 1 - Ciclo de vida de uma activity



Fonte: Google Android (4ª edição).

- Created: nesta etapa é realizada a inicialização da aplicação carregando os componentes necessários para executar a aplicação. Em todo o ciclo de vida da Activity é passado apenas uma vez nesse estado.
- Started: este estado começa quando a aplicação começa a se tornar visível ao usuário, quando uma View está pronta para ser exibida.
- Resumed: quando de fato a View se torna visível ao usuário, podendo receber interações e atualizar dados na tela.
- Paused: esse estado é iniciado quando o usuário sai da activity, pressionando o botão home ou voltar do dispositivo.
- Stopped: neste estado a Activity deixa de ser visível ao usuário, ficando em segundo plano. Caso o sistema operacional precise liberar memória ela pode ir para o estado Destroyed onde será encerrada por completo.
- Destroyed: quando uma aplicação é finalizada ela passa por esse estado, onde os recursos são liberados da memória.

2.1.1.2 View

"A classe View é a classe-mãe de todos os componentes visuais do Android" [4]. Através dessa definição é possível perceber a papel da classe View em uma

aplicação Android. Todos os elementos que são apresentados para o usuário, a view é responsável por exibir e capturar as interações que ocorram.

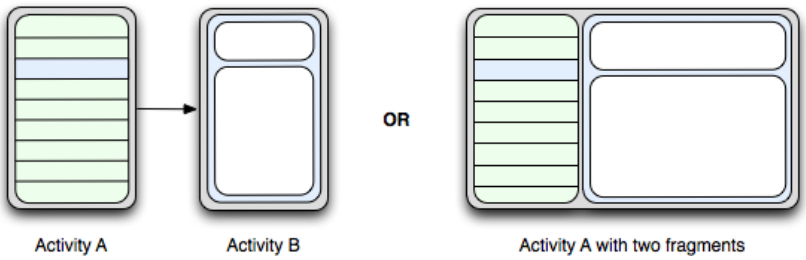
2.1.1.3 Fragmentos

Um componente fundamental presente no *Android* são fragmentos. “Um fragmento é um componente de código reutilizável, responsável por criar a sua própria *view*, tratar os eventos e gerenciar o seu próprio conteúdo. ” [4]

Com a utilização dos fragmentos é possível deixar o aplicativo mais limpo e leve, pois apenas um componente tem a responsabilidade de controlar qual *view* será exibida ao usuário e gerenciar os eventos que nela ocorrerem. ”

Como podemos observar na figura 2, no primeiro caso não é utilizado o uso de fragmento, quando uma opção da View A é escolhida uma nova *Activity* é chamada. Já no segundo exemplo é possível observar que o conteúdo da opção selecionada é exibido na mesma Activity, o que torna a aplicação mais leve e com melhor usabilidade por parte do usuário.

Figura 2 - Utilização de fragmentos



Fonte: Google Android 4ª edição.

2.2 DESENVOLVIMENTO MOBILE

O desenvolvimento mobile possui três caminhos que podem ser seguidos que são eles:

1. Desenvolvimento de aplicativo Nativo: Quando o programa final vai ser executado em algum dispositivo móvel.
2. Desenvolvimento de aplicativo Web: Quando o programa final vai ser executado em algum browser.
3. Desenvolvimento de aplicativo Híbrido: É parte nativo, parte web. Possui a vantagem de utilizar recursos tanto do dispositivo móvel quanto do browser.

2.2.1 Aplicativo Nativo

Atualmente é o modo de desenvolvimento mais utilizado devido a sua facilidade e capacidade de utilizar os recursos disponíveis no dispositivo. Nesse tipo de desenvolvimento o aplicativo é feito especificamente para um tipo de plataforma, podendo utilizar de bibliotecas da linguagem nativa e SDK.

Como os aplicativos utilizam a linguagem nativa eles utilizam todas as funcionalidades e recursos disponíveis no aparelho, tais como câmera, sensores e GPS. Apresentam um melhor desempenho quando comparado aos outros modos de desenvolvimento como o *web* e o híbrido.

Devido ao fato de um aplicativo nativo ter acesso a todas as informações do aparelho, as empresas que fazem a distribuição desses aplicativos fazem uma verificação de integridade, evitando que seja feita a distribuição de aplicativos maliciosos.

2.2.2 Aplicativo Web

Neste modelo de desenvolvimento a aplicação roda em um navegador, que pode ser do próprio dispositivo móvel ou de um computador. É capaz de utilizar grande parte dos recursos presentes nos aparelhos como os aplicativos nativos e, devido ao fato de utilizar HTML5, acaba sendo multiplataforma, visto que a maioria dos dispositivos são capazes de renderizar em um *browser*.

Um termo que ganha força ao passar dos anos na área de aplicações web é o PWA (*Progressive Web App*). Ao contrário dos tradicionais aplicativos web, um *Progessive Web App* pode ser visto como uma junção entre aplicativos mobile e páginas web. Se inicia como uma simples aba web, vai se tornando progressivamente um app ao ponto de adicioná-lo a sua tela inicial e seu comportamento passa a ser igual de um app nativo com algumas funções que eram, até então, exclusivas do mesmo como geolocalização, notificações etc.

Aplicativos web progressivos possuem diversas vantagens técnicas comparados a aplicativos nativos e web, sendo algumas delas: Independência de conexão, responsividade, não utiliza memória do aparelho entre outras.

2.2.3 Aplicativo Híbrido

Aplicativos híbridos, diferentemente de aplicativos nativos, não são desenvolvidos completamente na linguagem do sistema operacional. Como aplicativos nativos, devem ser baixados em uma loja. Como aplicativos web, dependem de um HTML sendo renderizado em um navegador podendo, por exemplo, integrar informações do site para o app.

São populares pelo fato de que permitem o desenvolvimento multiplataforma, utilizando o mesmo código para diferentes sistemas operacionais. Por ser um código híbrido, utiliza-se pouca implementação nativa nesse tipo de aplicação, o que favorece caso queira manter a estrutura do aplicativo em outra plataforma.

Existem diversos frameworks específicos para desenvolvimento de aplicativos híbridos como, por exemplo, PhoneGap, Titanium e Sencha Touch. Essas plataformas

fazem uma espécie de interligação entre chamadas JavaScript e o código nativo do sistema pretendido. Uma empresa bastante conhecida que fez seus aplicativos baseada no conceito de aplicação híbrida foi a Netflix, que utiliza a mesma interface para todos os dispositivos, fazendo apenas alguns ajustes de acordo com testes de usabilidade. O serviço de *streaming* foi todo inserido em código nativo, para alcançar um melhor desempenho e satisfação do usuário.

2.3 COMPARAÇÃO DOS DIFERENTES TIPOS DE DESENVOLVIMENTO

Nesta subseção será comparado os três tipos de aplicativos citados anteriormente de modo que sejam avaliados em diversos aspectos que vão desde custo de desenvolvimento até interface do usuário. A comparação usará alguns dos critérios utilizados e analisados por [6] .

2.3.1 Recursos do dispositivo

Este tópico trata de recursos que os diferentes tipos de aplicativos podem acessar, entre eles GPS, giroscópio, câmera e envio de e-mails.

- **Nativo:** Possui acesso à totais recursos disponíveis.

- **Web:** Possui acesso limitado a alguns recursos, como GPS e envio de chamadas por exemplo.

- **Híbrido:** Por ser uma mescla de aplicativo nativo com aplicativo web, seus componentes implementados de forma nativa possuem acesso total aos recursos disponíveis.

2.3.2 Desempenho

- **Nativo:** Aplicativos Nativos sempre possuem o melhor desempenho, pois são desenvolvidos especificamente para a plataforma. Em 2012, Mark Zuckerberg declarou que ter apostado em uma aplicação web foi o maior erro do Facebook[2]. Até essa época, era um aplicativo com núcleo em HTML onde, no mesmo ano, foi substituído por uma aplicação inteiramente nativa.

- **Web:** Baixo desempenho, visto que na maioria dos aplicativos web é necessária conexão constante com a internet com o objetivo de buscar as informações.

- **Híbrido:** Baixo desempenho, além do fato de que grande parte da aplicação é projetada em HTML e não em forma nativa, existe também o aspecto que os navegadores não acompanham o avanço de hardware dos dispositivos em si, comprometendo o desempenho.

2.3.3 Custo de desenvolvimento

Trata-se do quão custoso se torna desenvolver uma aplicação para cada tipo, seja o custo financeiro seja o custo intelectual.

- **Nativo:** Na maioria das vezes, aplicativos nativos são mais custosos do ponto de vista de desenvolvimento, devido ao fato que necessitam de um conhecimento específico para a sua criação. Além disso, grandes empresas quando lançam um aplicativo móvel costumam lançar nas três maiores plataformas, são elas: Android, iOS, Windows Phone. Como já citado, aplicativos nativos utilizam de linguagem específica para cada sistema

[2] https://www.terra.com.br/noticias/tecnologia/internet/nosso-maior-erro-foi-apostar-em-html5-diz-zuckerberg,4b98201fd70ea310VgnCLD200000bbcceb0aRCRD.html

operacional, implicando que, para cada SO que se deseja lançar uma aplicação, será necessário um projeto quase que novo aumentando o custo do projeto final.

- **Web:** Custo reduzido, visto que utiliza tecnologias mais difundidas como HTML, JavaScript e CSS. Pelo fato do seu desenvolvimento ser genérico influencia na redução do custo.

- **Híbrido:** Seu custo normalmente é entre nativo e web, dependendo de quantos recursos tanto web quanto nativo o mesmo utiliza. Aplicações nativas exigem um conhecimento especializado, o HTML5 é novo. Logo quem consegue compreender por completo a ferramenta e consegue extrair o seu máximo para o desenvolvimento de aplicativos, têm seu custo mais elevado comparado a aplicativos web e híbridos abordados de forma mais rasa na área web.

2.3.4 Interface do usuário

A experiência do usuário é prioridade para uma aplicação móvel. Coisas como usabilidade e engajamento tem forte influência em quão bom será o aplicativo. A interface de usuário tem grande parcela de responsabilidade nesses aspectos.

- **Nativo:** É o mais recomendado no quesito interface do usuário. Seus gráficos e visuais já são previamente conhecidos pelo usuário, o que facilita em uma boa experiência do usuário.

- **Web:** Possui certa dificuldade de fornecer uma boa experiência ao usuário, pelo fato de ser o contrário de um aplicativo nativo e que será mais difícil aproveitar as ferramentas fornecidas a fim de deixar mais suave as limitações que tal possui.

- **Híbrido:** Muito semelhante ao aplicativo web.

2.3.5 Independência de plataforma

Independência de plataforma é o quão fácil é a utilização de uma aplicação em diversas plataformas.

- **Nativo:** Por ser um tipo de aplicativo atrelado a uma plataforma específica, não possui independência de plataforma.

- **Web:** É o mais amplo no que diz respeito à portabilidade. Como é acessado pelo browser, atinge grande variedade de plataformas.

- **Híbrido:** Pelo fato de que possui uma parte de código nativa, esta parte deve ser reescrita para plataformas diferentes.

Analisando os tipos de aplicações mencionadas anteriormente, chegou-se à conclusão de que uma aplicação nativa seria a melhor escolha. Questões como desempenho, interface do usuário e recursos do dispositivo tiveram prioridade maior que independência de plataforma e custo de desenvolvimento. Uma das ideias centrais era a construção de uma aplicação simples e com alto desempenho que atendesse a maior parcela de usuários da comunidade UFF. No início do presente projeto descartou-se a possibilidade da construção de uma aplicação multiplataforma por questões de prazo e tecnologias.

3 TRABALHOS RELACIONADOS

Nesta Seção serão apresentados alguns aplicativos que possuem objetivo semelhante ao presente trabalho. Apesar dos aplicativos citados a seguir possuírem mais funcionalidades, os mesmos exigem uma grande quantidade de configurações para um total aproveitamento do sistema, diferentemente do trabalho proposto, que possui foco nas funcionalidades primárias de um sistema de caronas colaborativas. Ao final da descrição dos produtos analisados, a tabela 2 trará comparações de cada um deles.

3.1 ZUMPY

O aplicativo Zumpy, fornece uma aplicação multiplataforma com opções de *download* tanto para Android, quanto para iOS. Esse aplicativo vai muito além de um simples sistema de oferecer e pedir caronas, possuindo, por exemplo, uma parceria com postos de gasolina com a finalidade do motorista ter a opção de usar seus créditos na aplicação para abastecer o automóvel. Outra funcionalidade que se estende ao funcionamento básico desse tipo de aplicação é o pagamento no próprio aplicativo, dando a possibilidade ao usuário de pagar com o cartão. Zumpy é a única aplicação dentre as analisadas que possui este tipo de recurso, podendo ser visto na tabela 2. Mais informações podem ser encontradas no *website*[3] do desenvolvedor.

Zumpy possui um sistema de avaliação de usuários semelhante ao proposto. Porém não há nada dedicado nesse aspecto nem mesmo um indicativo na tela das caronas mostrando a qualificação tanto de quem oferece quanto de quem pede uma carona.

Esse aplicativo tenta implementar o conceito de rede social de forma mais rasa. Os usuários podem adicionar amigos e ainda restringir a visualização das caronas oferecidas para amigos e até amigos de amigos. No presente trabalho isso não se aplica pois se trata de um ambiente totalmente universitário onde as caronas possibilitam uma ajuda de custo mútua. Devido a isso a maior quantidade de pessoas

[3] http://www.zumpy.com.br/

alcançadas tanto ao oferecer quanto ao pedir uma carona é melhor para o usuário. Restringir o alcance poderia afetar diretamente na experiência de alguns usuários.

3.2 BLABLACAR

BlaBlaCar é a aplicação mais popular deste segmento, como apresentado na tabela 2. O mesmo está presente em mais de 20 países, sendo um aplicativo para Android e iOS. Mais informações no *website*[4] da empresa.

O aplicativo tem seu foco maior na funcionalidade básica em si. Enquanto o Zumpy oferece opções que vão além da funcionalidade padrão, o BlaBlaCar se atrela apenas a oferecer e pedir caronas, porém com mais recursos e funcionalidades que o citado anteriormente. Um usuário pode, por exemplo, inserir sua habilitação de motorista ao perfil dando mais confiança a quem participa da carona, ou até mesmo especificar se animais e fumar dentro do carro são permitidos ou não.

A simplicidade ao procurar uma carona torna a busca pouco flexível. Não existe um layout próprio mostrando as caronas pedidas. Sempre que um usuário desejar visualizar caronas pedidas é necessária uma pesquisa, o que torna a ação um tanto quanto manual e repetitiva.

3.3 KARONAS

O Karonas é uma aplicação com opção para download tanto para Android quanto para iOS. Sua tela inicial é muito intuitiva, contendo apenas as opções de oferecer e pedir uma carona.

O fato do usuário ter que cadastrar um carro para poder oferecer uma carona limita a ação do mesmo. Sua usabilidade não é muito boa, visto que alguns mecanismos de buscas não ficam suficientemente claros ao usuário. Por exemplo, diferentemente do horário que está explícito ao procurar uma carona, o destino não é exibido de forma clara ao usuário dando a impressão de limitação da aplicação.

O aplicativo possui um sistema de favoritos bastante simples e direto, o que facilita o gerenciamento de caronas futuras sem precisar de um mecanismo mais

[4] https://www.blablacar.com.br/

complexo como sistema de amizade e grupos por exemplo. Mais informações podem ser obtidas em seu *website*[5].

3.4 CARONA PHONE

Diferentemente dos aplicativos citados anteriormente, o Carona Phone é uma aplicação voltada apenas para o sistema operacional Android.

Ao abrir o aplicativo pela primeira vez, é encontrada a primeira limitação ao usuário. O cadastro pode ser feito apenas via Facebook. Apesar de simples, requer uma série de configurações para seu pleno uso. Outra limitação é o registro de carro obrigatório ao se oferecer uma carona.

Possui um sistema de *ranking* claro e intuitivo, porém a procura de uma carona não torna boa a experiência para o usuário. A aba de filtros impossibilita uma busca clara visto que essa é a única forma de procurar uma carona e a aba rotas não mostra com clareza seu propósito. Mais informações podem ser encontradas em seu *site*[6].

Tabela 2 - Comparação entre os aplicativos analisados

Elementos Analisados	**Zumpy**	**BlaBlaCar**	**Karonas**	**Carona Phone**
Multiplataforma	Sim	Sim	Sim	Não
Sistema de pagamento	Sim	Não	Não	Não
Sistema de avaliação	Sim	Sim	Não	Sim
Cadastro veículo obrigatório	Sim	Não	Sim	Sim
Login variado	Sim	Sim	Sim	Não
Downloads	+ 50 mil	+ 10 Mi	+ 10 mil	+ 10 mil
Avaliação	4,7	4,3	3,9	3,8

Fonte: Elaborado pelo próprio autor com dados extraídos da Google Play em 20/12/2017[7]

[5] http://www.karonas.com.br/
[6] http://caronaphone.com/
[7] https://play.google.com/store?hl=pt

4 PROJETO DO CARONASUFF

Neste capítulo serão esmiuçadas as questões relacionadas ao projeto do sistema proposto. Na seção 4.1 será apresentada a modelagem do sistema abordando requisitos e casos de uso.

4.1 MODELAGEM DO SISTEMA

Nesta seção serão abordados tópicos como requisitos do sistema, onde são divididos em funcionais e não funcionais além da descrição e diagrama de casos de uso.

4.1.1 Requisitos

Requisitos em um software estão associados ao funcionamento do mesmo, mas não se limita a isso. Podem ser considerados requisitos restrições do sistema, recursos mínimos de hardware necessários e até mesmo aspectos visuais da aplicação. “O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado engenharia de requisitos (RE, do inglês *requirements engineering*). ” [7].

4.1.1.1 Requisitos funcionais

Segundo Sommerville [7] “Requisitos funcionais são declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer.”

Requisitos funcionais representam funcionalidades e serviços de um software. Mostram ações que o sistema pode fazer e até mesmo que é impedido de realizar. Requisitos funcionais são de suma importância no projeto de qualquer software, pois,

sem os mesmos, não há funcionalidades. Devem ser claros em suas respectivas descrições, sempre na busca de representar uma funcionalidade de forma clara e que não haja espaço para interpretações ambíguas da mesma. Essa clareza absoluta nem sempre consegue ser alcançada em sistemas maiores e mais complexos que exigem funcionalidades robustas, pois é fácil cometer erros e omitir alguma informação de forma involuntária.

Na tabela 3, serão apresentados os requisitos funcionais do presente trabalho, representados por um id, a descrição do requisito e a prioridade que tal requisito representa para o funcionamento da aplicação.

Tabela 3 - Requisitos funcionais do projeto

ID	Descrição
RF01	O sistema deve permitir que o usuário faça o *login* na aplicação utilizando seu uffmail e a senha cadastrada no aplicativo.
RF02	O sistema deve permitir que o usuário se cadastre indicando nome, sobrenome, uffmail, campus e senha
RF03	O sistema deve listar as caronas que o usuário está participando
RF04	O sistema deve listar as caronas que o usuário ofereceu
RF05	O sistema deve listar as caronas que o usuário solicitou
RF06	O sistema deve permitir que o usuário busque caronas solicitadas por nome de origem ou destino, data, hora, número de vagas, evento
RF07	O sistema deve permitir que o usuário busque caronas oferecidas por nome de origem ou destino, data, hora, preço, número de vagas, evento
RF08	O sistema deve listar todas as caronas pedidas por outros usuários
RF09	O sistema deve listar todas as caronas oferecidas por outros usuários
RF10	O sistema deve permitir oferecer uma carona indicando horário de partida, local de saída, local de destino, local de parada (opcional), data, preço, vagas, vincular com a volta (opcional), vincular com evento (opcional)
RF11	O sistema deve permitir pedir uma carona indicando horário desejado, local de saída, local de destino, vagas desejadas

RF12	O sistema deve permitir que o usuário edite seu nome, sobrenome senha e foto de perfil
RF13	O sistema deve permitir que o usuário edite os dados de uma carona oferecida
RF14	O sistema deve permitir que o usuário exclua uma carona oferecida
RF15	O sistema deve permitir que o usuário edite os dados de uma carona pedida
RF16	O sistema deve permitir que o usuário exclua uma carona pedida
RF17[8]	O sistema deve permitir que os usuários avaliem todos os outros usuários envolvidos na carona ao fim da mesma
RF18	O sistema deve listar os usuários com melhor avaliação
RF19	O sistema deve gerar ao usuário uma sinalização na carona que o mesmo pediu para entrar, indicando que a solicitação está pendente
RF20	O sistema deve gerar ao dono da carona uma sinalização indicando que outro usuário deseja ingressar na carona
RF21	O sistema deve permitir que o usuário cancele sua solicitação de entrada em uma carona
RF22	O sistema deve permitir que o dono da carona exclua participantes da mesma
RF23	O sistema deve permitir que o usuário crie um evento indicando o nome, data, hora, local e uma descrição

Fonte: Próprio autor

4.1.1.2 Requisitos não funcionais

Segundo Sommerville [7] “Requisitos não funcionais são restrições aos serviços ou funções oferecidas pelo sistema. Incluem restrições de *timing*, restrições no processo de desenvolvimento e restrições impostas pelas normas. Ao contrário das

[8] Este requisito não será implementado no presente trabalho

características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo. “

Requisitos não funcionais abrangem diversas áreas na construção de um *software*. Tempo de resposta para um evento, ocupação de área em determinados layouts e a confiabilidade da aplicação são alguns exemplos de requisitos não funcionais que, em alguns casos, pode inviabilizar todo projeto caso não sejam cumpridos. “Por exemplo, se um sistema de aeronaves não cumprir seus requisitos de confiabilidade, não será certificado como um sistema seguro para operar.” [7].

Diferentemente de requisitos funcionais, identificar requisitos não funcionais pode ser um trabalho árduo e complexo muitas vezes. Essa dificuldade se dá pelo fato de que um requisito funcional implica no sistema como um todo. “Por exemplo, para assegurar que sejam cumpridos os requisitos de desempenho, será necessário organizar o sistema para minimizar a comunicação entre os componentes. “ [7]. Os requisitos não funcionais do presente trabalho estão presentes na tabela 4.

Como requisitos não funcionais englobam diversas funções, são divididos em três grandes áreas de aplicação:

1. Requisitos de produto. Se refere as ações e comportamentos de um *software*. Engloba tempo de resposta aceitável em certa situação ou até mesmo quanto de memória é necessário para rodar uma aplicação. Usabilidade e confiança também são tópicos que requisitos de produto abordam.
2. Requisitos organizacionais. São requisitos que gerenciam a comunicação entre o cliente e o desenvolvedor. Engloba linguagem de programação utilizada e ambientes de desenvolvimento por exemplo.
3. Requisitos externos. Esse tipo de requisito atua de forma externa no processo de desenvolvimento. Fatores éticos, ou até mesmo legais para que atuem dentro da lei são tópicos abordados neste requisito.

Tabela 4 - Requisitos não funcionais do projeto

ID	Descrição
RNF01	O sistema não deve permitir que um usuário acesse o sistema sem estar logado.
RNF02	O sistema não deve permitir que mais de um cadastro seja efetuado para o mesmo uffmail.
RNF03	O sistema deve gerar uma chave e enviá-la ao uffmail corrente a fim de permitir o cadastro.
RNF04	O sistema necessita de conexão com a internet para o funcionamento
RNF05	A versão mínima de *Android* para executar a aplicação é 4.4 (KitKat)

Fonte: Próprio autor.

4.1.2 Casos de uso

Esta seção trará uma visão mais ampla sobre o que são casos de uso, a descrição dos casos de uso e seus variados tipos.

Modelo de casos de uso é a representação do sistema e suas funcionalidades vista de uma perspectiva externa de como elementos externos interagem com o sistema. Esses elementos externos são denominados com atores e possuem relacionamentos com um ou mais casos de uso. Na subseção 4.1.2.1 serão exibidas descrições de casos de uso, com seus respectivos fluxos. Por fim, a figura 3 exibirá o diagrama dos casos de uso do presente trabalho.

“Um caso de uso (do inglês *use case*) é a especificação de uma sequência de interações entre um sistema e os agentes externos que utilizam esse sistema.” [8]. Um caso de uso jamais deve transparecer operações internas do sistema. Cada caso de uso deve ser descrito através de uma narrativa onde a mesma representa uma interação entre o ator e o sistema. Como não há uma regra para o formato em que essa descrição deve ser feita, existem algumas opções que são mais utilizadas. Os modos mais conhecidos para se tratar uma descrição de casos de uso são:

- **Descrição contínua:** Descrição que tem como característica uma narrativa de texto livre.

- **Descrição numerada:** Narrativa feita através de uma série de passos numerados, dando uma melhor percepção da interação entre ator e sistema.
- **Descrição particionada:** O estilo da descrição particionada divide a sequência de interações entre sistema e ator em duas colunas, buscando um efeito visual melhor definido do que nos outros dois tipos.

No presente trabalho foi escolhida a descrição numerada para descrever os casos de uso, pois, a mesma tem uma abordagem mais simples e de fácil entendimento para o leitor.

4.1.2.1 Descrição dos casos de uso

A seguir serão apresentadas as seguintes descrições de caso de uso: Oferecer carona, Pedir carona, Cadastrar evento, Convidar usuário para carona, Solicitar entrada em carona, Buscar carona, Editar carona, Excluir carona, Excluir participante e Editar dados pessoais.

Oferecer carona (CS01)
Ator primário: Usuário
Descrição: Registra no sistema uma carona oferecida
Pré-condição: Usuário logado no sistema

Fluxo principal:

1. Usuário insere dados da carona (dia da partida, hora da partida, local de partida, local de destino, preço, vagas, e, sendo opcionais, adição de paradas, vinculação com evento e a volta) e confirma os dados inseridos.
2. Sistema atualiza a lista de caronas oferecidas.
   Fim do caso de uso

Fluxo alternativo:
Passo 1: Campos obrigatórios não preenchidos.

2. Sistema alerta que um ou mais campos obrigatórios não foram preenchidos.
3. Usuário preenche os campos faltantes e confirma os dados inseridos.
4. Sistema atualiza a lista de caronas oferecidas.
   Fim do caso de uso

Fluxo alternativo:
Passo 1: Campos preenchidos incorretamente.

2. Sistema alerta que um ou mais campos estão no formato incorreto.
3. Usuário preenche corretamente os campos incorretos e confirma os dados inseridos.
4. Sistema atualiza a lista de caronas oferecidas.
   Fim do caso de uso

Pedir carona (CS02)
Ator primário: Usuário
Descrição: Registra no sistema uma carona pedida
Pré-condição: Usuário logado no sistema

Fluxo principal:

1. Usuário insere dados da carona (dia da partida, hora da partida, local de partida, local de destino, vagas, e, sendo opcionais, vinculação com evento e a volta) e confirma os dados inseridos.
2. Sistema atualiza a lista de caronas pedidas.
   Fim do caso de uso

Fluxo alternativo:
Passo 1: Campos obrigatórios não preenchidos

2. Sistema alerta que um ou mais campos obrigatórios não foram preenchidos.
3. Usuário preenche os campos faltantes e confirma os dados inseridos.
4. Sistema atualiza a lista de caronas pedidas.
   Fim do caso de uso

Fluxo alternativo:

Passo 1: Campos preenchidos incorretamente

2. Sistema alerta que um ou mais campos estão no formato incorreto.
3. Usuário preenche corretamente os campos incorretos e confirma os dados inseridos.
4. Sistema atualiza a lista de caronas pedidas.

   Fim do caso de uso

Cadastrar evento (CS03)

Ator primário: Usuário

Descrição: Registra no sistema um evento

Pré-condição: Usuário logado no sistema.

Fluxo principal:

1. Usuário insere os dados do evento (nome, dia do evento, hora do evento, local do evento e uma descrição do mesmo) e confirma os dados inseridos.
2. Sistema atualiza a lista de eventos.

   Fim do caso de uso

Fluxo alternativo:

Passo 1: Campos obrigatórios não preenchidos.

2. Sistema alerta que um ou mais campos obrigatórios não foram preenchidos.
3. Usuário preenche os campos faltantes e confirma os dados inseridos.
4. Sistema atualiza a lista de eventos.

   Fim do caso de uso

Fluxo alternativo:

Passo 1: Campos preenchidos incorretamente.

2. Sistema alerta que um ou mais campos estão no formato incorreto.
3. Usuário preenche corretamente os campos incorretos e confirma os dados inseridos.
4. Sistema atualiza a lista de eventos.

   Fim do caso de uso

Convidar usuário para carona (CS04)

Ator primário: Usuário

Descrição: Usuário convida um pedinte para sua carona

Pré-condição: Usuário logado no sistema e ter pelo menos uma carona oferecida

Fluxo principal:

1. Usuário seleciona a carona pedida
2. Sistema mostra todas as caronas que o usuário ofereceu
3. Usuário seleciona para qual carona deseja convidar o pedinte
4. Sistema exibe qual carona foi selecionada
5. Usuário confirma
6. Sistema alerta o pedinte que uma carona foi oferecida a ele
   Fim do caso de uso

Solicitar entrada em carona (CS05)

Ator primário: Usuário

Descrição: Usuário solicita entrada em uma carona oferecida

Pré-condição: Usuário logado no sistema e haver pelo menos uma carona oferecida por algum outro usuário

Fluxo principal:

1. Usuário seleciona carona oferecida
2. Sistema mostra um resumo da carona
3. Usuário solicita entrada
4. Sistema alerta o dono da carona que o usuário solicitou a entrada
   Fim do caso de uso

Buscar carona (CS06)

Ator primário: Usuário

Descrição: Usuário busca uma ou mais carona de acordo com uma palavra chave

Pré-condição: Usuário logado no sistema

Fluxo principal:

1. Usuário seleciona o tipo de carona e digita a palavra chave da pesquisa
2. Sistema retorna uma lista com as caronas que contem a respectiva palavra chave

   Fim do caso de uso

Fluxo alternativo:

1. Nenhuma carona encontrada
2. Sistema alerta que nenhuma carona foi encontrada com a respectiva palavra chave

   Fim do caso de uso

Editar carona (CS07)

Ator primário: Usuário

Descrição: usuário altera dados da carona

Pré-condição: Usuário logado no sistema e haver pelo menos uma carona oferecida ou pedida existente

Fluxo principal:

1. Usuário seleciona a carona que deseja editar
2. Sistema mostra a carona na tela
3. Usuário altera os dados
4. Sistema atualiza a carona com os novos dados

   Fim do caso de uso

Fluxo alternativo:

3. Dados preenchidos incorretamente
4. Sistema alerta que dados não estão no formato correto
5. Usuário insere os dados corretamente
6. Sistema atualiza a carona com os novos dados

   Fim do caso de uso

Editar dados pessoais (CS08)

Ator primário: Usuário

Descrição: usuário altera seus dados pessoais exceto e-mail
Pré-condição: Logado no sistema

Fluxo principal:

1. Usuário seleciona a opção de exibir seus dados pessoais
2. Sistema mostra os dados do usuário com opção de editar foto de perfil, nome, sobrenome e senha.
3. Usuário altera os dados desejados
4. Sistema atualiza os dados do usuário
   Fim do caso de uso

Fluxo alternativo:

3. Dados preenchidos incorretamente
4. Sistema alerta que dados não estão no formato correto
5. Usuário insere dados corretamente
6. Sistema atualiza os dados do usuário
   Fim do caso de uso

Excluir carona (CS09)
Ator primário: Usuário
Descrição: exclusão de uma carona pedida ou oferecida
Pré-condição: Logado no sistema e existir, pelo menos, uma carona oferecida ou pedida

Fluxo principal:

1. Usuário seleciona carona que deseja excluir
2. Sistema mostra os dados da carona
3. Usuário seleciona excluir a carona
4. Sistema alerta que a carona foi excluída
   Fim do caso de uso

Excluir participante (CS10)
Ator primário: Usuário
Descrição: Usuário que ofereceu uma carona exclui um participante

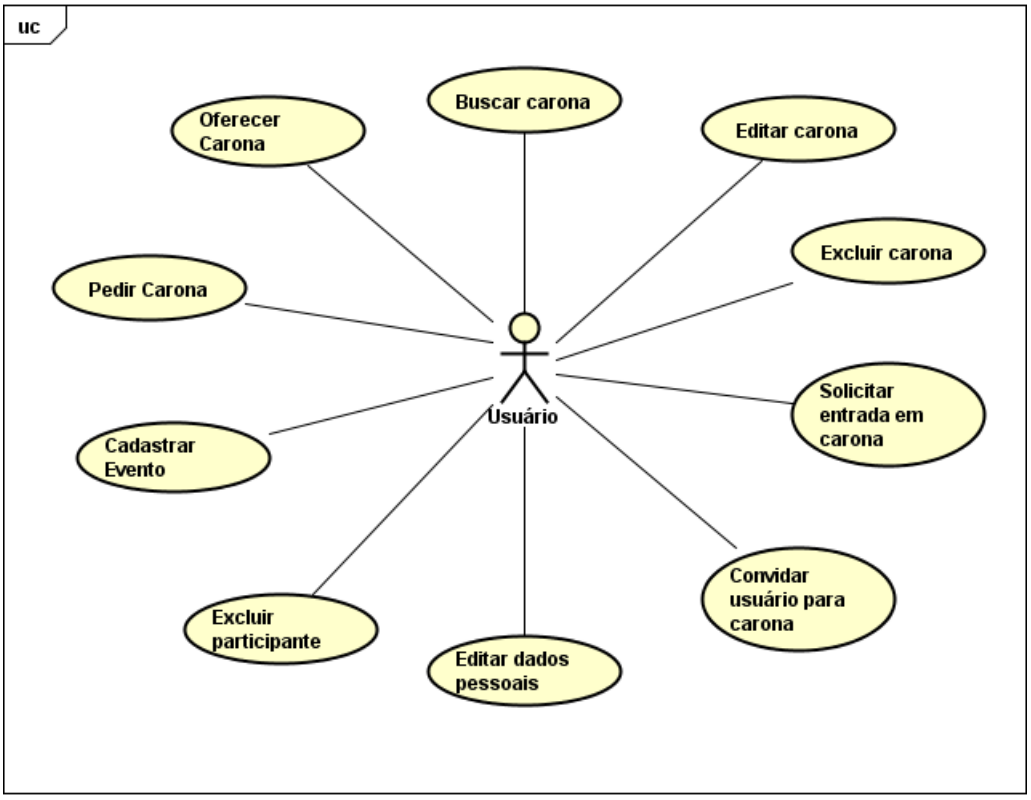
Pré-condição: Logado no sistema, existir ao menos uma carona oferecida e a mesma possuir pelo menos um participante

Fluxo principal:

1. Usuário seleciona carona que ofereceu
2. Sistema mostra os dados da carona
3. Usuário seleciona usuário a ser excluído da mesma
4. Sistema alerta que o usuário foi removido da carona

   Fim do caso de uso

Figura 3 - Diagrama de casos de uso



Fonte: Próprio autor.

4.1.3 Banco de Dados

Este tópico abordará sobre o banco de dados, como foi montada a sua estrutura, a separação e funcionalidade de cada tabela, as diferenças entre um banco de dados relacional e um não-relacional e o porquê ambos foram escolhidos para o trabalho. A implementação do mesmo pode ser encontrada no capítulo 4.3.3.1.

4.1.3.1 Sobre banco de dados

“Banco de dados e sistemas de banco de dados são um componente essencial na vida da sociedade moderna.” [9]. Depositar e retirar fundos em um banco, reservas de hotel ou em um voo são ações que envolvem algum mecanismo de acesso ao banco de dados.

Banco de dados é uma estrutura capaz de armazenar e estruturar informações que façam sentido dentro do contexto que é aplicado. Podem ter uma função local, ou seja, no próprio dispositivo que está executando a aplicação ou em um servidor web sendo acessado remotamente.

“Um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. “ [9]. Um SGBD auxilia o desenvolvedor em aspectos como definição, construção, manipulação e compartilhamento de dados entre usuários e aplicações.

Existem dois modelos de banco de dados que são:

- **Relacional:** Neste modelo os dados são agrupados em linhas e colunas, chamadas de tabelas, cada linda possui o seu identificador chamado de chave primária. Cada tabela possui as suas restrições e relações, organizando o modo em que se relaciona com as outras tabelas do banco de dados.

- **Não relacional:** Modelo alternativo ao relacional, busca-se uma maior escalabilidade dos dados, todas as informações são armazenadas em um registro não sendo necessário fazer uma relação entre tabelas. Podem ser agrupados de quatro maneiras distintas a partir da escolha

de armazenamento. São elas: chave-valor, documento, coluna, orientado a grafos.

Quando se fala em banco de dados relacional e não-relacional, é preciso citar algumas características que se fazem presente de maneira mais forte em um ou outro. São elas:

- **Confiabilidade:** Pelo fato de que modelos relacionais possuem suporte de forma nativa às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), se tornam mais confiáveis em relação a modelos não-relacionais que não possuem essas propriedades nativamente, sendo implementada externamente caso queira.
- **Escalabilidade:** Em alguns bancos de dados relacionais quando se deseja aumentar a escalabilidade, é feita a escalabilidade vertical. Essa escalabilidade consiste basicamente em aumentar o poder de processamento e armazenamento do servidor. Em contrapartida, modelos não-relacionais trabalham com escalabilidade horizontal, ou seja, distribuem uniformemente os processos do banco de dados. Com isso, modelos não-relacionais possuem uma escalabilidade melhor que modelos relacionais.
- **Coerência:** Como dito antes, modelos não relacionais não possuem suporte nativo para propriedades ACID. Essa ausência acarreta em perda na consistência, ou, de modo geral, na coerência do modelo. Perder consistência ocasiona em um ganho de desempenho e escalabilidade.

4.1.3.3 Estrutura do banco de dados

No presente projeto, se escolheu um modelo relacional para o armazenamento de dados textuais pois, como apresentado antes, esse tipo de modelo apresenta propriedades ACID implementadas. Como o trabalho manipula dados que se relacionam a todo instante, a exemplo o fato de em uma carona conter outros

usuários, essa correlação necessita de características que estão presentes de forma mais intensa em modelos relacionais.

Aplicando o modelo relacional ao trabalho obtêm-se cinco tabelas e a figura 4 para ilustra-las. São elas:

- **User:** Esta tabela possui como objetivo armazenar todos os usuários cadastrados e seus respectivos dados, tais como nome, sobrenome e-mail e senha. Seguindo os padrões de um banco de dados relacional cada usuário cadastrado vai possuir o seu código, como *primary key*.

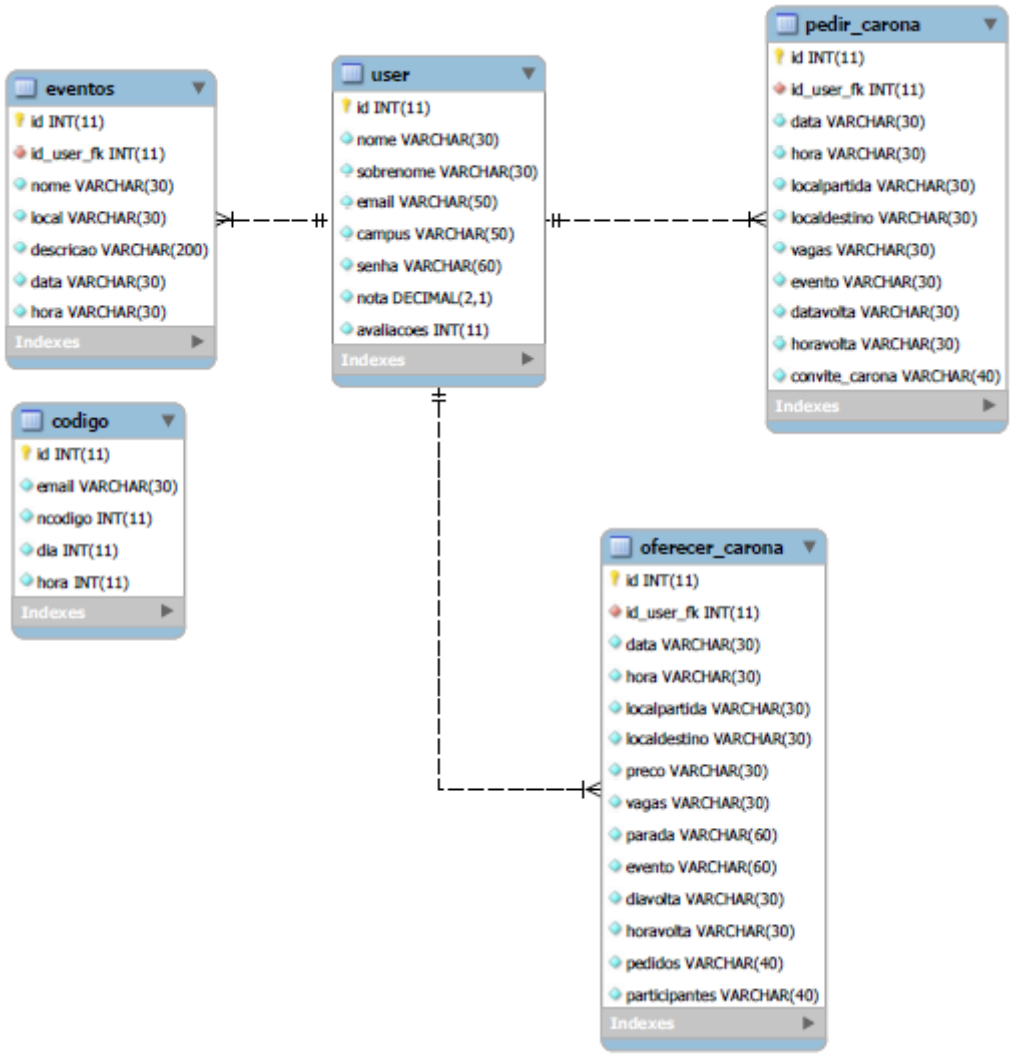
- **Código:** Essa tabela tem a finalidade de armazenar um código temporário que é enviado para o e-mail do usuário na hora que for realizado o cadastro.

- **Eventos:** Nessa tabela são armazenados todos os eventos cadastrados pelo usuário. São armazenados o nome, local, data e hora.

- **Oferecer_carona:** Tabela responsável por armazenar as caronas oferecidas, nela é armazenada todas as informações relacionadas a carona que foi oferecida pelo usuário, tais como data, hora e local de partida, destino, preço, número de vagas disponíveis, local de parada, usuários participantes da carona e se há algum evento relacionado com a carona.

- **Pedir_carona:** Essa tabela armazena os dados das caronas que são solicitadas pelo usuário, tais como local de partida e destino, hora e número de vagas necessárias.

Figura 4 - Modelo do banco de dados



Fonte: Próprio autor

5 DESENVOLVIMENTO

Neste capítulo será mostrado como a implementação do projeto foi feita. A seção 5.1 trará um pouco mais das tecnologias utilizadas para desenvolver o presente trabalho. A seção 5.2 se encarrega do desenvolvimento da aplicação, ou seja, como o software foi implementado. O funcionamento do sistema apresentado como um fluxo de telas será mostrado na seção 5.3. Na seção 5.4, foram realizados testes com estudantes da Universidade Federal Fluminense mostrando de uma maneira geral como foi sua usabilidade com potenciais clientes.

5.1 TECNOLOGIAS

Aqui serão mostradas tecnologias utilizadas como linguagens de programação, IDE e versionamento de código. Ferramentas de um modo geral a fim de auxiliar em um melhor produto final.

5.1.1 Java

A linguagem Java foi desenvolvida por James Gostling, Patrick Naughton, Cris Warth, Ed Frank e Mike Sheridan, na empresa Microsystems, Inc no ano de 1991. A grande motivação para a criação da linguagem foi devido ao fato da maioria das linguagens da época não serem multiplataforma, como por exemplo C e C++.

Java é uma linguagem de programação que utiliza o paradigma de orientação objeto, foi desenvolvida pela Sun Microsystems em 1995, o principal objetivo era fazer uma linguagem que fosse capaz de criar softwares para serem executados em diversos dispositivos com hardwares diferentes.

Para que a linguagem mantivesse uma segurança e que fosse multiplataforma, foi desenvolvido o formato *Bytecode*. O *Bytecode* é um arquivo otimizado que possui um conjunto de instruções para ser executado pela JVM. Assim seria possível desenvolver um *software* capaz de ser utilizado em diversos dispositivos sem precisar adaptar o código.

Outra particularidade da linguagem Java é utilizar do paradigma de orientação a objeto, no qual os seus elementos são tratados como objeto, permitindo uma melhor abstração dos dados. A programação orientada a objetos possui alguns princípios, que são esses:

- **Encapsulamento:** É um mecanismo responsável por manter o código e seus elementos seguros, prevenindo de serem acessados por outro mecanismo indevido.
- **Herança:** Conceito no qual um objeto adquire propriedades de um outro objeto acima no nível de hierarquia.
- **Polimorfismo:** É uma característica que permite duas ou mais classes derivadas da mesma superclasse conseguirem chamar métodos que possuem a mesma assinatura porem comportamentos distintos.

Quando bem aplicados os conceitos apresentados acima, as aplicações mais robustas ganham uma melhor escalabilidade e segurança. Com uma hierarquia bem definida a reutilização do código e os testes se tornam práticos, com o encapsulamento é possível realizar mudanças no código sem que a parte pública note as mudanças, e através do polimorfismo o ganho é um código mais limpo e legível.

Outro ponto importante é o conceito de classes, que são estruturas responsáveis por armazenar as características, comportamentos, propriedades e métodos referentes aos objetos, servindo como um modelo para todos os objetos referentes daquela classe.

Pelo fato da plataforma Android ter sido escolhida, optou-se por isso uma linguagem nativa visando o desempenho da aplicação.

5.1.2 PHP

O PHP (acrônimo para *Hypertext Preprocessor*, originalmente *Personal Home Page*) é uma linguagem *script* e interpretada, de código aberto e com foco maior em aplicações *web*. Muito utilizada no lado do servidor, a linguagem teve avanços possibilitando seu uso também no lado do cliente. Segundo [10], o PHP é uma linguagem que possui como vantagem a compatibilidade com uma variação considerável, incluindo o MySQL.

Criada por Rasmus Lerdorf, seu código foi disponibilizado em 1995, em forma de programas CGI, visando substituir *scripts* Perl que o mesmo utilizava para o desenvolvimento de uma página pessoal. CGI é uma tecnologia que permite a geração de páginas de forma dinâmica fazendo com que um navegador envie

parâmetros para uma aplicação inserido em um servidor *web*. O que diferencia PHP de *scripts* CGI como Perl ou C por exemplo, é que no PHP seu código fica inserido no próprio HTML, enquanto nos demais é necessário que o *script* CGI gere o HTML.

No presente trabalho fez-se o uso de uma extensão chamada PDO. “A extensão PHP *Data Objects* (PDO) define uma interface leve e consistente para acessar banco de dados no PHP” [11]. Uma vantagem do PDO é que, por possuir uma camada de abstração, o acesso aos dados independe de qual banco de dados está utilizando, todos usam as mesmas funções para a emissão de consultas.

A função do PHP no projeto foi o uso de um *Web service*, que fizesse a conexão da aplicação Android com o banco de dados relacional.

5.1.2.1 Web service

Um *Web service* é constituído em um conjunto que são chamados por outros softwares utilizando tecnologias web. Como são linguagens diferentes, é necessária uma linguagem intermediária que faça com que a requisição do sistema chegue a linguagem do *Web service*. Com isso, existem protocolos de comunicação como o SOAP (*Simple Object Access Protocol*) e o REST (*Representational State Transfer*).

O uso de um *Web service* propicia uma integração de diferentes aplicações, o que faz com que seja uma das suas principais vantagens. Outro ponto positivo é o reuso de código, por ser uma tecnologia que independe de plataforma. Atrelado a isso é possível definir outra vantagem, a redução de custos. Tendo um *Web service*, não é necessário criar aplicações para integrar dados, com isso o investimento necessário diminui.

Por não acessar diretamente os dados, um *Web service* é visto como mais seguro deste ponto de vista. Porém, de um ponto de vista interno o *Web servic*e tem como a segurança uma das suas principais desvantagens. Existem diversos mecanismos de segurança, o problema é que não existe um senso comum de qual deve ser adaptado ao *Web service*.

5.1.3 MySQL

MySQL é um sistema gerenciador de banco de dados (SGBD) que possui código aberto mantido pela ORACLE. Protegido por uma licença de *software* livre, desenvolvida pela GNU, é um sistema que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. MySQL é o segundo SGBD mais popular atualmente, perdendo apenas para o Oracle, o que mostra sua força no mercado [12].

Foi criado David Axmark, Allan Larsson e Michale “Monty” Widenius em meados dos anos 90, que trabalhavam juntos desde a década de 80. Seu desenvolvimento e manutenção geram empregos para algo em torno de 400 pessoas, fora as milhares de pessoas que testam o *software* pelo mundo todo.

Vale ressaltar algumas características do MySQL e Microsoft SQL Server a fim de esclarecer o motivo pela escolha do MySQL como o sistema para gerenciamento do banco de dados. O Microsoft SQL Server trabalha melhor com linguagem .NET, que não será utilizada no presente trabalho, enquanto o MySQL apresenta uma maior afinidade com o PHP, que foi a linguagem utilizada para a comunicação com o banco de dados. Por ser um *software* proprietário e, mantido por uma das maiores empresas do mundo, o Microsoft SQL Server vai muito além de um gerenciador de banco de dados. Possui funcionalidades para análise do banco, geração de relatórios, que dão vantagem ao SQL Server em relação ao MySQL, que, por exemplo, se necessitasse de tais funcionalidades, precisaria utilizar ferramentas de terceiros o que não é algo recomendado.

Comparando com o Oracle, é possível notar algumas diferenças no que diz respeito a finalidade de uso. O Oracle, por ser um SGBD mais robusto e com implementações especificas, tem seu maior uso no ambiente corporativo.

Salvar dados binários (imagens nesse caso) no banco de dados pode causar uma sobrecarga considerando a possibilidade de muitos acessos ou muitos dados binários salvos. Tendo em vista tal desvantagem e buscando uma solução que reduza o impacto no desempenho do banco de dados à medida que a quantidade de acessos aumenta, optou-se por salvar imagens em um *storage*. Vale ressaltar que escolhendo separar dados textuais de binários, necessita um gerenciamento manual para garantir a integridade das operações envolvendo esses dados.

Visto que o PHP e o MySQL foram algumas das ferramentas escolhidas para o presente trabalho, optou-se pela utilização do XAMPP para o gerenciamento do servidor em fase de produção. XAMPP é um servidor de código aberto, tendo como ponto central o fato de englobar MySQL e o PHP, reduzindo custos e agilizando o processo de criação e configuração de um servidor.

5.1.4 Firebase

O Firebase é uma plataforma de desenvolvimento *mobile* e *web*, com o objetivo principal em oferecer um *back-end* robusto e de fácil uso. Foi criada pela Envolve, por James Tamplin e Andrew Lee em 2011. Eles disponibilizavam uma API com o intuito de integração de um bate-papo online em seus sites. Porem perceberam que desenvolvedores estavam utilizando o Firebase para sincronizar outros tipos de dados, como o estado em tempo real de um jogo por exemplo. Após perceberem isso, fundaram o Firebase como uma empresa separada em 2012. Em 2014 a Google o adquiriu dando uma escala global a seus serviços.

Por ser uma ferramenta completa possuindo desde serviços de desenvolvimento até de monitoramento de atividade de uma aplicação, tais serviços são modularizados, permitindo uma maior flexibilidade de quem utiliza a ferramenta. Tais serviços são divididos em 4 segmentos. Estes estão ilustrados na figura 6. São eles:

- **Analytics:** “O Google Analytics para Firebase é uma solução gratuita para avaliação de aplicativos. Nele são fornecidos insights sobre uso de aplicativos e engajamento do usuário. ” [13]. O Firebase Analytics é considerado o núcleo da aplicação, sendo possível gerar relatórios para até 500 eventos distintos. É a ferramenta ideal para analisar como os usuários utilizam a aplicação.
- **Develop:** Possui uma série de serviços para criar aplicações que otimizam o tempo de desenvolvimento e, principalmente, geram bons resultados. São eles:
    - Cloud Messaging: entrega e recebimento de mensagens e notificações via *web* e *mobile*.
    - Authentication: Serviço de autenticação do Firebase. Fundamental para aplicações que desejam identificar usuários. Além disso possui suporte para autenticação com Facebook, Google, Twitter e GitHub.

- Realtime Database: Banco de dados NoSQL do tipo chave-valor para armazenamento de dados. Além disso, esses dados são sincronizados em tempo real com todos clientes conectados.
- Storage: Armazena objetos como imagens, áudios, vídeos.
- Hosting: Oferecido para hospedar arquivos HTML, CSS e JavaScript possuindo, ainda, certificado SSL de forma automática.
- Remote Config: Utiliza-se de pares chave-valor inseridas pelo desenvolvedor para personalizar a renderização de acordo com cada usuário, realiza testes de usabilidade ou até mesmo oferecer conteúdo personalizado para alguns usuários.
- Test Lab: Fornece testes automáticos da aplicação em dispositivos virtuais e físicos. É possível utiliza-lo nas etapas do desenvolvimento que necessitem de testes mais precisos, com o intuito de identificar bugs e inconsistências no sistema.
- Crash Reporting: Ferramenta que cria relatórios detalhados de erros da aplicação. Além disso, o Crash Reporting os agrupa em conjuntos e organiza por impacto aos usuários.

Apenas alguns serviços podem estar disponíveis dependendo da plataforma (iOS, Android, *Web*). Para uma melhor visualização, a figura 5 mostra a relação de serviços disponíveis em cada plataforma.

Figura 5 - Serviços x plataformas firebase

	Android	iOS	Web
Cloud Messaging	✓	✓	✓
Authentication	✓	✓	✓
Realtime Database	✓	✓	✓
Storage	✓	✓	✓
Hosting	✗	✗	✓
Remote Config	✓	✓	✗
Test Lab	✓	✗	✗
Crash Reporting	✓	✓	✗

Fonte:https://www.treinaweb.com.br/blog/firebase-descubra-no-que-esta-plataforma-pode-te-ajudar/.

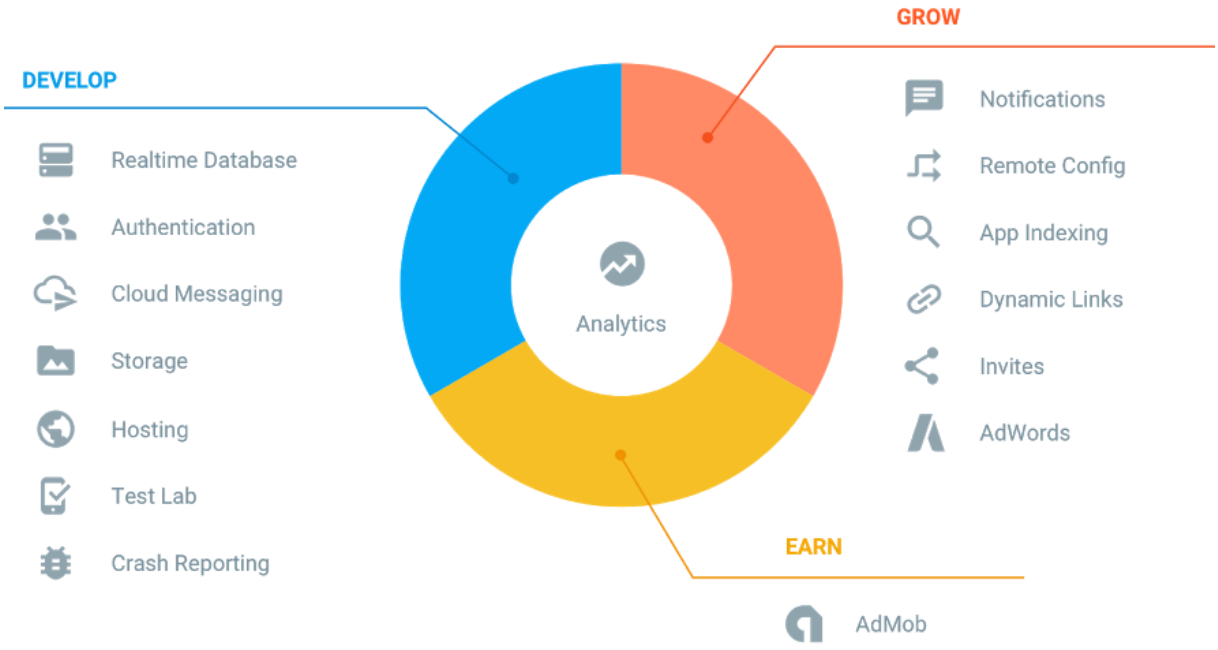
- **Grow:** Este segmento disponibiliza ferramentas para o crescimento da aplicação, ou seja, maior engajamento de usuários.
    - Notifications: Ferramenta de notificação para os usuários. Quando integrado ao *Analytics*, é possível enviar notificações a um grupo especifico de usuários. Compatível com mobile.
    - App Indexing: Integra a aplicação nas pesquisas do Google. Se o usuário já tiver o aplicativo é direcionado para o mesmo, se não, é aberta a opção para instalar. Está disponível apenas para aplicações *mobile*
    - Dynamic Links (Invites): São links dinâmicos que direcionam usuários atuais e potenciais para as aplicações. É compatível com Web, iOS, Android, por isso é possível fazer a migração de usuários de um site para um aplicativo por exemplo.
    - Ad Words: Está vinculado ao *Analytics*, porém se encaixa no presente segmento pois com ele é possível enviar propagandas para usuários específicos, o que potencializa o marketing. Disponível apenas para aplicações mobile.

- **Earn:** O segmento de ganho possui o serviço de *AdMob*, que integra de forma bem fácil propagandas a sua aplicação. Com tais propagandas é possível monetizar sem que atrapalhe a usabilidade do aplicativo.

Vale ressaltar serviços que ainda estão na sua versão beta, como o Cloud Firestore que armazena e sincroniza dados em escala global. ML Kit é um serviço que proporciona operações de *machine learning* a aplicação. *Predictions* aplica a capacidade de *machine learning* do Google para criar previsões, sugestões e afins para seus usuários. A/B *Testing* possui uma infraestrutura planejada para a execução de testes A/B.

Visando o aspecto que a escalabilidade proporciona a um modelo deste tipo, foram utilizados no trabalho os serviços *storage* para o armazenamento de imagens de perfil do usuário e *authentication* para que apenas usuários autenticados pudessem inserir ou requisitar uma imagem do banco.

Figura 6 - Serviços do firebase



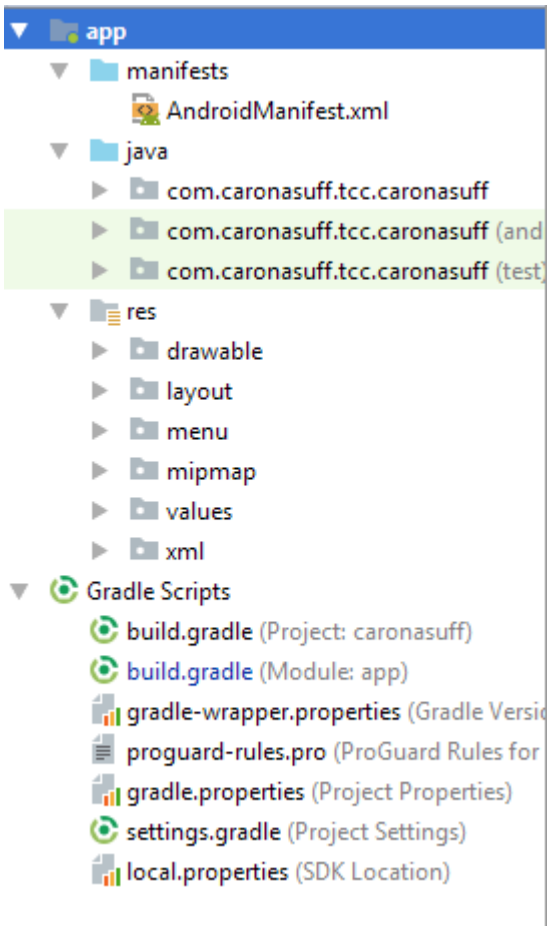
Fonte: https://www.treinaweb.com.br/blog/firebase-descubra-no-que-esta-plataforma-pode-te-ajudar/.

5.1.5 Android Studio

O Android Studio é a IDE oficial para a construção de aplicações Android. Foi lançado no Google I/O em 2013, e, a partir de então, foi adotado como plataforma oficial da Google para desenvolver aplicações Android causando uma migração em massa dos desenvolvedores do Eclipse para o Android Studio.

Com sua versão 3.1.4 no presente momento, o Android Studio possui características que o destaque de outros ambientes de desenvolvimento. A primeira a ser notada é a organização das pastas do projeto. O Android Studio possibilita a visualização do projeto de forma diferente a encontrada em disco. Uma organização modularizada em três pastas sendo elas **manifestos**, contendo o arquivo AndroidManifest.xml. **Java**, onde é encontrado todo código-fonte do projeto, incluindo testes. **Recursos**, é a pasta que possui arquivos que não são código, como configurações de elementos visuais, layouts XML e imagens. A figura 7 ilustra tal estrutura no Android Studio.

Figura 7 - Estrutura do projeto



Fonte: Próprio autor.

Outro fator crucial é a possibilidade de utilizar o Gradle dentro do projeto. Gradle é um sistema para automatizar o processo de compilação da aplicação, que, por padrão, estrutura todo projeto no Android Studio para utilizá-lo. Utilizar o Gradle internamente ao projeto simplifica o uso de bibliotecas externas o que possibilita mais agilidade no processo de importação de ferramentas.

A renderização em tempo real dos *layouts* gera facilidade ao desenvolvedor para detectar erros em suas telas e identificar exatamente em qual ponto o *layout* demonstre erro ou até mesmo visualizar o comportamento de cada componente no *layout*.

Atualmente qualquer desenvolvimento necessita de um versionamento de código, com o intuito de manter o projeto seguro e reutilizar versões em pontos específicos do desenvolvimento por exemplo. O *Android Studio* é integrado com diversos sistemas para tal, como Git que segundo [14] é o mais popular sistema de controle de versão de arquivo.

Mesmo tendo conhecimento do consumo computacional elevado comparado a concorrentes, os autores optaram por utilizar o Android Studio devido as ferramentas citadas acima e outras além dessas, que propiciam agilidade no desenvolvimento e a possibilidade de fornecer uma aplicação de alta qualidade.

5.2 IMPLEMENTAÇÃO

A seção de implementação tem como objetivo mostrar como foi feita a transcrição da modelagem para código. Tópicos como configurações e classes do projeto por exemplo serão abordados nessa seção.

5.2.1 Configurações

Antes de iniciar a programação do presente trabalho, se fez necessário algumas configurações como permissões no manifesto e inclusão de bibliotecas externas.

5.2.1.1 Android Manifest

O arquivo AndroidManifest.xml se encarrega de configurações específicas para *activities*, porém aqui o foco é mostrar permissões que são definidas no arquivo. A figura 8 mostra tal configuração de permissão para o uso da aplicação.

Figura 8 - Permissões de uso Android Manifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Fonte: Próprio autor.

- android.permission.INTERNET: permissão para utilizar a internet. Esta permissão é necessária para a aplicação enviar e receber dados tanto do MySQL quanto do Firebase.
- android.permission.ACCESS_NETWORK_STATE: permissão necessária para checar o estado da internet. Seu papel no projeto é permitir o tratamento de erro quando não houver internet no dispositivo.

- android.permission.READ_EXTERNAL_STORAGE: necessária para que o usuário consiga modificar sua foto de perfil, fazendo a leitura no cartão microSD ou na própria memória interna do aparelho.
- android.permission.WRITE_EXTERNAL_STORAGE: permite o uso de preferências do usuário e o cache de imagem por exemplo.

5.2.1.2 Bibliotecas externas

Muitas vezes quando se deseja implementar algo, é necessário importar ferramentas para o auxílio de uma função ou até mesmo pelo *design* que a mesma proporciona. Porém, como dito no tópico de Android Studio, com o Gradle o uso de bibliotecas externas se torna uma atividade extremamente simples. Abaixo serão listadas as bibliotecas utilizadas descrevendo cada uma delas.

- Android-crop: biblioteca utilizada para o corte da imagem que o usuário selecionará como foto de perfil. Segundo [15], aplicações como SoundCloud, Depop e TextSecure utilizam esta biblioteca.
- Circle imagem view: Responsável pelo formato circular das imagens. Segundo [16] a biblioteca é baseada em RoundedImageView de Vince Mi. É a ferramenta adequada para imagens de perfil pois seu *ScaleType* será sempre CENTER_CROP, que fornece uma escala de imagem uniforme tanto em largura quanto em altura.
- Universal *Image loader*: Segundo [17] UIL *(Universal Image Loader*) é a biblioteca número um de Android em uso no Github. No presente trabalho, a biblioteca foi utilizada com finalidade de cache de imagem com o intuito de reduzir o consumo de internet do usuário.
- Firebase Storage e Authentication: Como citado anteriormente, o serviço de *Storage* foi utilizado com o intuito de armazenar as imagens de perfil e o *Authentication* para assegurar que só usuários autenticados alterem essa imagem.

5.2.2 Classes

Como citado no capítulo 5.1.1, buscando uma melhor estruturação do código, foram criadas diversas classes na qual agrupam métodos que são utilizados em diversas partes do código. Neste capítulo vamos explicar o conceito das classes e quais as suas funcionalidades para o projeto.

- **ConexaoBd:** Classe responsável por realizar a conexão com o banco de dados MySQL. As requisições são feitas através do método POST, no qual os parâmetros são enviados para o webservice via URL.

Duas classes nativas do java.net são utilizadas para fazer a comunicação com o webservice, a classe URL e a classe HttpURLConnection. O método principal da classe chama-se postDados que possui como parâmetros duas *Strings*, uma é a URL do usuário e a outra são os parâmetros que vão compor essa URL, podendo ser vazio ou não.

Algumas configurações são feitas com o intuito de deixar a conexão mais confiável, como não usar cache na conexão e usar o tamanho do parâmetro variável, ou seja, o tamanho do conteúdo proporcional a quantidade de parâmetros inseridos.

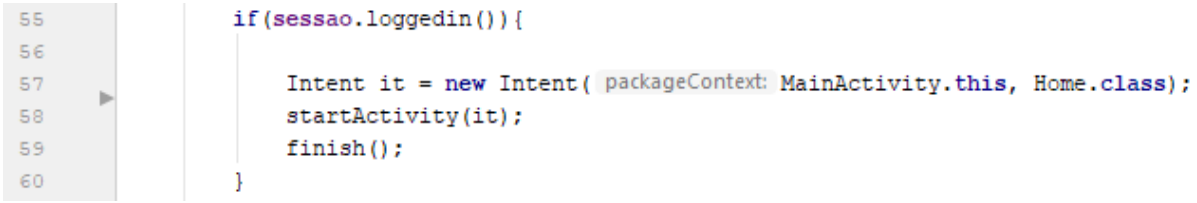
- **Criptografia:** Esta classe é responsável pela criptografia da senha do usuário, tanto para salvar no banco de dados, quanto para autenticação do mesmo. Ela possui dois métodos, o primeiro chamado gerarHash que recebe a senha e o algoritmo de criptografia que será utilizado e gera um *array* de *bytes*. O outro método chamado de getHexa recebe um *array* de bytes e retorna uma *string* hexadecimal. Esse processo garante uma confiabilidade para a senha do usuário.

- **MainActivity:** Esta classe possui os métodos que controlam o login ou cadastro do usuário. Se a opção escolhida for o login, é chamado o método responsável pela verificação do *email* e senha no webservice e posteriormente no Firebase. Como as imagens são armazenadas no Firebase Storage, se faz necessária uma autenticação dupla, primeiro a verificação no MySQL e em sequida no Firebase *Authentication*. Caso o usuário seleciona a opção de cadastro é chamada a classe SolicitaCod.

Ao iniciar a execução deste arquivo, o método onCreate criará uma instancia da classe Sessão, onde será feita uma verificação se o usuário já está logado. Essa validação é feita na figura 9 onde, na linha 55, o método loggedin retorna *true* caso o usuário esteja logado. Na linha 57 é criado um novo *Intent* para fazer a transição entre

activities. O método na linha 58 significa a execução do *Intent*, e por fim na linha 59 a *activity* atual é finalizada.

Figura 9 - Método loggedin



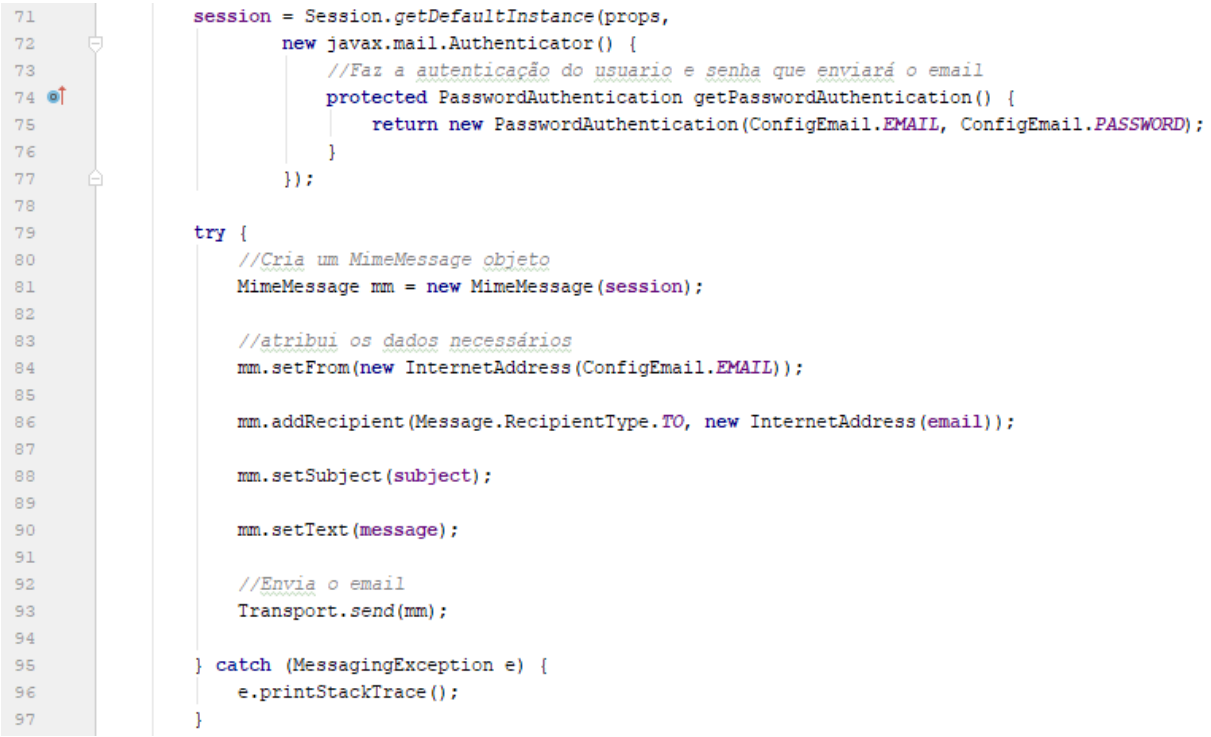
Fonte: Próprio Autor.

Por serem utilizadas informações sensíveis do usuário, como a senha, é necessário realizar uma criptografia nos dados antes da autenticação ser requerida. Dentro do método onPostExecute é realizada uma chamada ao método getHexa pertencente à classe Criptografia, passando como parâmetro a senha e o nome do algoritmo MD5. Vale ressaltar que o uso da classe nativa do Android ConnectivityManager permite que o sistema teste se há ou não conexão com a internet, verificando a conexão sempre que requisições aos bancos de dados são feitas.

- **EnviaEmail:** Extende a classe AsyncTask, que é uma classe que trabalha de forma assíncrona. A classe EnviaEmail possui a finalidade de enviar um e-mail com ó código de cadastro para o e-mail do usuário que o solicitou.

Para o envio desse se fez uso de uma API chamada JavaMail. Um dos *imports* feitos é o da classe MimeMessage, onde será instanciado um objeto vazio do mesmo e, a partir daí, preencher com os dados necessários. Como o nome da classe já diz, a mensagem a ser enviada é do tipo MIME (*Multipurpose Internet Mail Extensions*). A figura 10 ilustra o procedimento de inicialização de um objeto MimeMessage e a atribuição do mesmo. Das linhas 71 até 77 cria-se uma instância vazia e faz a inserção do e-mail com a devida senha que enviará o código ao usuário. Na linha 81 essa instância do remetente criada servirá de parâmetro para o novo objeto MimeMessage. A linha 84 insere o e-mail que enviará o código, que será comparado com o e-mail autenticado anteriormente. O destinatário é inserido na linha 86, o assunto do e-mail na linha 88 e na linha 90 o texto em si. Na linha 93 é iniciado o envio da mensagem.

Figura 10 - Código para envio de e-mail



```
session = Session.getDefaultInstance(props,
        new javax.mail.Authenticator() {
            //Faz a autenticação do usuario e senha que enviará o email
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(ConfigEmail.EMAIL, ConfigEmail.PASSWORD);
            }
        });

try {
    //Cria um MimeMessage objeto
    MimeMessage mm = new MimeMessage(session);

    //atribui os dados necessários
    mm.setFrom(new InternetAddress(ConfigEmail.EMAIL));

    mm.addRecipient(Message.RecipientType.TO, new InternetAddress(email));

    mm.setSubject(subject);

    mm.setText(message);

    //Envia o email
    Transport.send(mm);

} catch (MessagingException e) {
    e.printStackTrace();
}
```

Fonte: Próprio autor.

- **Sessao:** A classe Sessao é responsável pelo cache de e-mail, senha e imagem de perfil do usuário. Se baseia na interface chamada SharedPreferences. Segundo [18], a classe fornece fortes garantias de consistência, porém é necessário o uso para dados pequenos, pois, pode causar queda de desempenho.

Sempre que for adicionar algum dado a sessão, é necessário passar como parâmetro uma tag também. Essa *tag* serve para identificar os dados, sejam eles *strings*, inteiros ou booleanos.

Para realizar uma modificação no SharedPreferences, tal como adicionar ou excluir dados, necessita o uso do método *commit* para que as alterações tenham validade. No presente trabalho usou-se o método *apply* para realizar o *commit*, onde o mesmo funciona da mesma forma que o *commit*, porém feito de forma assíncrona.

- **SolicitaCod:** Responsável por gerar um código numérico aleatório de cinco dígitos para ser enviado ao usuário. Também é feita uma checagem ao digitar o e-mail para evitar que o usuário gere muitos códigos em um curto espaço de tempo. Caso haja um código corrente o sistema alertará o usuário.

- **SearchableActivity:** Classe responsável pelo sistema de filtro ou pesquisa da aplicação. No momento em que o usuário digita a *string* que deseja pesquisar, a *activity* recebe a informação se a pesquisa é referente as caronas oferecidas, pedidas ou as caronas do usuário. O método que cuida dessa informação se chama SearchQuery e está ilustrado na figura 11. Na linha 156 recebe um objeto da classe *Intent* como parâmetro. As linhas 158,159 e 160 mostram a obtenção de duas *strings* que são passadas via *Intent*. São elas o tipo da carona e a palavra de busca a ser feita. Na linha 162 é feita a comparação da *string* tipo. Caso seja verdadeira executa o código presente na condição. Na linha 166 é feita a checagem do estado da rede, caso haja internet, são feitas as atribuições da URL e dos parâmetros para a requisição HTTP e na linha 171, é feita a execução, ou seja, é feita uma requisição ao arquivo PHP search_oferecidas.php. Pelo fato do método ser um tanto quanto extenso, apenas a condição para o tipo de caronas oferecidas foi inserido. Porém é feita a mesma checagem para caronas pedidas e as caronas do usuário fazendo a requisição aos respectivos arquivos PHP.

Figura 11 - Método para pesquisa

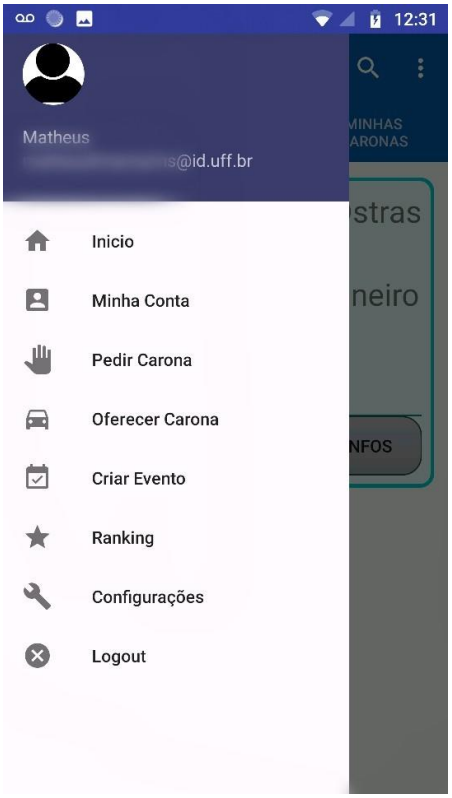
```
156     public void SearchQuery(Intent intent)
157     {
158         Bundle bundle = intent.getExtras();
159         tipo = bundle.getString( key: "tipocarona");
160         query = bundle.getString( key: "query");
161
162         if(tipo.equals("oferecida"))
163         {
164             txtLabel.setText("CARONAS OFERECIDAS");
165             getSupportActionBar().setTitle("'"+query+"'");
166             if(networkInfo != null && networkInfo.isConnected())
167             {
168                 String idusuario = sessao.getString( tag: "id_usuario");
169                 url = conexaoBd.getUrl()+"/caronasuff/search_oferecidas.php";
170                 parametros ="query="+query+ "&idusuario="+idusuario;
171                 new SolicitaDados().execute(url);
172             }else{
173                 Toast.makeText( context: this, text: "Sem conexão com a internet",
174                         Toast.LENGTH_LONG).show();
175             }
176         }
```

Fonte: Próprio autor.

A filtragem dos dados fica totalmente encarregada do *web service*. No momento em que a query para pesquisar no banco de dados é feita, já é aplicada a restrição baseada na *string* que o usuário digitou. Isso permite que o *json* retornado só precise ser exibido, dando ganho de desempenho ao sistema.

- **Home:** É o ponto central da aplicação. Nessa classe é exibida um menu lateral exatamente como mostra a figura 12, com as opções disponíveis na aplicação. Nesse menu estão contidas as funcionalidades do sistema. Além disso, outro componente primordial presente nesse *layout* é o *ViewPager*.

Figura 12 - Menu Lateral



Fonte: Próprio autor.

ViewPager é um gerenciador de layouts, onde é possível deslizar para direita ou esquerda para visualizar o layout desejado. No presente trabalho existem três layouts no ViewPager da Home: oferecidas (para as caronas oferecidas), pedidas (para as caronas pedidas) e minhas caronas (para as caronas do usuário, criadas por ele ou na qual está inserido).

O ViewPager é sincronizado a um TabLayout, com o intuito de informar visualmente de uma forma simples em qual layout o usuário se encontra e quais os nomes de todos os layouts. A figura 13 demonstra como é a sincronização entre os dois elementos para o usuário.

Figura 13 - ViewPager com TabLayout

Fonte: Próprio autor.

Repare que existe uma marcação embaixo da palavra para demonstrar que a mesma é a página atual.

- **Cadastro:** Classe java responsável pelo cadastro do novo usuário no sistema. Nesta classe não tem nenhuma implementação elaborada, apenas um formulário para preenchimento e envio ao *web service*.

- **ExibirPerfil:** Responsável pelas informações de um usuário que é visível para outros. Imagem de perfil, nome, avaliações geradas por outros usuários são dados exibidos.

- **PerfilCarona:** Classe encarregada de exibir uma carona oferecida. Todas funcionalidades primárias do sistema teu seu fluxo gerenciado nessa classe. Gerenciar os usuários que estão participando da carona, aceitar ou recusar novos usuários na carona. Quando a carona é do tipo pedida, é possível aceitar ou recusar uma carona que foi oferecida.

Existem duas formas da classe ser chamada através de um *Intent*. Primeiro pela aba de caronas oferecidas. Ao clicar para solicitar a entrada em uma carona oferecida, o usuário é direcionado para esta classe. A segunda forma é pela aba minhas caronas. Nessa aba são encontradas caronas que o usuário está participando, e caronas oferecidas e pedidas pelo próprio usuário sendo diferenciadas através de cores e textos. Essa diferenciação é necessária para a manipulação visual dos pedidos e participantes de cada carona.

Quando um usuário clica em uma carona oferecida por outro usuário ele não consegue visualizar solicitações de entrada de outros usuários e nos atuais participantes ele só consegue excluir a sua própria participação na carona. No

momento que o usuário clica para visualizar a própria carona oferecida, ele tem total controle sobre os participantes. Consegue gerenciar as solicitações, aceitando ou não a entrada de outros usuários, é possível também excluir qualquer integrante da carona além de conseguir excluir a carona em si.

Como mostrado na seção 2.1.1.3, o conceito de fragmento foi utilizado no presente trabalho. Presente nas páginas do menu lateral e na Viewpager da *Home*. Abaixo estarão listadas as classes usadas como fragmentos.

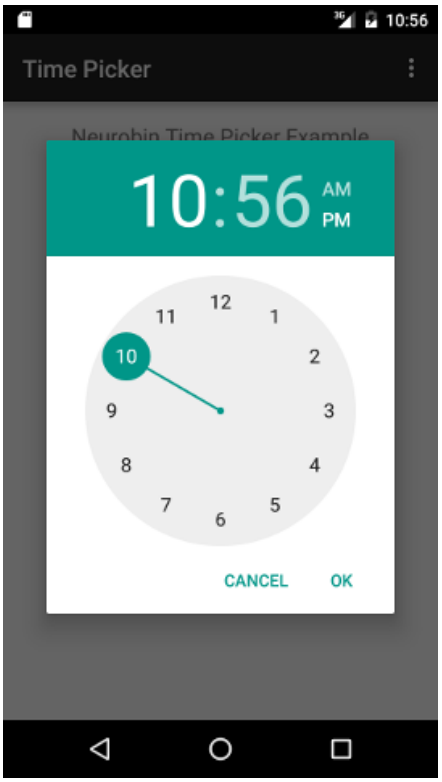
- **CriarCaronaMotorista:** Formulário para inserção dos dados das caronas que serão oferecidas a outros usuários. Assim como citado no capítulo de modelagem na seção correspondente aos requisitos funcionais, um usuário poderia adicionar uma ou mais paradas, vincular a carona com a data de volta. Funcionalidades como essas são gerenciadas na classe CriarCaronaMotorista.

- **CriarCaronaPassageiro:** Idêntico a classe CriarCaronaMotorista. Tirando o fato dos campos a serem preenchidos serem um pouco diferentes, o princípio de gerenciamento dos campos do formulário é o mesmo.

- **CriarEvento:** Formulário para criação de um novo evento. Seja uma palestra, competições acadêmicas ou até mesmo uma festa universitária. Os eventos criados são gerenciados nas duas classes citadas anteriormente. São carregados e exibidos, onde o usuário pode escolher um para vincular a sua carona.

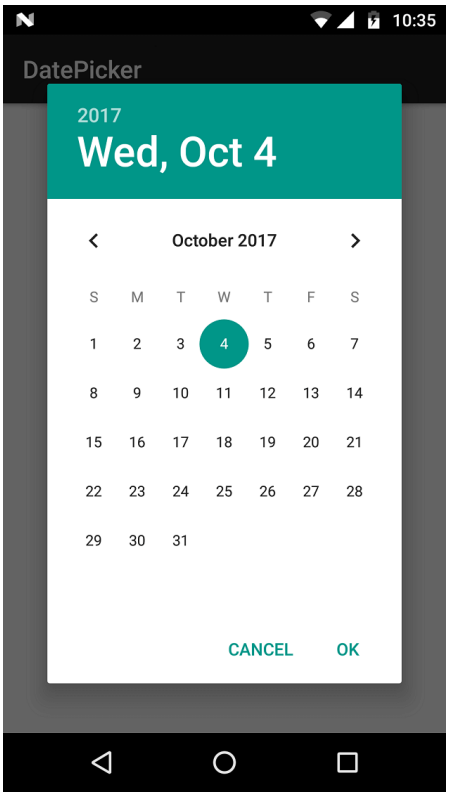
- **TimePickerFragment** e **DatePickerFragment:** Essas classes são responsáveis por criarem as caixas de diálogo referente a escolha de data e hora da carona tanto para criar e pedir caronas quanto para criar eventos. A figura 14 mostra um TimePicker e a figura 15 ilustra um DatePicker.

Figura 14 - TimePicker



Fonte: https://neurobin.org/docs/android/android-time-picker-example/.

Figura 15 - DatePicker.



Fonte: http://www.zoftino.com/android-datepicker-example.

A estrutura dessas classes é de certa forma bem simples. Elas possuem dois métodos. O primeiro é um método para setar os argumentos que irão compor o diálogo, no caso o dia mês e ano atual para a DatePickerFragment e a hora e o minuto atual para a classe TimePickerFragment. Após setar as variáveis, o método onCreateDialog é chamado onde o mesmo retorna o diálogo de cada classe a partir dos parâmetros como tema do diálogo e as variáveis privadas que foram setadas anteriormente. No caso da data, é necessário realizar uma configuração a mais.

A figura 16 ilustra tal procedimento. Da linha 42 até a 44 é feita a inicialização de um objeto do tipo DatepickerDialog inserindo algumas configurações como o tema do diálogo e o dia, mês e ano atuais. Na linha 45 é feito o procedimento para colocar a data mínima sendo a atual, para que o usuário não consiga oferecer ou pedir caronas para datas anteriores a atual.

Figura 16 - Método onCreateDialog

```
41      public Dialog onCreateDialog(Bundle savedInstanceState) {
42          DatePickerDialog dialog = new DatePickerDialog(getActivity(),
43                  AlertDialog.THEME_DEVICE_DEFAULT_DARK ,
44                  ondateSet, year, month, day);
45          dialog.getDatePicker().setMinDate(System.currentTimeMillis()-1000);
46          return dialog;
47      }
48  }
```

Fonte: Próprio autor.

- **FragmentOferecidas**, **FragmentPedidas** e **FragmentMinhasCaronas:**

Esses três fragmentos compõem a ViewPager da Home já citada. Todos eles utilizam o conceito de RecyclerView para exibir os dados. RecyclerView é uma evolução do ListView e do GridView, que são componentes presentes desde o início do Android. O RecyclerView possui algumas vantagens em relação aos seus antecessores. Possui uma melhor performance pois, como o próprio nome sugere, ele identifica e reutiliza views que não estão visíveis ao usuário e insere novos valores de acordo com a posição da lista. Além disso, em um RecyclerView, a atualização é feita apenas em itens alterados, diferentemente de um ListView que quando um item era adicionado, removido ou modificado permitia apenas a alteração de toda a lista.

Para utilização de um RecyclerView, é necessário o uso de mais alguns componentes. São eles:

- LayoutManager: Gerencia a posição dos itens no layout (lista horizontal, vertical, em grade são algumas das possibilidades). Tal vantagem faz com que não seja necessário recriar toda a estrutura do RecyclerView a nível de execução. Em um ListView por exemplo, só existe o suporte para uma lista vertical. Uma modificação para GradView exigiria ajustes mais trabalhosos do que com RecyclerView.

- Adapter: Responsável por associar o conteúdo a View. Cada item dessa lista de conteúdo se tornará um item na lista da View. É no Adapter que é decidido se um item será exibido ou não.

- ViewHolder: É a referência visual de cada item da lista, que se aplica a todos os elementos.

Nos fragmentos FragmentOferecidas e FragmentPedidas a lista é feita em forma de cartões, utilizando o CardView. Já na FragmentMinhasCaronas a exibição é feita no formato de uma lista simples. Com o intuito de ganhar desempenho a exibição dos cartões, é feita uma divisão da lista de objetos obtida do *web service*.

A lista recebida é dividida em pequenas partes, que são exibidas conforme o usuário desce a lista. O método onScrolled ilustrado na figura 17 mostra o funcionamento do mesmo, na linha 169, é feita a checagem se o tamanho do *array* é igual a posição do último item completamente visível.

A adição de um se dá pelo fato de que a posição se inicia em zero. Caso a condição seja atendida, inicia o próximo de carregar mais itens da lista. Nas linhas 172 e 173 as variáveis *Start* e *End* são incrementadas pois elas vão indicar o início e o fim do novo pedaço a ser inserido.

Na linha 175 é chamado o método Splitjson criado pelos próprios autores que recebe o json2 que no caso é o todo o JSON retornado pelo *web service* e as variáveis para controlar o corte do mesmo. Retornada a *string*, na linha 178 é criado um novo JSONArray a partir do pedaço gerado.

Da linha 179 até 182 cada elemento do *array* é transformado em um JSONObject e a partir daí chamado o método do adapter addItemJson para inserir o novo elemento na lista que é exibida ao usuário.

Figura 17 - Método onScrolled

```
163         @Override
164         public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
165             super.onScrolled(recyclerView, dx, dy);
166
167             LinearLayoutManager llm = (LinearLayoutManager) recyclerView.getLayoutManager();
168             CaronasOferecidasAdapter adapter = (CaronasOferecidasAdapter) recyclerView.getAdapter();
169             if(jsonArray.length() == llm.findLastCompletelyVisibleItemPosition()+1)
170             {
171
172                 Start = End+1;
173                 End += 3;
174
175                 String newjson = Splitjson(json2,Start,End);
176
177                 try {
178                     JSONArray j = new JSONArray(newjson);
179                     for(int i =0;i< j.length();i++)
180                     {
181                         JSONObject row = j.getJSONObject(i);
182                         adapter.addItemJson(row,jsonArray.length());
183                     }
184                 } catch (JSONException e) {
185                     e.printStackTrace();
186                 }
187
188
189             }
190         }
```

Fonte: Próprio autor.

- **MinhaConta:** Fragmento que gerencia dados pessoais do usuário.

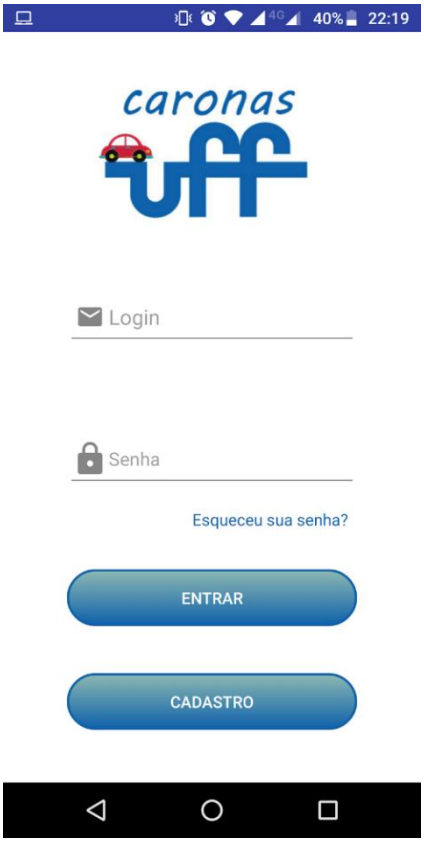
5.3 FLUXO DE TELAS

Nesta seção serão apresentados alguns cenários que podem ser executados pelo usuário no aplicativo, tais como criar o cadastro, oferecer e solicitar uma carona.

5.3.1 Criar Cadastro

Ao entrar no aplicativo pela primeira vez, será apresentada a tela da figura 18 que terá a opção de login ou realizar o cadastro.

Figura 18 - Tela inicial



Fonte:Próprio autor.

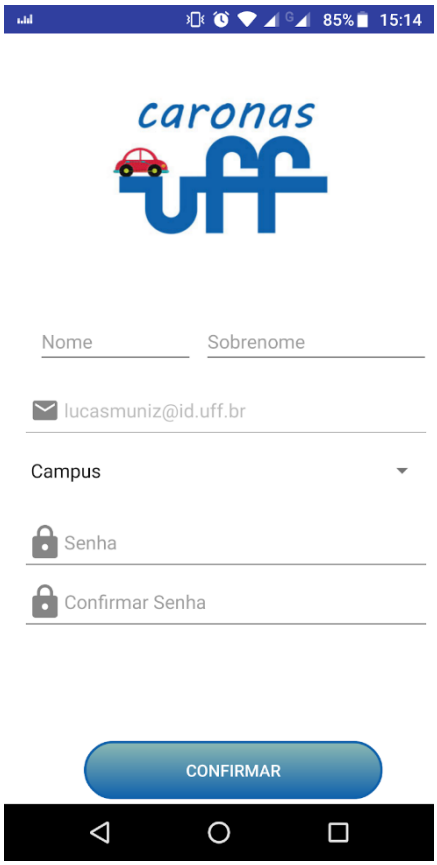
Quando selecionada a opção de cadastro uma numa tela será exibida e será solicitado o código que foi enviado para o e-mail informado pelo usuário como mostra a figura 19.

Figura 19 - Tela solicitação código

Fonte: Próprio autor.

Após inserir o código o usuário será direcionado para uma tela que será solicitado algumas informações pessoais para finalizar o cadastro, como nome, sobrenome campus e uma senha, como apresentado na figura 20.

Figura 20 - Tela cadastro usuário



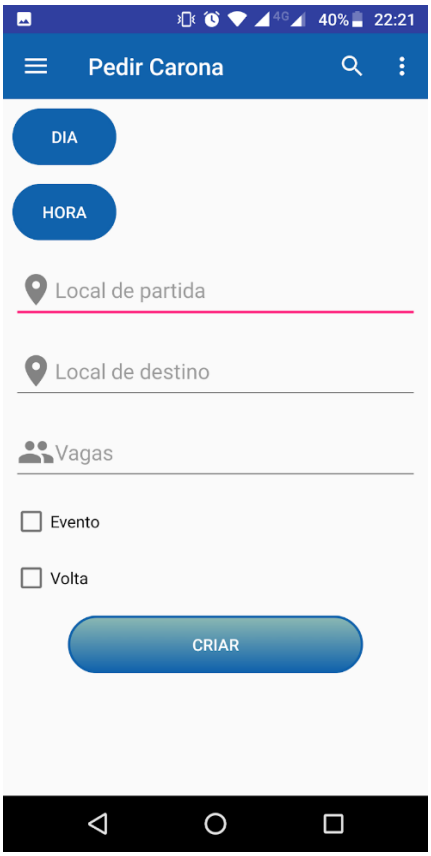
Fonte: Próprio autor.

5.3.2 Pedir e oferecer carona

Neste cenário o usuário deseja oferecer ou solicitar uma carona, o fluxo inicia ao acessar o menu lateral, onde será apresentado algumas opções entre elas pedir e oferecer carona.

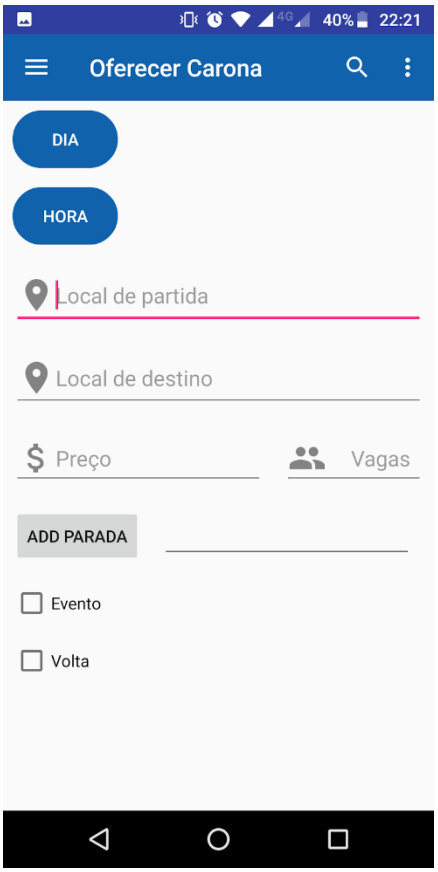
Caso a opção selecionada for pedir carona, uma nova tela será exibida solicitando as informações referentes a carona que o usuário deseja solicitar, como mostra a figura 21. Se o usuário selecionar a opção de oferecer carona uma tela similar é apresentada, porém com a opção de incluir o valor que será cobrado pela carona, conforme a figura 22.

Figura 21 - Tela pedir carona



Fonte: Próprio autor.

Figura 22 - Tela oferecer carona

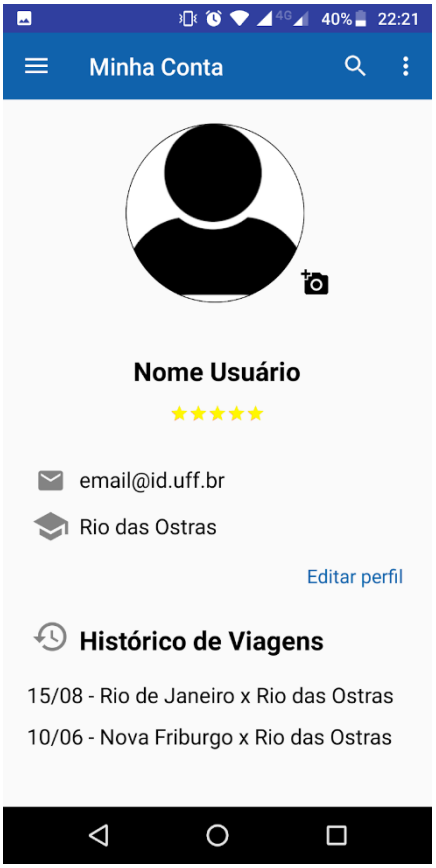


Fonte: Próprio autor.

5.3.3 Perfil do usuário

O usuário será capaz de acessar o seu perfil, onde serão exibidas algumas informações sobre a sua conta, como por exemplo o e-mail, campus e o seu ranking atual. Informações sobre as suas últimas caronas serão exibidas e poderão ser acessadas ao clicar em cima da carona desejada. A figura 23 ilustra o *layout*.

Figura 23 - Tela perfil usuário



Fonte: Próprio autor.

5.4 TESTES

Nesta seção serão abordados testes de usuário com o usuário final, alunos da UFF. Segundo [7] Teste de usuário é uma etapa na qual usuários fornecem entradas e conselhos sobre as funcionalidades do sistema. Esse tipo de teste é essencial em qualquer desenvolvimento de software pois, para um desenvolvedor, é impossível replicar em um ambiente de programação comportamentos do usuário final.

Os usuários de teste pertencem a comunidade UFF. Como esse é o único requisito pessoal para o uso da aplicação, não se faz necessário conhecer dados como cidade, sexo ou até mesmo o curso do usuário.

Visando testar a usabilidade da aplicação e suas principais funcionalidades, os autores requisitaram aos usuários de teste que executassem na aplicação oito atividades. São elas:

- Oferecer uma carona
- Requisitar uma carona
- Aceitar um usuário na sua carona
- Recomendar carona a pedinte
- Aceitar carona sugerida a você
- Mudar imagem de perfil
- Sair de uma carona já inserido
- Criar um evento

Após realizadas as atividades, os usuários foram submetidos a quatro perguntas contidas na tabela 5. As tabelas 6 a 9 representam as respostas dos usuários de teste. São elas:

Tabela 5 - Perguntas feitas ao final do teste

Número	Pergunta
1	Sentiu dificuldade em realizar algumas das ações propostas? Se sim, quais?
2	O que você consideraria ponto (s) positivo (s) na aplicação?
3	O que você consideraria ponto (s) negativo (s) na aplicação?
4	Considerações finais?

Fonte: Próprio autor.

A pergunta de número 1 é focada na usabilidade do sistema, se o usuário conseguiu encontrar corretamente as funcionalidades que eram necessárias para realizar as funções pedidas. As perguntas de número 2 e 3 tem como objetivo saber na opinião dos usuários o que lhe mais chamou atenção, tanto positiva como negativamente. Por último, a pergunta 4 abre espaço para o usuário de teste fazer comentários sobre o que bem entender a respeito da aplicação. Sugerir mudanças em

alguma tela, funcionalidades ou comentar sobre a aplicação de um modo geral são alguns exemplos do que pode ser informado aos desenvolvedores.

Com isso temos as seguintes respostas dos usuários de teste baseadas nas atividades feitas:

Tabela 6 - Resposta usuário 1

Pergunta	Resposta
1	Não
2	Facilidade de encontrar caronas
3	Não entendi o que são os eventos e para que eles servem
4	Aplicativo muito útil para quem não mora na cidade onde estuda. Por considerar apenas cadastro de alunos com email da UFF, isso torna o app muito mais seguro e confiável

Fonte: Próprio autor.

Tabela 7 - Resposta usuário 2

Pergunta	Resposta
1	Não, o aplicativo é bem específico ao que se propõe fazer
2	A confiança que se passa em precisar ter um e-mail da UFF criado, além de ser uma solução interessante para o ramo que se aplica
3	Achei que faltou algum tipo de filtro para realizar um “convite de carona”. Por exemplo, eu posso convidar qualquer pessoa que tenha pedido uma carona, mesmo que eu não tenha uma carona oferecida para tal destino em tal data.
4	O aplicativo tem uma ideia muito inovadora por ser algo específico para alunos. Algumas funcionalidades ainda parecem estar em desenvolvimento, e algumas validações precisam ser feitas, mas no geral é um aplicativo que atende ao que foi proposto

Fonte: Próprio autor.

Tabela 8 - Resposta usuário 3

Pergunta	Resposta
1	Não
2	Caso o aplicativo seja abraçado por seu público alvo será de grande utilidade. Além disso, pode-se dizer que a interface é bem simples e intuitiva.
3	Alguns pequenos bugs foram encontrados durante o cadastro para utilizar o aplicativo.
4	O aplicativo é útil e apresenta facilidade e bom desempenho em suas funcionalidades, gostaria de acompanhar e utilizar após lançamento.

Fonte: Próprio autor.

Tabela 9 - Resposta usuário 4

Pergunta	Resposta
1	Não.
2	A opção de eventos, a qual podemos encontrar caronas que irão para o evento desejado.
3	Algumas falhas e problemas, como a que eu tive ao criar o cadastro.
4	Achei a ideia e as funcionalidades do aplicativo ótimas. Seria uma ótima ferramenta para se usar no meio acadêmico.

Fonte: Próprio autor.

Baseado no que foi respondido pelos usuários, algumas observações podem ser feitas:

- A usabilidade foi aceita por todos que testaram a aplicação.
- A parte de eventos, que é um dos diferenciais da aplicação, precisa ser lapidado. Algumas ideias são:

    - Remover a opção de inserir um destino quando se vincula um evento a uma carona.
    - Listar todos os eventos disponíveis
- Outra característica considerada um diferencial foi o fato da exclusividade para a comunidade UFF.

6 CONCLUSÃO

O presente trabalho apresentou o desenvolvimento de uma aplicação móvel para o gerenciamento de caronas compartilhadas de integrantes como um todo da Universidade Federal Fluminense. Por ser uma universidade bastante interiorizada, o CaronasUFF surge como uma alternativa exclusiva para a comunidade UFF com o intuito de garantir segurança e facilidade nas viagens semanais de diversos estudantes e professores.

Antes de iniciar o trabalho foi feita uma prospecção abordando trabalhos relacionados, ou seja, aplicações que propunham como funcionalidade principal a mesma do presente trabalho. O intuito da pesquisa é detalhar o que as aplicações similares no mercado oferecem e, assim, comparar suas funcionalidades e o que cada uma possui de positivo e negativo.

O desenvolvimento trata a modelagem do sistema gerando requisitos, casos de uso e os devidos diagramas, com o intuito de solidificar e identificar possíveis problemas de implementação antes mesmo que qualquer código tivesse sido escrito. Na implementação é mostrada de forma ampla todas tecnologias e ferramentas utilizadas na implementação, juntamente com o código em si. Por fim testes de usuário com a intenção de testar funcionalidades básicas com o usuário final.

CaronasUFF é uma aplicação com grande potencial pois caronas compartilhadas podem gerar desconfiança em alguns usuários, e o presente projeto limita o uso ao ecossistema da Universidade Federal Fluminense, gerando mais uma opção para usuários deste tipo de segmento.

6.1 TRABALHOS FUTUROS

Muitas possibilidades para funcionalidades existem para esse trabalho. Algumas ideias de trabalhos futuros que agregariam para a manutenção de um bom serviço seriam:

- Avaliação de usuários: Ferramenta muito difundida nesse tipo de aplicação, a avaliação entre os usuários é a melhor forma de outros participantes conhecerem um pouco uns aos outros.
- Punição de usuários: Percebeu-se durante a pesquisa em grupos de carona que pessoas que não cumprem com suas caronas, seja motorista, seja passageiro não recebem nenhum tipo de punição. Esse sistema de punição seria uma quantidade de tempo em que o usuário não poderia utilizar a aplicação, com o tempo de restrição aumentando conforme o número de punições.
- Compartilhar localização: Funcionalidade que agregaria na segurança. O usuário compartilhando a localização, outros usuários visualizariam a posição atual dos passageiros aumentando o nível de segurança.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] IDC, “Worldwide Smartphone Volumes Will Remain Down in 2018 Before Returning to Growth in 2019 and Beyond,” [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS43856818. [Acesso em 05 06 2018].

[2] F. G. d. A. COUTINHO, “Consumo Colaborativo: o compartilhamento de produtos e serviços que está modificando os negócios do Brasil,” 05 06 2018. [Online]. Available: http://portalintercom.org.br/anais/nacional2015/resumos/R10-0794-1.pdf.

[3] B. FLING, Mobile design and development: pratical concepts and techniques for creating mobile sites and web apps., O’Reilly Media, 2009.

[4] R. LECHETA, Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK 4ed, São Paulo: novatec, 2015.

[5] “Fossbytes,” [Online]. Available: https://fossbytes.com/most-popular-android-versions-always-updated/. [Acesso em 2018].

[6] R. BUDIU, “Mobile: native apps, web apps and hybrid apps.,” [Online]. Available: https://www.nngroup.com/articles/mobile-native-apps/. [Acesso em 01 12 2017].

[7] I. SOMMERVILLE, Engenharia de Software 9ed, São Paulo: pearson, 2011.

[8] E. BEZERRA, Principios de análise e projeto de sistemas com uml, São Paulo: Campus, 2007.

[9] R. ELMASRI e S. B. NAVATHE, Sistema de banco de dados 6ed, São Paulo: pearson, 2011.

[10] “thoughtco,” [Online]. Available: https://www.thoughtco.com/why-use-php-2694006. [Acesso em 2018].

[11] “Introdução PDO,” The PHP Group, [Online]. Available: http://php.net/manual/pt_BR/intro.pdo.php. [Acesso em 20 08 2018].

[12] “DB-Engines,” DB-Engines, 20 08 2018. [Online]. Available: https://db-engines.com/en/ranking. [Acesso em 20 08 2018].

[13] Firebase, “Firebase Analytics,” Google, 22 06 2018. [Online]. Available: https://firebase.google.com/docs/analytics/?hl=pt-br. [Acesso em 21 08 2018].

[14] “Rhodecode,” [Online]. Available: https://rhodecode.com/insights/version-control-systems-2016. [Acesso em 2018].

[15] “Github Android Crop,” [Online]. Available: https://github.com/jdamcd/android-crop. [Acesso em 25 01 2018].

[16] “Github Circle Image View,” [Online]. Available: https://github.com/hdodenhof/CircleImageView. [Acesso em 25 01 2018].

[17] “Github Universal Image Loader,” [Online]. Available: https://github.com/nostra13/Android-Universal-Image-Loader. [Acesso em 02 02 2018].

[18] “Android SharedPreferences,” Google, [Online]. Available: https://developer.android.com/reference/android/content/SharedPreferences. [Acesso em 10 09 2018].