
**Aplicativo híbrido para mototáxi com
recomendação de pilotos**

Hítalo Santos

S237 Santos, Hítnalo.
Aplicativo híbrido para mototáxi com recomendação de pilotos / Hítnalo Santos. – 2016.
54 f.

Orientador: Willyan Michel Ferreira.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)-Centro Universitário de Formiga-UNIFOR-MG, Formiga, 2016.

1. Cordova. 2. Geolocalização. 3. Transporte. I. Título.

CDD 005

PROJETO DE CONCLUSÃO DE CURSO

Data de Depósito: 03/11/2016

Assinatura: _____

Aplicativo híbrido para mototáxi com recomendação de pilotos

Hítalo Santos

Profº. Mestre Willyan Michel Ferreira

Monografia apresentada ao Instituto Superior de Educação do UNIFOR/MG, como requisito parcial para obtenção do título de bacharel em Ciência da Computação, sob a orientação do Profº. Mestre Willyan Michel Ferreira

Unifor-MG - Formiga

11/2016

Eu dedico este trabalho a todos que sempre me apoiam e confiaram na minha capacidade

”O futuro dependerá daquilo que fazemos no presente”

Mahatma Gandhi

Sumário

Lista de Figuras	ii
1 Introdução	1
1.1 Considerações Iniciais	1
1.2 Objetivos do Trabalho	2
1.3 Estrutura da Monografia	3
2 Referencial Teórico	5
2.1 Considerações Iniciais	5
2.2 Mototaxistas	5
2.3 Aplicações Híbridas	6
2.4 Java	6
2.5 Interface EntityManager	7
2.6 Intel XDK	7
2.7 Cordova	8
2.8 HTML	9
2.9 CSS	9
2.10 JavaScript	10
2.11 jQuery	10
2.12 AngularJS	10
2.13 Banco de dados MySQL	11
2.14 Webservices	11
2.15 Webservices RESTful	12
2.15.1 Principais comandos do RESTful	12
2.16 Padrão de comunicação JSON	13
2.17 Considerações Finais	13
3 Estudo da Arte	15
3.1 Considerações Iniciais	15

3.2	TaxiDroid	15
3.3	PedidosAki	16
3.4	Taxi Calculator	17
3.5	Traveller	18
3.6	Considerações Finais	19
4	Metodologia e Desenvolvimento	21
4.1	Considerações Iniciais	21
4.2	Comunicação	21
4.2.1	Descrição do Problema	21
4.3	Planejamento	24
4.4	Modelagem	25
4.5	Construção	30
4.5.1	Construção do Banco de Dados	30
4.5.2	Construção do Servidor RESTful	32
4.5.3	Construção do Aplicativo Móvel	35
4.6	Implantação	44
5	Testes e Resultados	45
6	Conclusões	47
6.1	Trabalhos Futuros	48
A	Questionário de Avaliação	53

Lista de Figuras

1.1	Arquitetura da aplicação	3
2.1	Estruturação de aplicativos híbridos - Fonte Mattje (2014)	6
2.2	Arquitetura de uso Cordova - Fonte Cordova (2016)	8
2.3	Exemplo de uso do JavaScript - Fonte Serra (2011)	10
3.1	Arquitetura geral do protótipo do TaxiDroid - Fonte Kuhn (2012)	15
3.2	Empresas disponíveis no PedidosAki - Fonte Fontana Junior (2013)	16
3.3	Central de pedidos do PedidosAki - Fonte Fontana Junior (2013)	17
3.4	Tela Inicial do Taxi Calculator - Fonte Vettorazzi e Schutz (2013)	18
3.5	Tela Inicial do Traveller - Fonte Melchior (2011)	19
4.1	Forma de pagamento aceita - Fonte Silva (2015)	22
4.2	Necessidade de Encontrar um Mototaxista	23
4.3	Cartões de alguns mototaxistas de Formiga - MG	24
4.4	Diagrama de Caso de Uso	25
4.5	Diagrama Entidade Relacionamento	26
4.6	Diagrama Classe dos Modelos de Entidades	28
4.7	Diagrama Classe dos Serviços disponíveis	29
4.8	Trigger que atualiza a pontuação do Mototaxista	30
4.9	Trigger que atualiza o tempo de resposta do Mototaxista	31
4.10	Exemplo de associação da coluna com a variável java	32
4.11	Classe ApplicationConfig	33
4.12	Busca baseada em JPQL, método contido em AbstractFacade	33
4.13	Querys JPQL da classe Corrida	33
4.14	Método GET para buscar corridas por período	34
4.15	Tela de Splash com nome do aplicativo	35
4.16	Tela de Login	36
4.17	Tela Criando Novo Login	36

4.18	HTML da tela Criando Novo Login	37
4.19	Tela Inicial do Cliente	37
4.20	Tela de Detalhes do Mototaxista	38
4.21	Buscando Geolocalização	38
4.22	Confirmação do Chamado	39
4.23	Notificação <i>push</i> da resposta	39
4.24	Tela de Notificações do Cliente	40
4.25	<i>Popup</i> de avaliação de corrida	41
4.26	Histórico de Corridas do Cliente	41
4.27	Menu Lateral do Mototaxista	42
4.28	Tela inicial do Mototaxista	42
4.29	Notificação <i>push</i> de Corrida	43
4.30	Tela de Histórico do Mototaxista	43

Lista de Abreviaturas e Siglas

ADT: Android Development Tools
API: Application Programming Interface
ASP: Active Server Pages
CRUD: Create Read Update Delete
CSS: Cascading Style Sheets
DER: Diagrama Entidade Relacionamento
GPS: Global Position System
HTML: HyperText Markup Language
HTTP: HyperText Transfer Protocol
IDE: Integrated Development Environment
JDK: Java Development Kit
JPQL: Java Persistence Query Language
JRE: Java Runtime Environment
JSE: Java Standard Edition
JSON: JavaScript Object Notation
JVM: Java Virtual Machine
MVC: Model View Controller
REST: Representational State Transfer
SDK: Software Development Kit
SMS: Short Message Service
SQL: Structured Query Language
USB: Universal Serial Bus
XML: eXtensible Markup Language
YAML: Yet Another Markup Language

Resumo

Santos, H. *Aplicativo híbrido para mototáxi com recomendação de pilotos*. Monografia (Graduação) — Centro Universitário de Formiga – Unifor-mg – Formiga, 2016.

Como consequência da precariedade de alguns transportes coletivos nas cidades brasileiras, as pessoas estão optando por meios de locomoção alternativos como os mototáxis, pois estes estão em uma linha mediana entre o baixo custo dos ônibus públicos e o conforto dos táxis e se destacam também pela rapidez no deslocamento. Este trabalho tem como objetivo a criação de um aplicativo móvel em plataforma *web* utilizando Cordova que pode ser compilado para vários sistemas operacionais, este software propõe ajudar os passageiros de mototáxis a encontrarem pilotos de acordo com as recomendações de outros usuários. Os métodos usados para que tal comunicação se estabeleça foram: a criação de um aplicativo *mobile* para os clientes e mototaxistas se cadastrarem e o desenvolvimento de um servidor *web* que faz a conexão entre os dois interessados. Os resultados obtidos foram uma comunicação com requisições entregues corretamente aos pilotos e passageiros, o gerenciamento de corridas dos mototaxistas podendo eles aceitarem ou não um chamado e a possibilidade de escolha entre pilotos com maior pontuação ou menor tempo de resposta por parte dos passageiros. Conclui-se que a comunicação entre o cliente e o mototaxista pode ser facilitada devido a criação de mais um meio de comunicação entre eles, as tecnologias de programação para aplicações híbridas foram mais eficazes na produção deste software visto que a digitação de código fonte é reduzida permitindo maior foco nas regras de negócio, o aplicativo poderá contribuir para o desenvolvimento da categoria dos mototaxistas e também poderá facilitar as requisições feitas pelos clientes.

Palavras-chave: Cordova, Geolocalização, Transporte, RESTful

Abstract

Santos, H. *Hybrid application for motorcycle taxi with riders recommendation.* Monograph (Graduation) — Centro Universitário de Formiga – Unifor-mg – Formiga-MG, 2016.

As a result of the precariousness of some public transport in Brazilian cities, people are opting for alternative means of locomotion as the motorcycle taxis because they are in a median line between the low cost of public buses and taxis comfort are also by the faster displacement. This work has as objective the creation of a mobile application in web platform using Cordova that can be compiled for various operating systems, but with an emphasis on the Android platform, this software proposes help passengers of taxis to find pilots in accordance with the recommendations of other users. The methods used for such communication was established, creating a mobile application for customers and motorcycle taxi drivers sign up and the development of a web server makes the connection between the two. The results were a communication with answers in seconds and that did not change the performance of the devices tested visually, managing the racing motorcycle taxi and may they accept or not a calling and a choice between top scoring pilots or less response time on the part of the passengers. It is concluded that communication between the client and the motorcycle taxi driver might be facilitated due to creation of another means of communication between them, programming technologies for hybrid applications were more effective in producing this software since entering source code is reduced allowing greater focus on business rules, the application can contribute to the development of the category of motorcycle taxi drivers and can also facilitate requests made by customers.

Keywords: Cordova, Geolocation, Transport, RESTful

Introdução

1.1 Considerações Iniciais

A utilização dos *smartphones* em segmentos de educação, trabalho, lazer, entre outros, vem alcançando novos patamares de uso, ele não é mais um aparelho usado apenas para ligar para outras pessoas, o que aliás já pode ser substituído por alguns aplicativos que usam internet.

Segundo pesquisas efetuadas com pessoas entre 16 e 25 anos, 73% dos participantes têm o aparelho, entre os homens de 16 a 17 anos 55%, entre as mulheres na mesma faixa de idade, as que possuem *smartphone* chega a 73%. 94% das pessoas entrevistadas que têm ensino superior e 90% dos que têm renda familiar mensal acima de 10 salários mínimos possuem o aparelho. O percentual de posse de telefone é maior entre os solteiros sendo 74%, do que entre as pessoas casadas 68%. (DATAFOLHA, 2008).

A todo momento surgem novos *softwares* para auxiliar as pessoas nas tarefas cotidianas como: calculadora, mapas, calendários, transações bancárias, guias culinários, professores virtuais, entre outros. Os *smartphones* têm uma presença perceptível na vida das pessoas, pois eles facilitam a interação entre indivíduos. A função de um *smartphone* de efetuar ligações é mais uma entre as diversas possibilidades disponíveis nele.

Aparelhos que antes ocupavam espaços físicos nas casas, hoje podem ser substituídos por um único dispositivo. Câmera fotográfica, telefone fixo, lanterna, calculadora, gravador de voz, rádio, despertador e videogame portátil são alguns exemplos.

Aplicativos de mapas por exemplo, indicam as rotas com melhor caminho onde são avaliados menor distância, pedágios, segurança do local, entre outros fatores. (GOOGLE, 2015). Não se pode ignorar o fato de que nem sempre o aplicativo *Global Position System* (GPS) está totalmente correto, podendo te guiar para bairros perigosos, mas é uma grande ajuda para quem vai de veículo próprio a alguma localidade que não conhece.

Quando não se tem um veículo, uma alternativa é esperar até que um ônibus ou metrô

passe, o que muitas vezes pode demorar, além disso, dentro desse tipo de condução existe a possibilidade de não ter poltronas disponíveis para todos e a segurança e o conforto nem sempre são eficientes. Estar nessa situação pode ser mais complicado quando se utiliza dois ou três ônibus por dia.

Outra alternativa para quem tem mais pressa e pode pagar um pouco mais é chamar um taxista. Às vezes cobram taxas abusivas das pessoas que não têm tanta noção dos preços, não prestam um serviço de qualidade e tendem a demorar mais para completar o trajeto por serem veículos maiores e mais pesados.

Existem também como escolha para se locomover de modo rápido e barato, os mototaxistas. Essa classe de trabalhadores está crescendo muito nas cidades pequenas e também longe dos grandes centros. (ABREU, 2012).

É notável a facilidade de deslocamento e rapidez de uma moto, por ser um veículo relativamente pequeno, econômico e leve, consegue percorrer por ruas e avenidas sem muitos problemas. Quanto aos preços e taxas, podem sair bem em conta se comparados aos dos táxis. O problema que geralmente encontra-se nas motos são as bagagens, não dão tanto suporte assim para malas e bolsas grandes.

Perante este cenário, muitas pessoas optam pelas motocicletas. As dúvidas surgem no momento em que se quer alguém de confiança e que faça todo o trajeto de modo seguro. Quando encontra-se um mototaxista que presta os serviços de modo idôneo e com preço razoável todos querem manter o contato com ele para transportes futuros.

O problema é quando não se tem referências de pilotos capacitados, indicações de usuários que já passaram pela experiência são uma saída para fazer a escolha mais rapidamente, pode-se pegar cartões, anotar o número no celular ou então memorizar.

1.2 Objetivos do Trabalho

O objetivo deste trabalho foi tentar reduzir as dificuldades de comunicação entre os usuários de mototáxi descritas anteriormente, provendo ou melhorando a comunicação entre o cliente que necessita de algum tipo de serviço prestado pelos mototaxistas e os trabalhadores da categoria. A proposta para que tal objetivo fosse alcançado foi abordar o desenvolvimento de um aplicativo para dispositivos móveis que possibilite a solicitação de corridas ou serviços em geral aos mototaxistas, permitindo ao cliente a escolha entre pilotos com maiores classificações.

Voltado para os mototaxistas, os objetivos foram: criar um ambiente para que ele possa gerenciar todos os chamados de maneira simplificada dando-lhe a opção de aceitar ou não um chamado para corrida. O piloto poderá visualizar todo histórico de corridas direcionadas a ele, inclusive as corridas que não aceitou.

A figura 1.1 mostra a arquitetura da aplicação na qual é possível observar que o servidor faz a ligação entre o cliente e o mototaxista através do serviço em nuvem. O servidor recebe as informações de cada requisição feita pelos usuários, trabalha em cima das regras de negócio em conjunto com o banco de dados e faz o retorno contendo os dados de resposta.

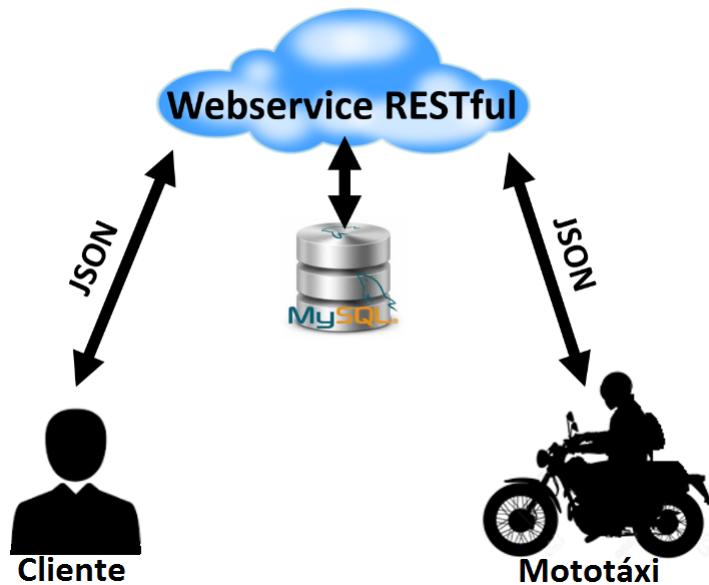


Figura 1.1: Arquitetura da aplicação

1.3 Estrutura da Monografia

Esta monografia está estruturada da seguinte maneira, o capítulo 1 descreveu a introdução com as considerações iniciais e os objetivos do trabalho. O capítulo 2 apresenta o referencial teórico contendo os conceitos e tecnologias abordadas para este desenvolvimento. O capítulo 3 contém o estudo da arte que serviu de base para este projeto. O capítulo 4 mostra a metodologia utilizada e o desenvolvimento da aplicação. No capítulo 5 são apresentados os testes feitos e os resultados extraídos e por fim, o capítulo 6 contém as conclusões, as contribuições e os trabalhos futuros.

Referencial Teórico

2.1 Considerações Iniciais

Nesta seção serão apresentadas as ferramentas, necessidades de mercado, *softwares* e padrões que compõem a base de conhecimento utilizada para efetuar a modelagem, planejamento e construção do *software*. Algumas destas ferramentas foram substituídas durante o andamento do desenvolvimento visto que o projeto se adaptou melhor às tecnologias substitutas.

2.2 Mototaxistas

Atualmente, há muitos profissionais conhecidos como *motoboys*, eles que ficam muito tempo ociosos durante o período de trabalho aguardando por uma chamada para efetuar algum serviço de entrega ou coleta de documentos ou produtos para algum cliente. (SASAKI, 2012).

A indústria automotiva iniciou suas atividades no Brasil em 1956, mas até 1960 as motocicletas ainda não tinham mercado, a partir de 1990, com os processos de liberação econômica, as motocicletas começaram a tomar seu espaço. "Algumas motocicletas eram fabricadas no país e outras eram importadas dos EUA ou do Japão. Elas eram usadas principalmente pelas pessoas de renda mais alta, por motivo de lazer." Uma grande parte das motocicletas vendidas atualmente é usada para fazer entregas rápidas de pequenas mercadorias, os *motoboys* surgiram a partir da falta de oportunidade de emprego para jovens com baixa escolaridade. (VASCONCELLOS, 2008).

Segundo Vasconcellos (2008), "Outra parte das motocicletas têm sido usadas como mototáxis, legal e ilegalmente, para transportar passageiros". Usando como diretriz os transportes coletivos ilegais como vans, perus e táxis não regulamentados, os mototaxistas encontraram nas ruas um mercado pouco explorado e que poderia render algum lucro.

2.3 Aplicações Híbridas

Devido ao tipo de programação utilizada para desenvolver aplicativos nativos como Android, IOS e Windows Phone é necessário que haja uma compilação em cada uma destas plataformas. Um aplicativo desenvolvido de forma híbrida utiliza linguagens que podem ser interpretadas pelo navegador, isso possibilita sua execução em diferentes plataformas. (MATTJE, 2014).

Os aplicativos híbridos possuem uma limitação de acesso aos recursos de hardware, o sistema operacional ainda não pode ser acessado diretamente por eles e possuem alguns problemas de compatibilidade entre os navegadores. Apesar disso, a cada novo lançamento estes problemas são amenizados tornado-os então uma opção de tecnologia nas empresas. (MATTJE, 2014).

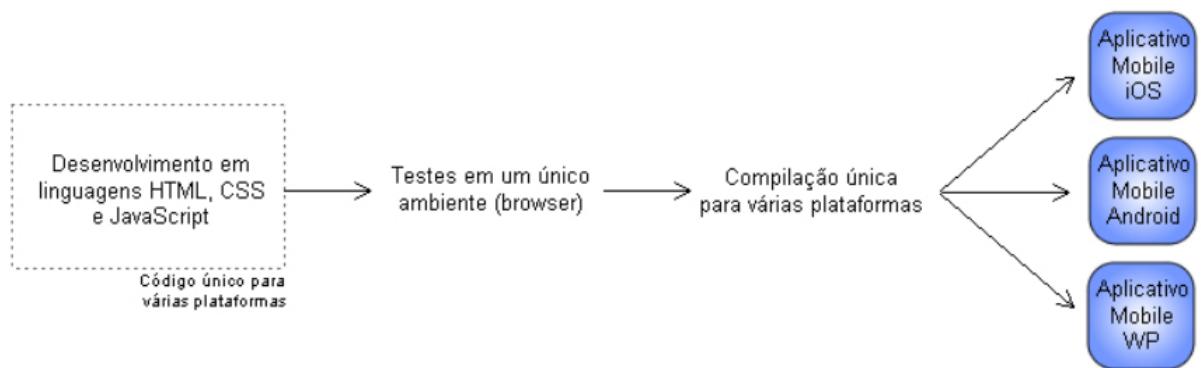


Figura 2.1: Estruturação de aplicativos híbridos - Fonte Mattje (2014)

A figura 2.1 mostra a estrutura de desenvolvimento de aplicações híbridas as quais utilizam linguagens de desenvolvimento *web* possibilitando os teste no navegador. Após a construção, é possível gerar instaladores nativos para várias plataformas.

As ferramentas *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS), Javascript, AngularJS, Ionic e CORDOVA podem conceder aplicações para facilitar o dia a dia dos seus usuários. Será mostrado a seguir, algumas dessas tecnologias que foram empregadas neste projeto.

2.4 Java

Java foi lançada pela Sun Microsystems em 1995 sendo uma linguagem de programação e plataforma computacional que hoje pertence a Oracle. Existem várias aplicações e sites que dependem de Java instalado na máquina para funcionarem e mais desses são projetados todos os dias. Java tem como características a rapidez, segurança e confiabilidade estando em vários lugares como notebooks, celulares, vídeo games, sites a supercomputadores científicos. (ORACLE, 2016)

“O Java é a base para praticamente todos os tipos de aplicações em rede e é o padrão global para o desenvolvimento e distribuição de aplicações móveis e incorporadas, jogos, conteúdo baseado na Web e softwares corporativos”. O Java foi feito para a mais ampla variedade possível de plataformas de computação para permitir o desenvolvimento de aplicações portáteis

com um desempenho superior. Quando as aplicações são feitas em ambientes heterogêneos, as empresas podem aumentar a produtividade, fornecer mais serviços e melhorar a comunicação com o usuário final. (ORACLE, 2016).

O Java tornou-se fundamental para os desenvolvedores, permitindo que eles escrevam um *software* em uma plataforma e o executem em qualquer outra plataforma por meio da máquina virtual Java (JVM), crie *softwares* robustos e eficientes para smartphones, gateways, microcontroladores, processadores remotos, sensores, módulos sem fio e praticamente qualquer outro aparelho eletrônico. A Oracle tem dois produtos que implementam *Java Standard Edition* (JSE) que são Java SE Development Kit (JDK) e Java SE Runtime Environment (JRE). O JRE fornece as bibliotecas, a JVM, e outros módulos necessários para executar *softwares* escritos em Java, já o JDK contém tudo que está na JRE e mais algumas ferramentas, como os compiladores e depuradores necessários para o desenvolvimento de aplicativos Java. (ORACLE, 2016).

2.5 Interface EntityManager

A EntityManager é usada no processo de persistência de dados de um contexto dentro de uma aplicação. Cada um destes contextos é dado como um conjunto de instâncias de entidades, cada instância referencia a uma única entidade. Os ciclos de vida das entidades são geridos dentro do contexto de instâncias e podem ser criados ou removidos pela EntityManager de acordo com seu identificador.

A unidade de persistência descreve o grupo de todas as classes que são usadas na aplicação e que devem ser mapeadas de acordo com o banco de dados. Dentro deste cenário de persistência existe a *Java Persistence Query Language* (JPQL) que é uma linguagem de consulta dinâmica usada para buscar ou enviar dados ao banco de acordo com os atributos das entidades. (ORACLE, 2016).

2.6 Intel XDK

O Intel XDK é um *Integrated Development Environment* (IDE) que combina tecnologias voltadas para o desenvolvimento *web* como por exemplo HTML, CSS, JavaScript e Cordova apresentando um ambiente em que é possível desenvolver usando o modo de clicar e arrastar para a geração das telas ou via código fonte. O modo de código permite alterar vários atributos dos componentes como cor, largura, responsividade ou ações de acordo com a necessidade da aplicação. Este IDE permite também emular o aplicativo em desenvolvimento disponibilizando vários dispositivos virtuais com diferentes tamanhos de telas. Usando-se as tecnologias de programação para *web*, é possível construir os módulos do *software* com uma programação mais intuitiva e compacta. (INTEL, 2016).

Como o Intel XDK possui as ferramentas necessárias para se trabalhar com aplicações híbridas que podem ser executadas em vários sistemas operacionais, ele foi usado para a construção da aplicação móvel.

2.7 Cordova

O Apache Cordova disponibiliza um conjunto de APIs que auxiliam no desenvolvimento de uma aplicação com HTML, CSS e JavaScript gerando um pacote de instalação de aplicação móvel nativa. A aplicação é executada no aparelho móvel e tem acesso às funcionalidades nativas do dispositivo, como Giroscópio, GPS ou câmera. (NETBEANS, 2016).

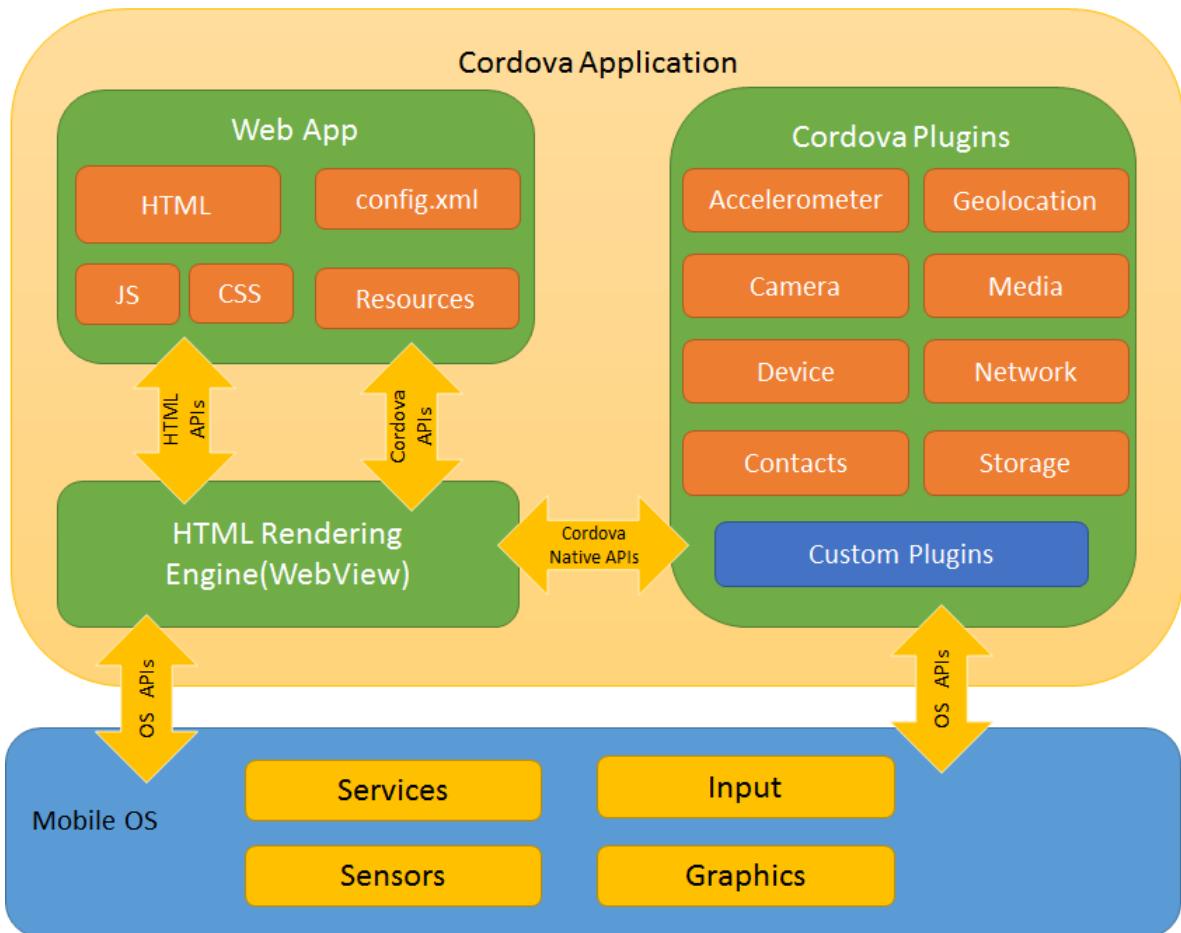


Figura 2.2: Arquitetura de uso Cordova - Fonte Cordova (2016)

A figura 2.2 mostra como funciona a estrutura de uma aplicação que trabalha com Apache Cordova. Todo o código fonte da aplicação fica localizado em *web App*. A aplicação é desenvolvida como se fosse uma página da *web*, como padrão, tem um arquivo principal que funciona como um ponto de partida para iniciar o aplicativo, chamado *index.html*, nele contém os links e caminhos das bibliotecas de JavaScript, CSS, AngularJS e todos os outros recursos necessários para que tudo funcione.

O *software* é executado dentro de um *WebView* que por sua vez, fica dentro de um aplicativo nativo o qual pode-se gerar, por exemplo, um pacote de instalação do Android. Este aplicativo nativo, funciona como um recipiente e tem um arquivo chamado *config.xml*, é este XML que fornece dados sobre o aplicativo e dita os parâmetros que afetarão o funcionamento do *software*, por exemplo, uma mudança de orientação. O *WebView* do Cordova fornece toda a interface de usuário da aplicação e pode também, estar dentro de outro aplicativo maior como se fosse um módulo de um *software* nativo.(CORDOVA, 2016).

O Cordova contém um núcleo de *plugins* que oferece ao aplicativo acesso aos recursos nativos do aparelho, como giroscópio, GPS, câmera ou contatos. Os *plugins* são muito importantes dentro deste contexto do Cordova. São eles que fornecem as conexões entre o Cordova e os componentes nativos provendo a comunicação entre eles. Isso permite que o desenvolvedor chame uma função nativa através de JavaScript. Além dos *plugins* deste núcleo, existem também os *plugins* de terceiros que fornecem conexões para componentes que não fazem parte dos padrões de todas as plataformas, ou seja, os componentes personalizados. (CORDOVA, 2016).

2.8 HTML

HTML é uma linguagem de marcação de hipertexto que pode ser resumida como algum conteúdo que está alocado dentro de uma página da *web* e que tem como característica principal a alternativa de se conectar com outras páginas *web*. (SILVA, 2014).

Segundo Serra (2011) o HTML faz uso de *tags* que são marcadores que descrevem uma página web. Essas *tags* são elementos descritos entre <>, que posteriormente, são interpretados pelos browsers que exibem as páginas formatadas com elementos visuais, essas páginas, podem conter vários componentes de mídia como textos, vídeos, imagens ou áudios. O HTML foi desenvolvido por Tim Berners-Lee em 1989 no *European Organization for Nuclear Research*, também conhecido por CERN.

O aparecimento do HTML5 alterou muitos quesitos no mundo do desenvolvimento para web, com novos componentes, novas funcionalidades e várias outras novidades que permitem melhor usabilidade e interações que antes eram apenas expectativas de alguns programadores desta área. Mesmo com algumas dessas novas funções ainda em processo de determinação, já se pode aproveitar diversas novidades que o HTML5 trouxe consigo. (MAZZA, 2014).

O HTML5 sozinho, não pode controlar uma aplicações *web*, como é apenas para marcação dos elementos de uma página precisa do JavaScript e do CSS para implementar a apresentação e interação do *front-end*. (SERRA, 2011)

2.9 CSS

O CSS é uma linguagem aplicada na formatação, apresentação e padronização de componentes de alguma página escrita em HTML. Ele viabiliza a organização entre os conteúdos das páginas. Assim como o HTML, ele não necessita de nenhum outro *software* além do navegador para ser utilizado. (SERRA, 2011)

Com o surgimento do CSS3, vários padrões e imagens que antes eram usados para adicionar cores e formas aos componentes HTML puderam ser substituídos, o que não era possível apenas com CSS. Outro fator importante foi que os tamanhos dos documentos HTML tiveram uma redução significativa visto que menos códigos eram escritos. O desprendimento de dependências externas elevou o desempenho dos sites e a flexibilidade de uso das propriedades permite várias combinações de estilos diferentes. (MAZZA, 2014).

2.10 JavaScript

Segundo Serra (2011) a linguagem JavaScript é multi-plataforma, orientada a objetos e geralmente é utilizada com alguma outra tecnologia, como HTML5 nos casos de aplicações *web*. Ele implementa um grupo de objetos como Dates, Arrays, operadores, estruturas de repetição e de controles de fluxo. O JavaScript é inspirado em ECMAScript e normalmente é usado dentro das aplicações do lado do cliente em validações de formulários e *inputs*, mas também pode ser usado nos servidores como manipulador de ficheiros ou como conexão com bases de dados.

As qualidades mais marcantes dessa linguagem são sua sintaxe estruturada e dinamicidade dos objetos pois não se associam às variáveis e sim aos seus valores. As funções são consideradas como objetos e podem ser usadas como variáveis, passadas como parâmetros ou retornadas de alguma outra função. (SERRA, 2011).



Figura 2.3: Exemplo de uso do JavaScript - Fonte Serra (2011)

Como pode ser visto na figura 2.3 o JavaScript é uma linguagem que normalmente é usada em conjunto com HTML e CSS, sendo ele, a parte de regras de negócio, interação e finalização de *design*.

2.11 jQuery

O jQuery é um *framework* que facilita a maneira de programar utilizando a linguagem JavaScript. Ele é compatível com a maioria dos navegadores e é capaz de gerar animações, manipular HTML e eventos. Sendo um dos pioneiros a trabalhar com JavaScript ele revolucionou a maneira de programadores usarem esta linguagem. (JQUERY, 2016).

2.12 AngularJS

Como dito por Branas (2014) o AngularJS é um *framework open source* usado na criação de projetos *web* utilizando a linguagem de programação JavaScript. Ele permite fazer a estruturação da aplicação em camadas melhores definidas, a criação de componentes reutilizáveis e já possui toda a estrutura necessária para integrar-se com *back-end* via Ajax não necessitando de alguma outra biblioteca para isso.

Foi criado em 2009 por Misko Hevery e Adams Abrons com a intenção de facilitar a criação de aplicações *web* focando a redução de código fonte e melhorias na produtividade. Hoje em dia, o AngularJS é mantido pelo Google e tem uma grande comunidade de desenvolvedores e

fóruns. Um ponto forte do AngularJS é o *Two-way Data Binding* que capta qualquer alteração de dados feita na variável `$scope` (versão 1.x) a qual faz a ligação entre a *view* e o *controller* da aplicação facilitando a mudança de estado dos componentes.

2.13 Banco de dados MySQL

Um banco de dados é um conjunto de informações estruturadas, ele pode ser desde uma lista de compras até uma grande quantidade de informações de negócios de uma empresa. Para acessar, editar, remover ou processar informações de um banco em alguma máquina, há necessidade de um *software* que gerencie todas essas requisições como o Servidor MySQL. Os computadores são bons em processar uma grande massa de dados, nesse ambiente, o gerenciamento de bancos se torna uma peça fundamental para que tudo funcione corretamente. (SUEHRING, 2002).

O banco de dados a ser usado para armazenar todas as informações de corridas e cadastros será o MySQL. Suehring (2002) disse que o MySQL é um servidor de banco de dados *Structured Query Language* (SQL) multiusuário, multitarefas, robusto e muito rápido. ”MySQL, o mais popular sistema de gerenciamento de banco de dados SQL *Open Source*, é desenvolvido, distribuído e tem suporte da MySQL AB”. O MySQL é um banco de dados relacional, ou seja, ele armazena os dados em tabelas separadas e depois as relaciona quando necessário proporcionando uma maior velocidade e flexibilidade. Ele é *Open Source*, isso quer dizer que qualquer pessoa pode usá-lo sem pagar nada.

”O Servidor MySQL foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida”. (SUEHRING, 2002). Segundo Lecheta (2015), ”O MySQL é um banco de dados relacional bastante popular no mundo inteiro[...]”.

2.14 Webservices

Os *webservices* são utilizados para fazer a integração e troca de mensagens entre sistemas, podendo estes, realizar chamadas de outros sistemas a fim de obter dados. Os dois formatos mais conhecidos de troca de informações são *eXtensible Markup Language* (XML) e o *JavaScript Object Notation* (JSON). Os *webservices* permitem acessar serviços de uma forma padronizada independente da linguagem o que acaba sendo uma grande vantagem, um exemplo disso é que se o seu programa for escrito em Java, ele pode trocar informações com sistemas feitos em C#, PHP, Python ou qualquer outra linguagem. SOAP e REST são duas das várias formas de criar *webservices* e também são as mais usadas. (LECHETA, 2015).

Loudon (2010) disse que a internet que conhecemos hoje é muito diferente daquela de dez anos atrás pois a complexidade das aplicações *Web* aumentou muito e as mudanças acontecem a todo momento, isso dificulta muito a manutenção das mesmas. Ele cita sete boas práticas de desenvolvimento que têm que ser observadas durante o ciclo de vida das aplicações.

Disponibilidade contínua: sugere que a maior parte das grandes aplicações *Web* devem rodar o tempo todo, sem pausas. Além disso, todas as repostas têm quer ser em tempo real

e a todo momento, mesmo nos instantes em que está ocorrendo vários acessos simultâneos. Desenvolvedores *Web* têm que projetar sistemas que sejam muito robustos. (LOUDON, 2010).

Grande base de usuários: o autor aponta que "Grandes aplicações *web* normalmente apresentam volumes impressionantes de usuários. Isso requer o gerenciamento de um grande número de conexões simultâneas ou de várias camadas de *caching*". (LOUDON, 2010).

Entrega em partes: ao contrário de aplicações que são entregues como um pacote fechado de módulos, aplicações *Web* muitas vezes são entregues módulo a módulo e mantendo sempre a disponibilidade dos serviços que já foram entregues. (LOUDON, 2010).

Diversidade: como existem várias possibilidades de desenvolvimento como aplicações financeiras, mecanismos de busca ou site de notícias, os desenvolvedores devem se preocupar em escrever códigos genéricos que possam ser reutilizados em várias situações diferentes. (LOUDON, 2010).

Longevidade: mesmo as aplicações *web* que já existem há anos ainda estão no começo de seu ciclo de vida. Desenvolvedores *Web* têm de escrever códigos que sejam capazes de se manter aplicáveis mesmo após anos de alterações e manutenção. (LOUDON, 2010).

Múltiplos ambientes: a *Web* é um ambiente muito heterogêneo e com mudanças rápidas, existem vários navegadores e dispositivos diferentes que não oferecem tanto suporte. Usuários buscam várias aplicações a partir de inúmeros ambientes e com todo tamanho de tela inclusive smartphones. Os desenvolvedores devem se preocupar em tratar o maior número possível de situações diferentes. (LOUDON, 2010).

Atualizações em tempo real: as grandes aplicações *Web* estão o tempo todo sofrendo modificações e evoluindo, a todo momento, várias alterações são enviadas aos servidores, cada módulo deve acompanhar essas mudanças. (LOUDON, 2010).

2.15 Webservices RESTful

Representational State Transfer - Transferência de Estado Representacional (REST) é um conjunto de técnicas e padrões para desenvolver *webservices* com a maioria dos conceitos baseados nos métodos *GET*, *POST*, *PUT*, e *DELETE* que são do protocolo *HyperText Transfer Protocol* (HTTP). O RESTful nada mais é do que os *webservices* que são criados seguindo este padrão. (LECHETA, 2015).

2.15.1 Principais comandos do RESTful

No REST o método *GET* é normalmente usado para fazer consultas e requisições que usam poucos parâmetros, uma requisição do tipo *POST* é geralmente usada caso o objetivo seja inserir um novo registro. De acordo com as boas práticas de programação do REST, deve-se utilizar o *PUT* quando pretende-se atualizar algum dado e para finalizar o *Create, Read, Update e Delete* - Criar, Pesquisar, Atualizar e Remover (CRUD) deste padrão de desenvolvimento, usa-se o *DELETE* quando se quer remover registros.

2.16 Padrão de comunicação JSON

O JSON representa a estrutura de dados de um objeto JavaScript. O seu formato é mais simples e leve se comparado com o XML, por isso vem substituindo-o como forma de troca de informações entre sistemas. (LECHETA, 2015).

Fonseca e Simões (2007) apontaram nos seus estudos que nos últimos anos o *eXtensible Markup Language* (XML) tem tomado conta do mercado em várias áreas desde finanças a agricultura. Ele tem sido a forma de comunicação escolhida na maior parte das aplicações. Jogadores salvam seus progressos em XML, torcedores recebem resultados do seu time via XML, são muitas aplicações que usam esta tecnologia. O XML é uma das sintaxes mais robustas e confiáveis já criadas, mas também tem seus pontos fracos.

A sua sintaxe é cheia de palavras redundantes o que pode afetar o entendimento humano e obriga as aplicações a terem um espaço de armazenamento a mais, este fator também atrapalha quando a largura de banda de transmissão é baixa. Construir um parágrafo em XML não é tão simples, tem que se colocar vários elementos aninhados e fazer testes para detectar dados mal formatados. O XML não possui tipos de dados complexos como número de ponto flutuante, por exemplo, ele reconhece 3.14159 como uma palavra de sete caracteres e não um número. O conjunto de caracteres usados para escrever uma sentença XML são particularmente difíceis de se usar em um teclado convencional, isso dificulta a rápida criação de documentos. O uso de editores próprios pode amenizar este problema. Existem várias tecnologias no mercado aptas a substituir o XML, mas duas delas têm se destacado das demais, a *Yet Another Markup Language* (YAML) e o JSON. (FONSECA; SIMÕES, 2007).

O JSON consegue codificar e decodificar quatro tipos primários, sendo eles, strings, números, booleanos e nulos e também objetos e vetores. Cada objeto é uma coleção de zero ou mais pares representando nome/valor de forma não ordenada sendo, o nome uma string e o valor qualquer um dos quatro tipos primários citados anteriormente ou um objeto ou um vetor, "Assim o JSON foi desenhado com o objetivo de ser simples, portável, textual, e um subconjunto do JavaScript". (FONSECA; SIMÕES, 2007).

2.17 Considerações Finais

As tecnologias citadas anteriormente foram entendidas para formar o conhecimento básico sobre o assunto de aplicações móveis e as ferramentas necessárias para conectar dispositivos via servidor. Para este projeto, as ferramentas utilizadas na construção de aplicações híbridas se mostraram mais eficazes do que as de aplicações nativas enquanto desenvolvimento, elas foram combinadas e podem ajudar os mototaxistas a amenizar a questão de comunicação com seus clientes permitindo assim atenderem o maior número de passageiros e da forma mais eficiente possível.

Estudo da Arte

3.1 Considerações Iniciais

Nesta seção serão apresentados alguns trabalhos que de certa forma estão relacionados com os assuntos ou tecnologias deste projeto bem como as ferramentas de apoio e métodos usados pelos autores, também serão apresentados os resultados obtidos em tais estudos.

3.2 TaxiDroid

Alguns problemas atingem o sistema de táxi no Brasil. Os centros de atendimento ao usuário podem estar saturados por causa das inúmeras chamadas e as entregas das solicitações aos taxistas podem não estar atingindo um patamar otimizado. Estes problemas citados podem causar insatisfação dos usuários. A solução proposta pelo autor é, automatizar o serviço de solicitação de táxis, otimizando todo o processo. Seu intuito é amenizar o peso das centrais telefônicas, reduzindo então, o tempo gasto para solicitar uma corrida e diminuindo a espera por parte dos passageiros. (KUHN, 2012).

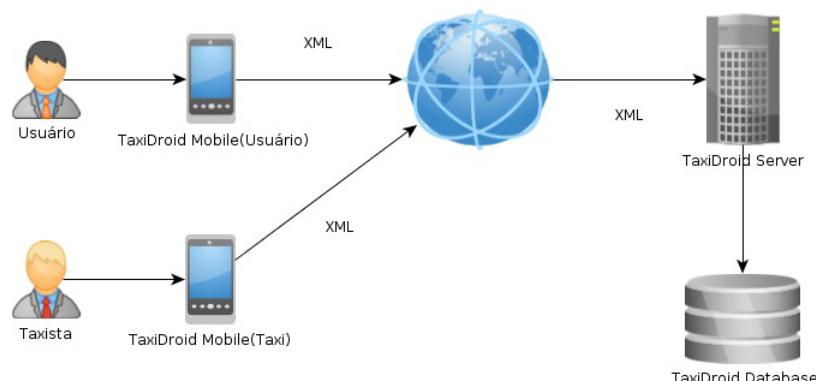


Figura 3.1: Arquitetura geral do protótipo do TaxiDroid - Fonte Kuhn (2012)

A figura 3.1 mostra o protótipo de funcionamento do projeto que irá gerenciar as solicitações de corridas aos taxistas. A solicitação sai do celular do cliente via XML, navega pela internet até o servidor que gerencia todos os dados da aplicação e chaga até o taxista com as informações de endereço e localização do GPS.

Os objetivos a serem alcançados para prover tais soluções, estão apontados para a criação de uma plataforma automática onde, o cliente solicita um serviço de táxi sem passar pela central telefônica, consegue acompanhar o status da chamada e o táxi é despachado automaticamente. A plataforma utilizada foi o Android. O autor concluiu que os objetivos foram alcançados, com a criação de um aplicativo feito em Android, foi possível automatizar a solicitação e despacho de táxis através do smartphone bem como a busca de taxistas mais próximos ao solicitante da corrida. (KUHN, 2012).

3.3 PedidosAki

Os estabelecimentos voltados a área alimentícia se mostram mais presentes atualmente, mas mesmo assim não conseguem atender todos os seus clientes de forma satisfatória. Apesar das várias novas empresas deste ramo, os clientes, por sua vez, nem sempre conseguem opções variadas para solicitar o seu pedido como por exemplo smartphone ou computador, tendo que enfrentar linhas ocupadas ou trânsito. Os objetivos deste trabalho é criar um programa para Android usando Java onde o cliente pode solicitar os seus pedidos e um servidor *Web* usando *Active Server Pages* (ASP), .NET, *Model-View-Controller* (MVC) e C# para o gerenciamento da empresa fornecedora dos alimentos. (FONTANA JUNIOR, 2013).

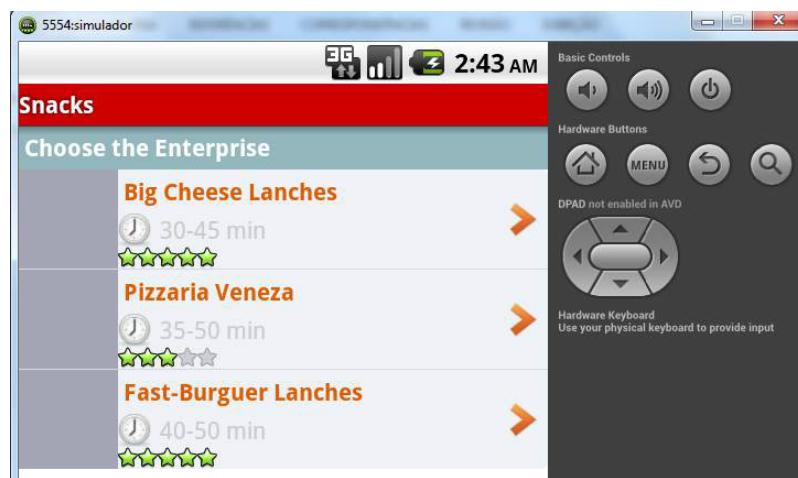


Figura 3.2: Empresas disponíveis no PedidosAki - Fonte Fontana Junior (2013)

A figura 3.2 exibe a apresentação das empresas disponíveis para a realização de pedidos próximas ao local em que o cliente está. Quando selecionada, exibe as opções de compras existentes no cardápio permitindo efetuar o pedido.

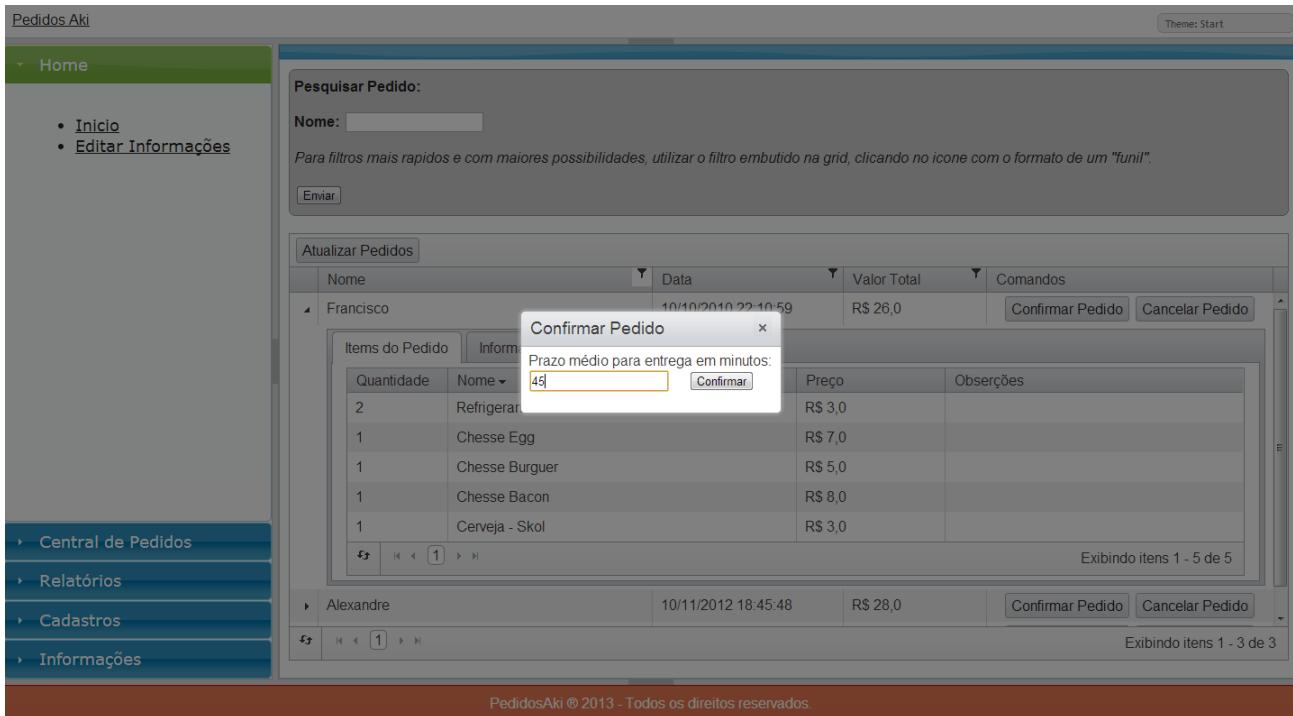


Figura 3.3: Central de pedidos do PedidosAki - Fonte Fontana Junior (2013)

A figura 3.3 mostra a central de gerenciamento de pedidos que os estabelecimentos irão usar, nele é possível aceitar ou recusar algum pedido feito pelo cliente e informar o tempo em minutos para a entrega ser efetivada.

Para alcançar os objetivos propostos, foi necessário aprofundar os conhecimentos nas áreas Web e móvel. A parte em que se refere ao servidor Web demandou mais trabalho por falta de experiências anteriores ao projeto. Foram utilizados alguns componentes gráficos como Telerik e jQuery UI para aumentar a usabilidade do servidor. Os objetivos propostos no trabalho foram atingidos com máxima eficácia, mas deixando algumas ressalvas sobre a aplicação móvel. Não tem portabilidade para outros sistemas operacionais e quando está *off-line* fica totalmente fora de operação. (FONTANA JUNIOR, 2013).

3.4 Taxi Calculator

Os smartphones se mostram cada vez mais potentes o que levam os usuários ao consumismo. Os preços, cada vez mais baixos, atraem novas pessoas todo ano. O sistema operacional Android tem uma aplicação de mapas que é muito usada, o Google Maps, este possibilita conexão com GPS, permitindo o roteamento de viagens. Na cidade de Curitiba existem vários taxistas regulamentados que foram submetidos a uma vistoria devido a adulterações nos preços das corridas, as quais são tabeladas. A Secretaria de Segurança Pública apontou irregularidades como taxímetro em local pouco visível e que subia em 30% o valor da corrida, como o passageiro não podia se localizar, não se desconfiava do abuso. (VETTORAZZI; SCHUTZ, 2013).

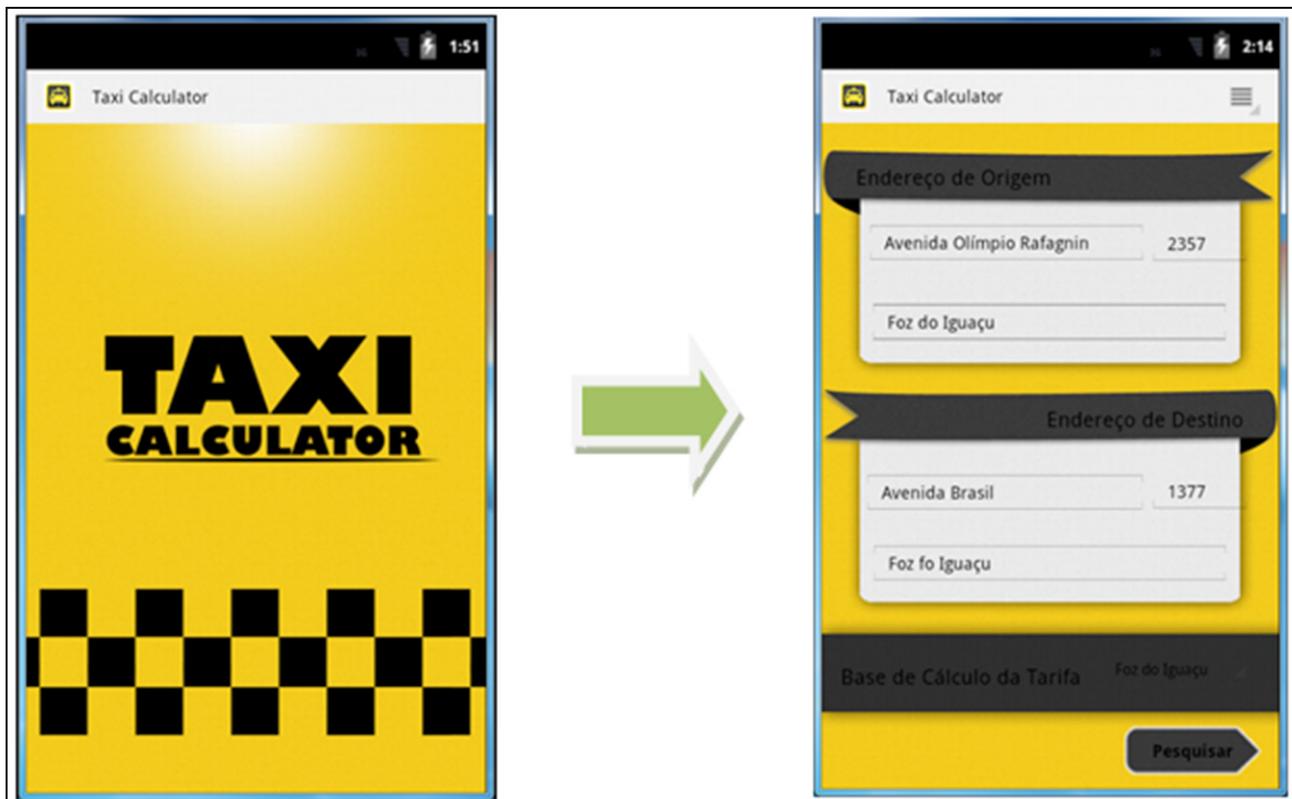


Figura 3.4: Tela Inicial do Taxi Calculator - Fonte Vettorazzi e Schutz (2013)

A figura 3.4 mostra a tela inicial do aplicativo em que contém os campos para informar os endereços de origem e destino, um menu para a seleção da tarifa base e outro que permite exibir os pontos de táxis que se encontram mais próximos da localidade do cliente.

Os objetivos deste trabalho visam a criação de um software de controle de rotas de táxis baseado no Google Maps onde, é possível calcular o valor da corrida de acordo com o ponto de origem e chegada e um *web service* de gestão das informações processadas. As ferramentas utilizadas foram a API Google Maps, IDE Eclipse SDK versão Indigo Release 2, ambos usando a versão 7 do JDK e banco de dados SQLite Expert. Concluiu-se que, A API de mapas do Google mostrou-se eficiente ao buscar locais e formar trajetos através do *Google Directions* e que também foi de fácil aprendizado pois os dados retornados eram em formato JSON. O sistema operacional Android conseguiu se comunicar com o servidor *Web* sem nenhuma dificuldade onde este realiza os cálculos dos preços finais das corridas. (VETTORAZZI; SCHUTZ, 2013).

3.5 Traveller

Com o crescente aumento do desempenho dos dispositivos móveis, é necessário criar novas tecnologias e softwares para aproveitar seus recursos. O serviço de GPS ajuda os usuários que se encontram em locais desconhecidos, fornecendo a eles informações úteis como clima e estabelecimentos comerciais próximos a sua área. (MELCHIOR, 2011).

A figura 3.5 mostra duas telas do aplicativo sendo, a tela inicial a que contém um botão para buscar as coordenadas geográficas através do GPS e a tela do mapa, a qual contém as opções de previsão do tempo e estabelecimentos comerciais de determinada região.



Figura 3.5: Tela Inicial do Traveller - Fonte Melchior (2011)

Os objetivos deste trabalho constituem na criação de um aplicativo chamado Traveller que será baseado em localização via GPS provendo auxílio aos seus usuários com o fornecimento de informações sobre o clima, posicionamento geográfico, cálculo de rotas e estabelecimentos nas áreas urbanas. As ferramentas utilizadas foram o sistema operacional Android, a linguagem de programação Java, o IDE Eclipse, o banco de dados SQLite e o kit de ferramentas de desenvolvimento em Android *Android Development Tools* (ADT). Os testes do aplicativo foram feitos em um celular Samsung i5700 Spica Galaxy rodando Android que se mostrou eficiente no quesito localização. A biblioteca do Google Maps mostrou-se eficaz nos métodos de geolocalização e clima. A classe *GeoCoder* mostrou algumas falhas no pacote *android.location* e a geolocalização reversa mostrou-se instável. (MELCHIOR, 2011).

3.6 Considerações Finais

Analizando os trabalhos feitos por outros autores nota-se que a automatização de processos é necessária quando se espera um padrão otimizado. As buscas via geolocalização auxiliam os usuários dos softwares quanto aos endereços e distâncias de localidades de seu interesse dentro do contexto das aplicações, assim como no TaxiDroid este projeto visa otimizar as requisições realizadas pelos clientes amenizando a dependência por telefones fixos. Os projetos analisados inspiraram também a exibição de apenas os mototaxistas disponíveis para solicitações. Este projeto torna-se útil aos pilotos e aos clientes que querem realizar corridas pois cria mais uma forma de comunicação entre eles.

Metodologia e Desenvolvimento

4.1 Considerações Iniciais

Para a construção de um *software* espera-se a existência de um roteiro, um conjunto de passos a se seguir para que o objeto de trabalho venha a alcançar a qualidade desejada. Todas as funcionalidades devem aproximar-se ao máximo das necessidades encontradas dentro do meio de atuação ao qual os usuários se encontram.

A metodologia utilizada para a criação deste *software* é baseada no arcabouço¹ genérico de processos em que, basicamente as atividades são divididas em cinco etapas principais: comunicação, planejamento, modelagem, construção e implantação.

4.2 Comunicação

Uma reunião com dois representantes da categoria dos mototaxistas e dois potenciais passageiros apontou as dificuldades apresentadas em seus cotidianos como por exemplo, a ineficácia de uma linha de telefone fixo nos pontos de mototáxi e a falta de aplicativos para dispositivos móveis que operem na região de Formiga - MG que ajudem a localizar pilotos.

Descrição dos participantes da reunião: o primeiro representante trabalha há 4 anos no ramo de mototáxi e está atualmente ativo na profissão, o segundo representante trabalhou por 3 anos no setor e não exerce a profissão atualmente. Os dois passageiros já fizeram uso do serviço de mototáxi, o primeiro cliente usa com frequência e o segundo usa apenas esporadicamente.

4.2.1 Descrição do Problema

Os mototaxistas nem sempre trabalham apenas com corridas ou viagens, muitos se dispõem a fazer outras prestações como serviços de banco ou entregas de produtos para lojas que não

¹O conjunto do que é necessário para uma construção.

possuem *motoboy*, eles não têm um único meio de divulgação para todos estes tipos de serviços. Alguma observação como formas de pagamentos e horários de trabalho geralmente ficam fixos na impressão dos cartões o que não apresenta agilidade de mudanças.



Figura 4.1: Forma de pagamento aceita - Fonte Silva (2015)

A figura 4.1 e a figura 4.3 exemplificam os problemas de apresentação dos mototaxistas descritos anteriormente, os tipos de serviços e formas de pagamento estão fixos e não possibilitam alterações instantâneas.

Um problema apontado pelos representantes foi a dificuldade no gerenciamento de atendimentos que têm horários próximos visto que eles precisam atender a várias ligações sequenciais, isso demanda um tempo pois alguns clientes demoram a confirmar os dados da corrida.

Não existe nenhum processo automatizado para acesso aos históricos das corridas efetuadas por período, de acordo com os dados obtidos no levantamento, o piloto normalmente não sabe sobre as corridas efetuadas durante o decorrer de uma semana.

Apesar dos cartões obterem os horários de trabalho do piloto muitos clientes ligam fora dele requerendo algum serviço. Geralmente, após cadastrarem o número do mototaxista no celular, as pessoas se desfazem ou perdem os cartões que contém esses tipos de dados.

Segundo os passageiros, há dificuldades de encontrar mototaxistas disponíveis sem a ajuda de alguém que já possua um número ou quando se está distante dos pontos. Outro empecilho que se encontra na parte do solicitante é a necessidade de efetuar a ligação. Mesmo com vários sistemas de conversa virtual, o mototaxista é encontrado principalmente via número do ponto ou por ligações para o seu celular.

Os pontos de mototáxi de Formiga - MG geralmente têm uma ou duas linhas de telefone fixo que são divididas entre os pilotos, eles formam uma fila que vai girando na medida que os

chamados são recebidos e o cliente não sabe quem irá atendê-lo no momento da solicitação.

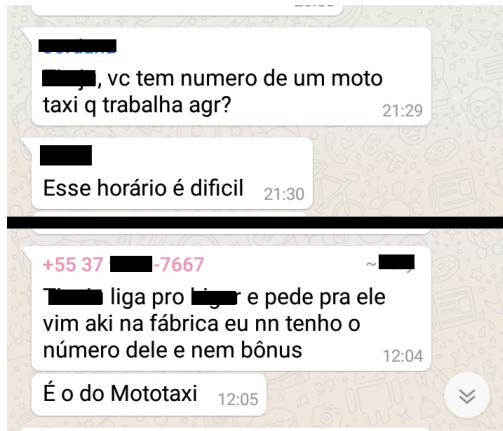


Figura 4.2: Necessidade de Encontrar um Mototaxista

A figura 4.2 exemplifica a dificuldade de encontrar um mototaxista disponível no determinado momento e também o problema da comunicação com o mototaxista visto que o cliente já possui o acesso à internet mas pode não ter créditos para ligar via operadora.

Um questionamento que surge momento que se tem acesso aos contatos dos pilotos é a escolha do mais qualificado para o seu tipo de necessidade. O cliente normalmente tem essa dificuldade quando não possui nenhuma indicação ou embasamento de corridas anteriores de cada mototaxista.

A figura 4.3 mostra os cartões de alguns mototaxistas da cidade de Formiga - MG. Estes cartões normalmente não têm capacidade de facilitar a escolha de um mototaxista por parte do solicitante pois não apresentam nenhuma indicação de outros usuários que já usaram seus serviços.

Quando o cliente se encontra em algum local que não tem costume de visitar, naturalmente tem alguma dificuldade de se localizar quando se trata do endereço. Algum ponto de referência pode te ajudar a descrever sua localização atual para que o mototaxista possa buscá-lo, porém há algum constrangimento nesse momento.



Figura 4.3: Cartões de alguns mototaxistas de Formiga - MG

4.3 Planejamento

Alguns requisitos foram apontados para que haja uma boa usabilidade no *software* como, alerta de solicitações de corrida em primeiro plano, possibilidade de criar uma fila com atendimentos simultâneos e controle das corridas efetuadas durante um certo período. Na visão do passageiro, o sistema deve apresentar uma listagem com alguns tipos de organização como, por pontuação, por tempo de resposta e por nome dando-lhe a opção de escolher com qual piloto ele quer efetuar a corrida.

Os processos que foram executados ao longo do desenvolvimento levaram em consideração o tempo disponível, e a complexidade do *software*. Os recursos utilizados compuseram uma série de *softwares* como a IDE Net Beans e a IDE Intel XDK, banco de dados MySQL, sistema operacional Windows, Android para testar a aplicação, linguagens HTML, Java, Css e Javascript, frameworks AngularJS, jQuery e Ionic. Os riscos foram mínimos visto que não havia uma equipe a ser coordenada e o escopo da aplicação era bem definido.

4.4 Modelagem

O Diagrama de Caso de Uso mostrado na figura 4.4 apresenta as funcionalidades possíveis para cada um dos três atores: Administrador, Mototaxista e Passageiro. O ator Administrador é o responsável por cadastrar os mototaxistas que serão apresentados no uso do aplicativo bem como suas identificações, foto, e tipos de serviços prestados. O ator Mototaxista é capaz de alterar sua própria disponibilidade alternando entre disponível ou indisponível, é capaz também de consultar as corridas que ele efetuou em um determinado período que contém data inicial e data final. Neste relatório ele não é capaz de visualizar as avaliações dos clientes. Ele pode interagir com os chamados realizados para ele escolhendo entre aceitar ou negar uma solicitação de corrida. O Mototaxista poderá ainda finalizar uma corrida que já tenha sido concluída permitindo ao Passageiro avaliá-la.

O ator Passageiro, assim como o Mototaxista, é capaz de consultar em um determinado período com data inicial e data final as corridas que ele solicitou mostrando as que foram aceitas e as que foram negadas pelos mototaxistas, ele será capaz também de avaliar as corridas que foram aceitas dando-lhe a opção de visualizar as pontuações no relatório. No momento em que o Passageiro quer chamar uma corrida, ele tem liberdade de escolher entre os mototaxistas disponíveis no momento da solicitação, em caso de estar em algum local desconhecido ou que não lembre do endereço, ele pode buscar seu ponto atual via Geolocalização facilitando a confirmação do logradouro, pode também estabelecer uma pontuação para o mototaxista referente a corrida que foi finalizada por parte do piloto.

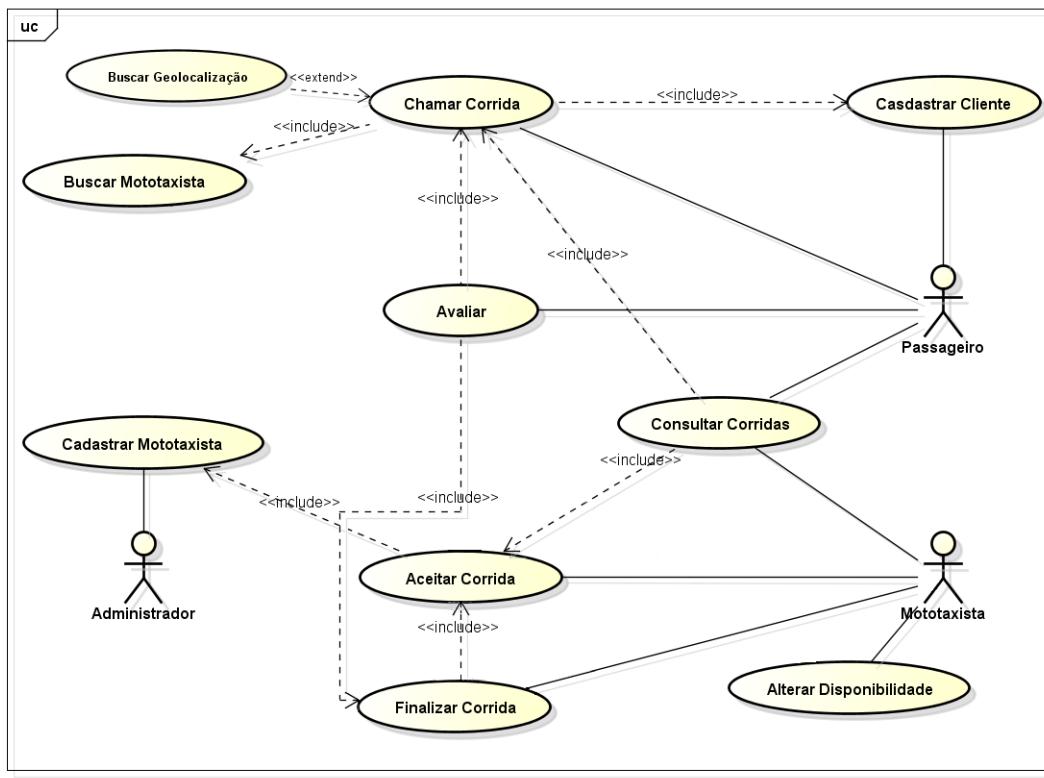


Figura 4.4: Diagrama de Caso de Uso

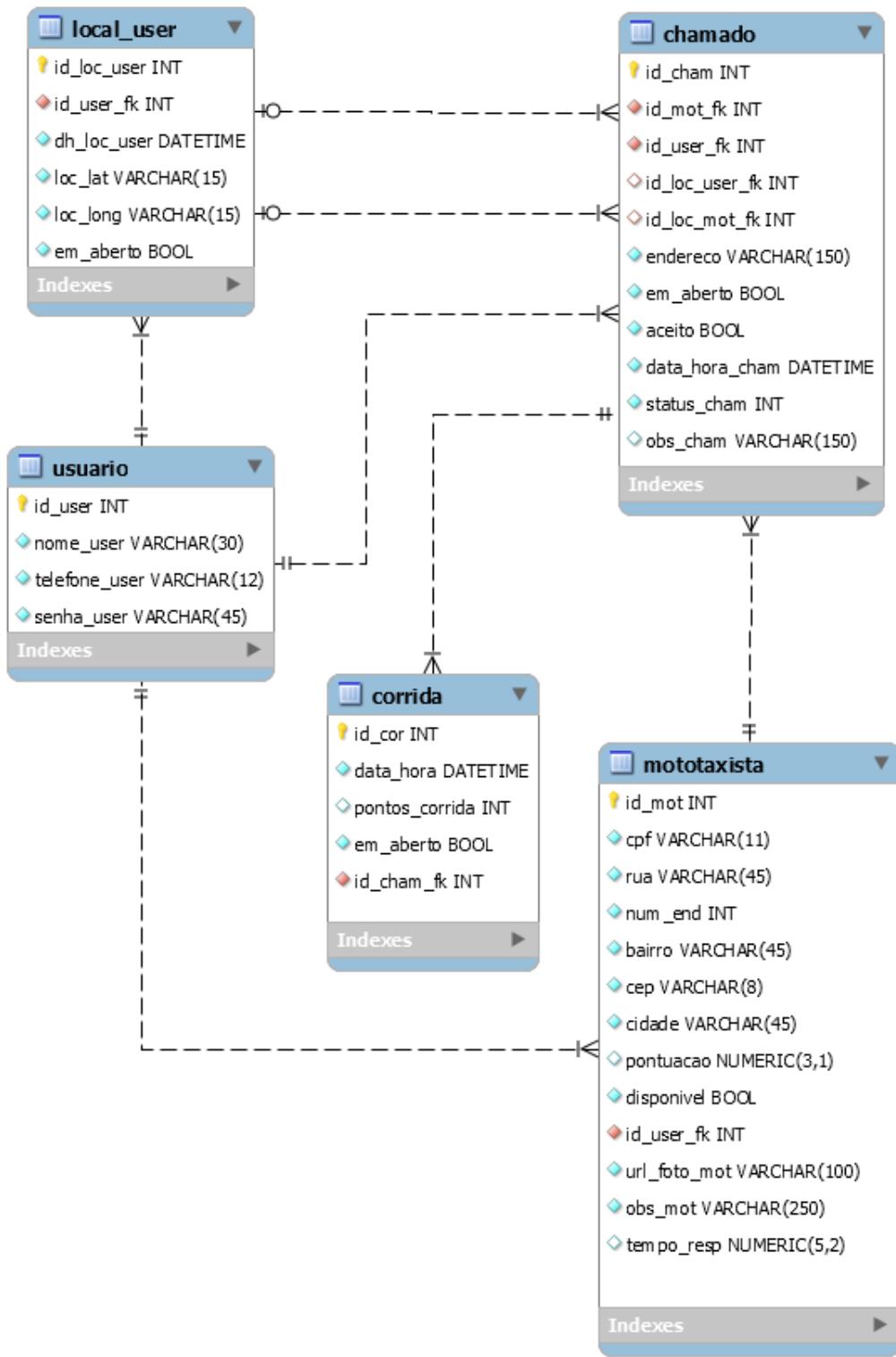


Figura 4.5: Diagrama Entidade Relacionamento

Na figura 4.5 é exibido o Diagrama Entidade Relacionamento (DER) que apresenta como foi montada a estrutura do banco de dados que serve como fonte de dados para o servidor RESTful que faz toda a troca de informações entre o passageiro e o mototaxista.

O DER apresenta cinco tabelas que são gerenciadas pelo banco de dados MySQL, a tabela *usuario* contém o identificador sequencial, o nome que ele quer que apareça nas trocas de solicitações, o número do celular que é tratado como chave única não permitindo a inserção do

mesmo com redundância e que também serve como *login* de acesso no aplicativo e a senha de acesso.

A tabela *mototaxista* apresenta uma estruturação mais elaborada de atributos visto que ele precisa ser melhor gerenciado pelo administrador do sistema. Nela é possível observar o identificador sequencial, o cpf, os dados de endereço, a pontuação que armazena a média total das avaliações feitas pelos seus cliente ao término das corridas, o atributo booleano que permite a visualização do mesmo como disponível para receber solicitações, a chave estrangeira para a tabela de *usuario* que armazena seu nome e dados de login, o atributo *url_foto_mot* que guarda a foto do mototaxista que será apresentada aos clientes na listagem dos pilotos e no histórico de corridas, o campo de observação guarda informações como tipos de serviço prestados e opções de pagamento. O atributo *tempo_resp* guarda a média do tempo que o mototaxista demora para responder os chamados de corridas. Esta média permite a ele alcançar uma melhor localização na listagem dos disponíveis quando ela está ordenada por tempo de resposta.

A tabela *local_user* é usada para salvar a geolocalização atual do cliente no momento da solicitação de corrida e também conta com o identificador primário sequencial, a chave estrangeira para identificação do usuário, data e hora corrente no momento da localização, os atributos que contém as coordenadas geográficas de latitude e longitude e uma variável booleana que indica se a localização já foi usada ou não.

A tabela de *chamado* é utilizada no momento em que o cliente realiza a solicitação de corrida para algum mototaxista. Ela contém o identificador primário sequencial, as chaves estrangeiras que identificam quem é o cliente que está efetuando o chamado e qual o mototaxista que ele está solicitando. Nesta tabela existem os identificadores estrangeiros de localização do cliente e do mototaxista que não são obrigatórios visto que nem sempre o GPS dos dois estarão ativados. O atributo de endereço guarda a localização atual do cliente no momento do chamado sendo que este pode ser preenchido manualmente sem o auxílio do GPS. O campo *em_aberto* indica se o chamado está pendente ou finalizado, o atributo denominado como *aceito* indica se o mototaxista aceitou ou recusou o chamado, a tabela disponibiliza também o campo de data e hora da solicitação.

O atributo *status_cham* é alterado em vários momentos durante o processo de solicitação de uma corrida. Quando o cliente inicia o chamado ele tem o valor *1*, quando o chamado chega para o mototaxista e aciona a notificação no aparelho muda o valor para *2*, no momento em que o mototaxista responde o chamado aceitando ou recusando o valor deste atributo muda para *3*, a resposta do chamado retorna para o cliente, então o valor passa a ser *4* e por fim, quando o cliente confirma a visualização da resposta o atributo recebe o valor *5* que indica que o chamado já passou por todos os processos possíveis dentro do escopo da aplicação. A tabela *chamado* conta com um atributo de observação que pode ser preenchido pelo mototaxista como justificativa em caso de recusa de um chamado, em caso de não preenchimento desta observação o cliente recebe uma mensagem padrão do sistema.

A tabela *corrida* é preenchida no momento em que o mototaxista aceita o chamado. Ela contém o identificado primário sequencial, a data e hora que a corrida foi aceita gerando o

tempo médio de resposta do mototaxista, os pontos que o cliente atribuiu ao atendimento do piloto, o booleano que indica se a corrida está aberta ou finalizada e o identificador estrangeiro para a tabela de chamado o qual provê informações detalhadas da solicitação como um todo.

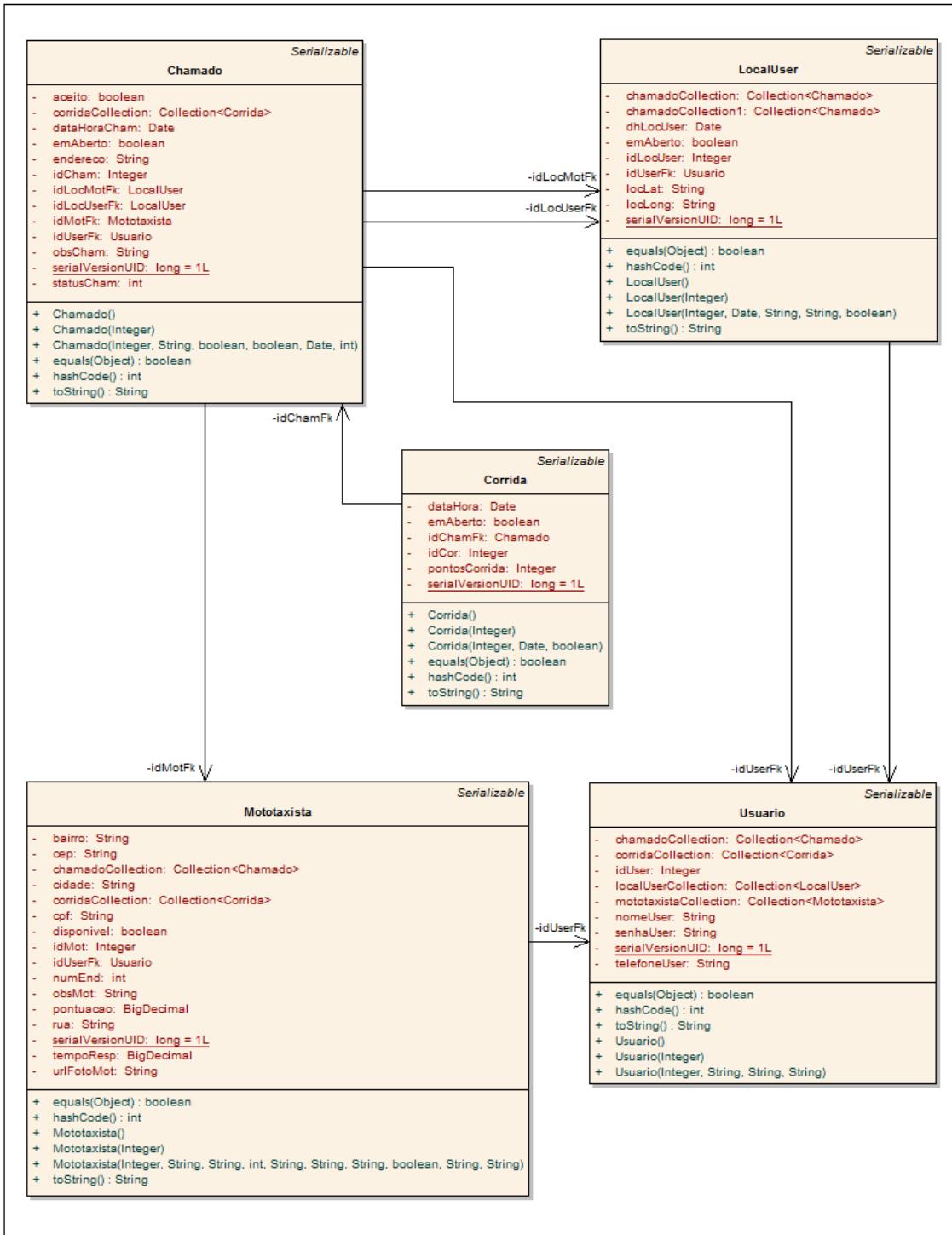


Figura 4.6: Diagrama Classe dos Modelos de Entidades

A figura 4.6 apresenta o digrama de classe dos modelos de entidades que são mapeados de acordo com as tabelas do banco de dados e são usados dentro da aplicação RESTful do servidor em conjunto com as classes de serviços.

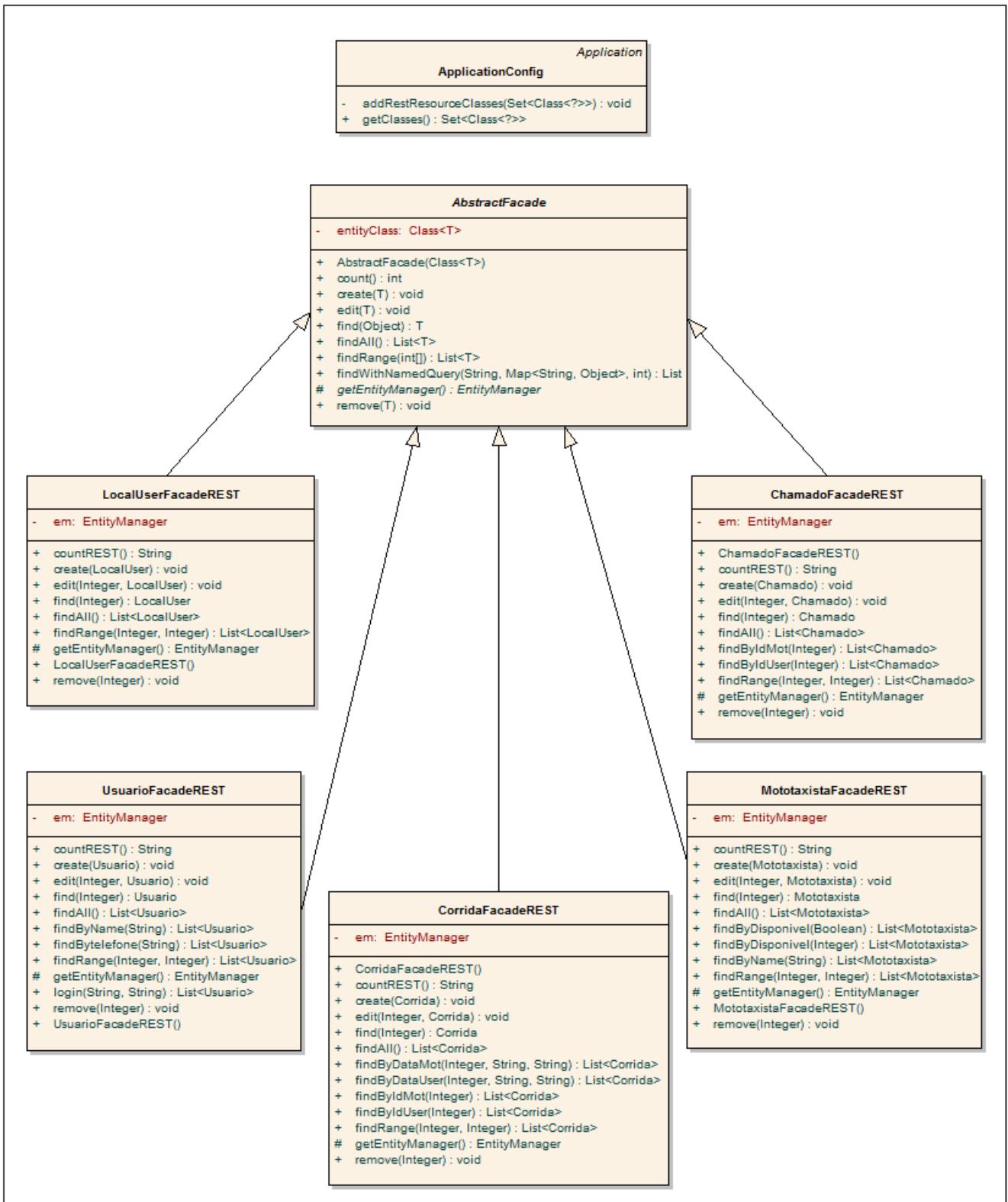


Figura 4.7: Diagrama Classe dos Serviços disponíveis

A figura 4.7 exibe o diagrama das classes localizadas no pacote de serviços. Estas são as responsáveis por usar os métodos HTTP do servidor de aplicações e prover a transição de informações entre o aplicativo móvel e o banco de dados do MySQL. Todos os métodos apresentados nestas classes são acompanhados de alguma notação padrão do RESTful: GET,

POST, PUT ou DELETE. Por padrão estas classes contém os métodos para realização da persistência de informações no banco de dados sendo eles o *create(Objeto)*, *edit(Integer, Objeto)*, *find(Integer)*, *findAll()*, *remove(Integer)*, *countREST()*, *findRange(Integer, Integer)* que servem para criar, editar, buscar, buscar todos, remover, contagem total e buscar por intervalo de código, respectivamente, estes métodos são os que receber e retornam informações via HTTP.

As classes de serviços *UsuarioFacadeREST*, *MototaxistaFacadeREST*, *LocalUserFacadeREST*, *ChamadoFacadeREST* e *CorridaFacadeREST* estendem à classe *AbstractFacade* que contém os métodos padrões que usam entidades abstratas para persistir as informações dentro do banco. Na classe *AbstractFacade* existe o método *findWithNamedQuery(String, Map<String, Object>, int)* que usa o conceito de JPQL para realizar buscas de informações mais específicas da aplicação.

A classe *ApplicationConfig* é a responsável por mapear o caminho principal para acessar as classes de serviços e adicionar todas elas ao escopo da aplicação deixando-as disponíveis para fazer transações HTTP.

4.5 Construção

A construção da aplicação como um todo utilizou três *softwares* sendo eles o gerenciador de banco de dados Workbench e os IDE's NetBeans e Intel XDK.

4.5.1 Construção do Banco de Dados

A criação do banco de dados do MySQL deu-se através da geração automática do *schema* baseado no diagrama entidade relacionamento mostrado na figura 4.5. Encontrada nas opções do gerenciador de bancos Workbench, a opção *Forward Engineer* permite essa criação.

Como foi necessário calcular a pontuação média das avaliações feitas pelos clientes, foi criado um *Trigger* mostrado na figura 4.8 que lê todas as corridas que foram avaliadas, faz uma média simples e atualiza a pontuação atual do mototaxista.

```

1  DELIMITER $$

2

3  CREATE TRIGGER attPontos_mot AFTER UPDATE
4    ON corrida
5    FOR EACH ROW
6  BEGIN
7    SET @id_mot_temp := (SELECT id_mot
8      FROM mototaxista
9      INNER JOIN chamado ch ON ch.id_mot_fk = id_mot
10     INNER JOIN corrida c ON ch.id_cham = NEW.id_cham_fk WHERE c.id_cor = NEW.id_cor);

11   SET @soma_geral := 0;
12   SET @soma_geral := (SELECT AVG(pontos_corrida) FROM corrida c
13     INNER JOIN chamado ch ON ch.id_cham=c.id_cham_fk
14     INNER JOIN mototaxista m ON m.id_mot=ch.id_mot_fk WHERE id_mot=@id_mot_temp AND c.pontos_corrida > 0);
15

16   UPDATE mototaxista SET pontuacao = @soma_geral WHERE id_mot=@id_mot_temp;
17
18 END$$
19
20 DELIMITER ;
21

```

Figura 4.8: Trigger que atualiza a pontuação do Mototaxista

Para obter-se os resultados de tempo médio de resposta aos chamados foi criado outro

Trigger que lê a data e hora que o cliente efetuou o chamado e a data e hora que mototaxista respondeu. Foram usadas funções SQL para extrair a diferença de tempo em minutos como apresentado na figura 4.9, este tempo é atualizado no cadastro do mototaxista.

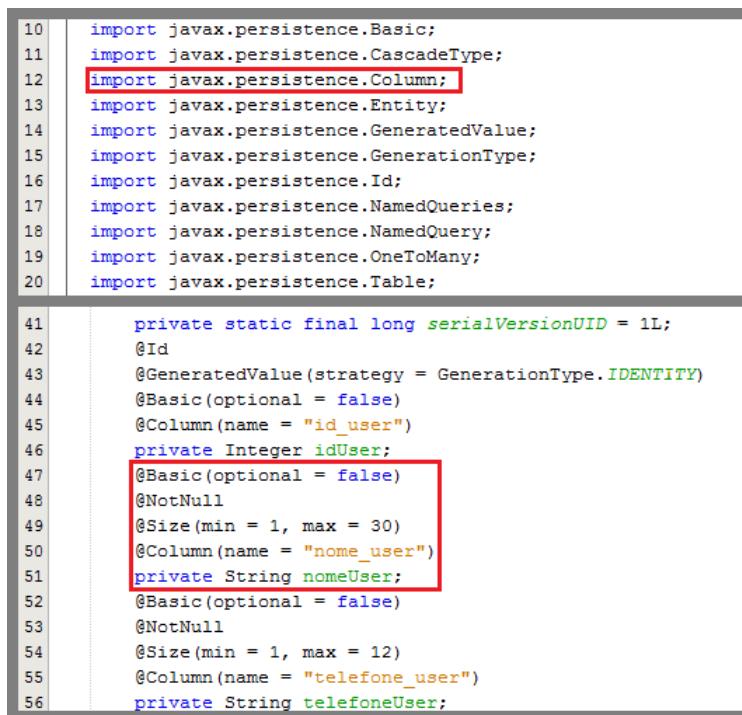
```
1  DELIMITER $$  
2  
3  CREATE TRIGGER att_tempo_resp_mot AFTER INSERT  
4  ON corrida  
5  FOR EACH ROW  
6  BEGIN  
7      SET @id_mot_temp := (SELECT id_mot  
8          FROM mototaxista  
9          INNER JOIN chamado ch ON ch.id_mot_fk = id_mot  
10         INNER JOIN corrida c ON ch.id_cham = NEW.id_cham_fk WHERE c.id_cor = NEW.id_cor);  
11  
12      SET @tempo_medio := 0;  
13      SET @tempo_medio :=  
14          (SELECT AVG(  
15              TIME_TO_SEC(  
16                  TIMEDIFF(data_hora, data_hora_cham)))  
17                  FROM corrida co  
18                  INNER JOIN chamado ch ON co.id_cham_fk = ch.id_cham  
19                  INNER JOIN mototaxista m ON m.id_mot = ch.id_mot_fk  
20                  WHERE id_mot=@id_mot_temp) / 60;  
21  
22      UPDATE mototaxista SET tempo_resp = @tempo_medio WHERE id_mot=@id_mot_temp;  
23  
24  END$$  
25  
26  DELIMITER ;
```

Figura 4.9: Trigger que atualiza o tempo de resposta do Mototaxista

4.5.2 Construção do Servidor RESTful

Para realizar a construção do servidor RESTful foi usado o IDE NetBeans. A geração das funcionalidades básicas dele foi dada de maneira automática através da opção *Novo Web Service RESTful a partir do banco de dados* que pode ser encontrada no menu *Novo, Outros, Web Services*.

O servidor utiliza as classes do pacote *javax.persistence* para fazer a persistência das informações dentro do banco de dados. Cada coluna da tabela é mapeada usando as bibliotecas referentes, como pode ser observado, a figura 4.10 exemplifica a associação de uma coluna da tabela à uma variável dentro da classe Java.



```
10 import javax.persistence.Basic;
11 import javax.persistence.CascadeType;
12 import javax.persistence.Column;
13 import javax.persistence.Entity;
14 import javax.persistence.GeneratedValue;
15 import javax.persistence.GenerationType;
16 import javax.persistence.Id;
17 import javax.persistence.NamedQueries;
18 import javax.persistence.NamedQuery;
19 import javax.persistence.OneToMany;
20 import javax.persistence.Table;
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41 private static final long serialVersionUID = 1L;
42 @Id
43 @GeneratedValue(strategy = GenerationType.IDENTITY)
44 @Basic(optional = false)
45 @Column(name = "id_user")
46 private Integer idUser;
47 @Basic(optional = false)
48 @NotNull
49 @Size(min = 1, max = 30)
50 @Column(name = "nome_user")
51 private String nomeUser;
52 @Basic(optional = false)
53 @NotNull
54 @Size(min = 1, max = 12)
55 @Column(name = "telefone_user")
56 private String telefoneUser;
```

Figura 4.10: Exemplo de associação da coluna com a variável java

Observa-se na figura 4.11 que a classe *ApplicationConfig* é a que indica qual o caminho principal para acessar todos os serviços disponíveis pelo servidor através do comando *@javax.ws.rs.ApplicationPath("rest")*, dentro dela também são adicionadas todas as classes que contém estes serviços deixando-os acessíveis para chamadas HTTP.

A classe *AbstractFacade* contém todos os métodos básicos do CRUD para busca e inserção de informações dentro do banco de dados, porém apenas estes métodos não foram suficientes para realizar uma recuperação otimizada. A figura 4.12 mostra como foi construído um método baseado em JPQL para buscas diversificadas dentro do banco.

O método mostrado na figura 4.12 precisa das JPQL's como um dos parâmetros para fazer as buscas. A figura 4.13 exemplifica as *queries* da classe Corrida que foram criadas para buscar informações de acordo com as regras de negócio do aplicativo móvel, como exemplo há a *query Corrida.findByDataMot* que busca todas as corridas efetuadas por um piloto em um determinado período.

```

1 package br.com.servidor.servicos;
2
3 import java.util.Set;
4 import javax.ws.rs.core.Application;
5
6 @javax.ws.rs.ApplicationPath("rest")
7 public class ApplicationConfig extends Application {
8
9     @Override
10    public Set<Class<?>> getClasses() {
11        Set<Class<?>> resources = new java.util.HashSet<>();
12        addRestResourceClasses(resources);
13        return resources;
14    }
15
16    private void addRestResourceClasses(Set<Class<?>> resources) {
17        resources.add(br.com.servidor.servicos.ChamadoFacadeREST.class);
18        resources.add(br.com.servidor.servicos.CorrvidaFacadeREST.class);
19        resources.add(br.com.servidor.servicos.LocalUserFacadeREST.class);
20        resources.add(br.com.servidor.servicos.MototaxistaFacadeREST.class);
21        resources.add(br.com.servidor.servicos.UsuarioFacadeREST.class);
22    }
23}

```

Figura 4.11: Classe ApplicationConfig

```

68 public List findWithNamedQuery(String namedQueryName,
69         Map<String, Object> parameters, int resultLimit) {
70
71     Set<Map.Entry<String, Object>> rawParameters = parameters.entrySet();
72     Query query = getEntityManager().createNamedQuery(namedQueryName);
73     if (resultLimit > 0) {
74         query.setMaxResults(resultLimit);
75     }
76     for (Map.Entry<String, Object> entry : rawParameters) {
77         query.setParameter(entry.getKey(), entry.getValue());
78     }
79     return query.getResultList();
80 }

```

Figura 4.12: Busca baseada em JPQL, método contido em AbstractFacade

```

@NamedQueries({
    //////
    @NamedQuery(name = "Corrida.findByDataUser", query =
    "SELECT c FROM Corrida c WHERE c.idChamFk.idUserFk.idUser = :idUser AND c.dataHora BETWEEN :dataInic AND :dataFim"),
    //////
    @NamedQuery(name = "Corrida.findByDataMot", query =
    "SELECT c FROM Corrida c WHERE c.idChamFk.idMotFk.idMot = :idMot AND c.dataHora BETWEEN :dataInic AND :dataFim"),
    //////
    @NamedQuery(name = "Corrida.findByPontosCorrida", query =
    "SELECT c FROM Corrida c WHERE c.pontosCorrida = :pontosCorrida"),
    //////
    @NamedQuery(name = "Corrida.findByEmAberto", query =
    "SELECT c FROM Corrida c WHERE c.emAberto = :emAberto"),
    //////
    @NamedQuery(name = "Corrida.findByIdUser", query =
    "SELECT c FROM Corrida c WHERE c.idChamFk.idUserFk.idUser = :idUser AND c.idChamFk.aceito = TRUE AND c.emAberto =
    FALSE AND c.pontosCorrida = 0"),
    //////
    @NamedQuery(name = "Corrida.findByIdMot", query =
    "SELECT c FROM Corrida c WHERE c.idChamFk.idMotFk.idMot = :idMot AND c.emAberto = TRUE"))
}

```

Figura 4.13: Querys JPQL da classe Corrida

Após a criação da JPQL mostrada na figura 4.13 foi necessário que houvesse um método que aceitasse as buscas HTTP para trabalhar em conjunto com ela. Na figura 4.14 é exibido este método, ele recebe os parâmetros da busca por meio do link da requisição GET, seta eles no *HashMap* *param* e depois chama o método existente na classe *super* mostrado na figura 4.12.

```
136     @GET  
137     @Path("idMot/{idMot}/data/{dataInic}+{dataFim}")  
138     @Produces(MediaType.APPLICATION_JSON+";charset=utf-8")  
139     public List<Corrida> findByDataMot(@PathParam("idMot") Integer idMot,  
140                                         @PathParam("dataInic") String dataInic, @PathParam("dataFim") String dataFim) {  
141  
142         Date dataI = null;  
143         Date dataF = null;  
144         SimpleDateFormat sdf= new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");  
145         try {  
146             dataI = sdf.parse(dataInic + " 00:00:00");  
147             dataF = sdf.parse(dataFim + " 23:59:59");  
148         } catch (ParseException ex) {  
149             Logger.getLogger(CorridaFacadeREST.class.getName()).log(Level.SEVERE, null, ex);  
150         }  
151         Map<String, Object> param = new HashMap<>();  
152         param.put("idMot", idMot);  
153         param.put("dataInic", dataI);  
154         param.put("dataFim", dataF);  
155         return super.findWithNamedQuery("Corrida.findByDataMot", param, 0);  
156     }
```

Figura 4.14: Método GET para buscar corridas por período

4.5.3 Construção do Aplicativo Móvel

Para a construção da aplicação móvel foi usada a IDE Intel XDK que é capaz de construir aplicações híbridas, este aplicativo tem um foco mais voltado para a plataforma Android. Todas as telas foram criadas usando as tecnologias HTML, CSS e Ionic, para as funcionalidades delas foram usados AngularJS e JavaScript e jQuery.

O AngularJS gerencia a transição de informações entre as telas e o *controller* através do *\$scope* que é injetado na criação do módulo da aplicação *angular.module("myApp").controller(...)*, neste controle há injeção de todas as dependências usadas para gerar notificações, mensagens, criar filtros de dados entre outros. O *\$scope* é responsável por guardar funções e variáveis e é atualizado pelo *ngBind* sempre que há algum evento na tela ou no *controller*.

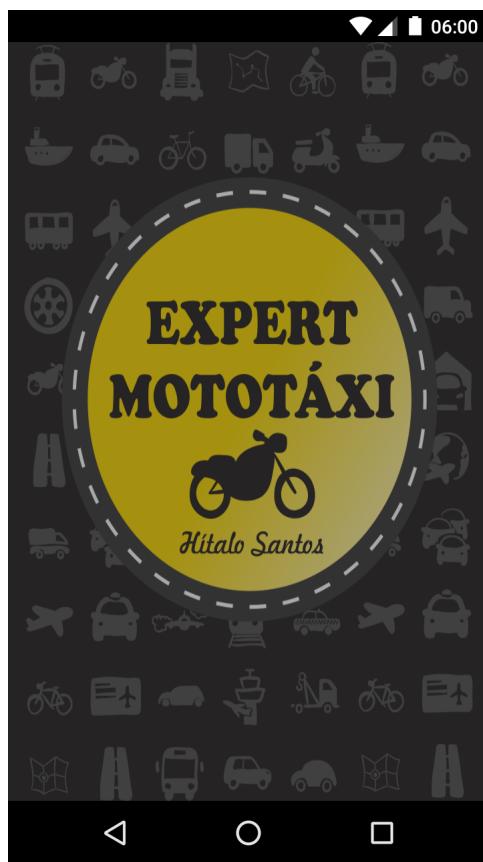


Figura 4.15: Tela de Splash com nome do aplicativo

A figura 4.15 mostra a tela de *Splash* da aplicação que é exibida na inicialização dando ênfase ao nome do *software*. Esta tela pode ser configurada na aba *PROJECTS* do IDE Intel XDK e tem disponibilidade de ajuste para várias resoluções de tela.

A figura 4.16 exibe a tela de login do aplicativo e usa o comando *ui-mask=""(99) 9 9999-9999"* para criar a máscara de entrada do número do celular, esta tela tem o botão que dá acesso a tela de criação de novo usuário. Todas as requisições feitas ao servidor que apresentarem alguma falha foram tratadas para serem exibidas ao usuário.

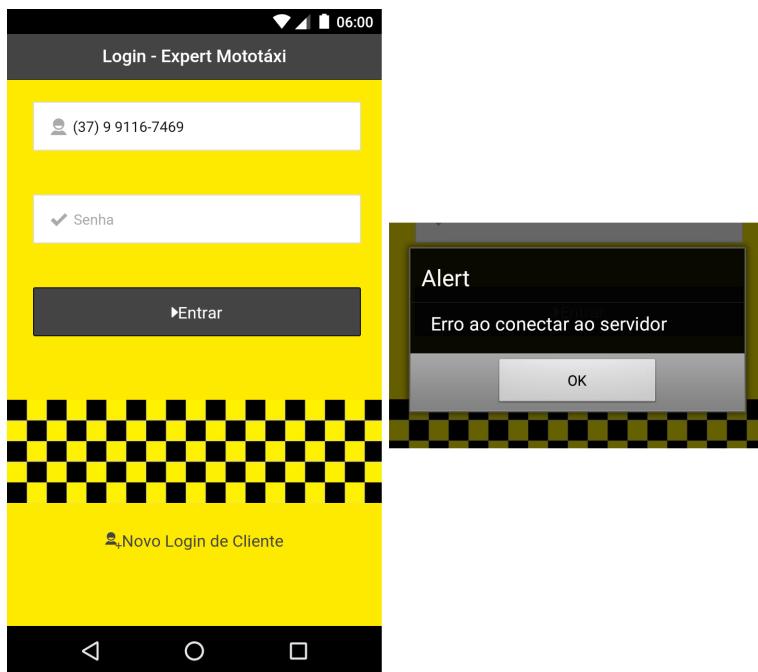


Figura 4.16: Tela de Login

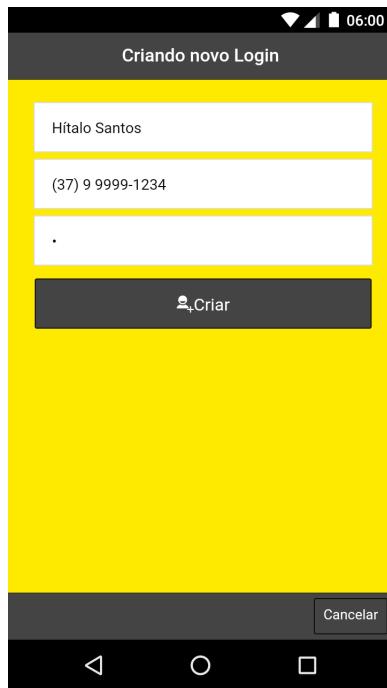


Figura 4.17: Tela Criando Novo Login

A figura 4.17 mostra tela para criação de um novo login de usuário que será cadastrado automaticamente como passageiro. Os usuários que quiserem se tornar mototaxistas devem acionar o administrador da aplicação visto que eles têm um gerenciamento de informações mais elaborado. Esta tela de criação de novo usuário segue um padrão de gerenciamento de objetos que é usado em praticamente toda a aplicação. A figura 4.18 mostra uma versão reduzida do código HTML da tela *Criando Novo Login*, nele é possível visualizar a diretiva do AngularJS chamada *ng-model* que neste caso, cria um objeto chamado *newUser* que contém os atributos

de nome, telefone e senha. Estes atributos são associados aos respectivos campos da *view*, quando uma informação é inserida na tela já fica setada dentro deste objeto.

Enquanto todos os campos não são preenchidos corretamente, a diretiva *ng-disabled= "newClienteForm.\$invalid"* desabilita o botão que faz a confirmação dos dados do formulário. Ao chamar a diretiva *ng-click="criarNovoUser(newUser)"* o AngularJS chama esta função que está previamente cadastrada no *\$scope* transportando as informações do formulário dentro do objeto *newUser*.

```
1 <div class="upage hidden cor_pg" id="pg_novo_login">
2   <div class="upage-content ac0 content-area vertical-col left"
3     id="page_18_67">
4     <form name="newClienteForm">
5       <input type="text" placeholder="Nome"
6         ng-model="newUser.nomeUser" ng-required="true">
7       <input type="tel" placeholder="Número do Celular"
8         ng-model="newUser.telefoneUser" ui-mask="(99) 9 9999-9999"
9           ui-mask-placeholder="" ui-mask-placeholder-char="_"
10          ng-required="true" ng-minlength="11">
11        <input type="password" placeholder="Senha"
12          ng-model="newUser.senhaUser" ng-required="true">
13      </form>
14      <button class="button widget uib_w_76 d-margins ion button-dark
15        ion-android-add-contact" data-uib="ionic/button" data-ver="0"
16        ng-disabled="newClienteForm.$invalid"
17        ng-click="criarNovoUser(newUser)">Criar</button>
18    </div>
19 </div>
```

Figura 4.18: HTML da tela Criando Novo Login

Telas do Cliente

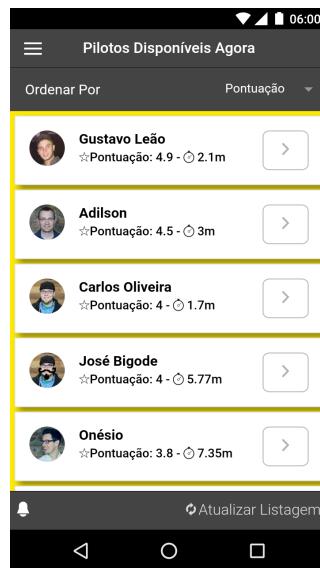


Figura 4.19: Tela Inicial do Cliente

A tela inicial do cliente mostrada na figura 4.19 é responsável por exibir todos os mototaxis disponíves no momento, inicialmente os pilotos são organizados por pontuação mas podem ser reclassificados por tempo de resposta ou por nome.



Figura 4.20: Tela de Detalhes do Mototaxista

Ao selecionar o botão existente em cada item da listagem dos pilotos, o cliente é direcionado para a tela que exibe os detalhes do mototaxista como mostrado na figura 4.20. Esta tela permite visualizar as observações referente ao seus tipos de serviços, tipos de pagamento, horários entre outros. É nessa tela também que o passageiro tem a opção de efetuar um chamado podendo informar o endereço manualmente ou então fazendo uma busca de sua geolocalização através da opção *Buscar no Mapa*.

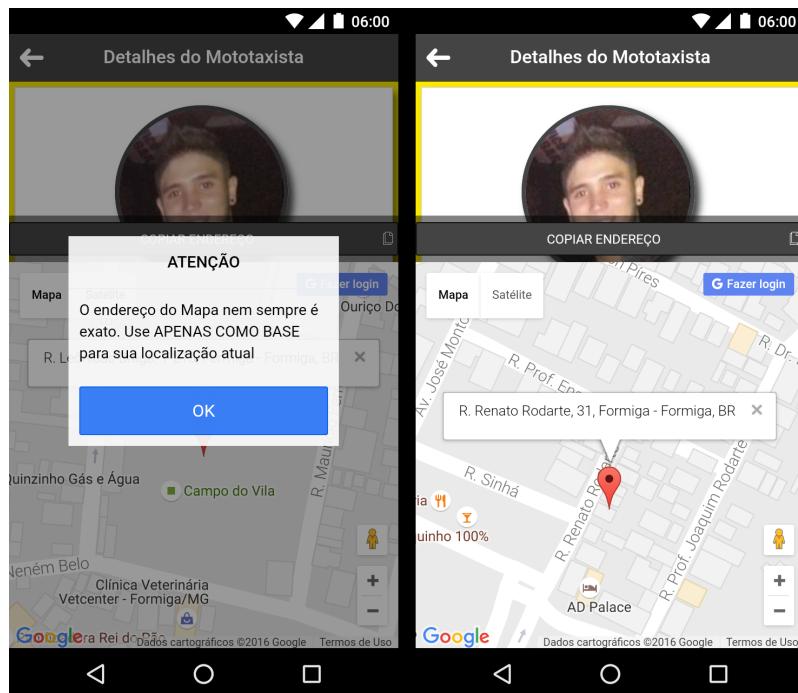


Figura 4.21: Buscando Geolocalização

A figura 4.21 apresenta a busca da geolocalização do cliente e informa que há margem de erro

no retorno do endereço. Para recuperar as coordenadas geográficas, foram usados os comandos do Cordova `latitude = position.coords.latitude` e `longitude = position.coords.longitude`, estes valores são passados para o objeto `latlong`, este objeto é usado na função `geocodeLatLng()` para fazer a recuperação do endereço textual de acordo com as coordenadas geográficas. O marcador que indica o ponto atual do cliente foi criado como `draggable: true`, quando verdadeiro este atributo permite que ele move o marcador dentro da tela do mapa aumentando a precisão de seu local atual.

Sempre que o marcador é movido o endereço atual é atualizado e setado no campo de endereço mostrado na figura 4.20. Mesmo buscando o endereço no mapa, o cliente ainda pode aditá-lo para acrescentar ou remover alguma informação antes de confirmar o chamado.



Figura 4.22: Confirmação do Chamado

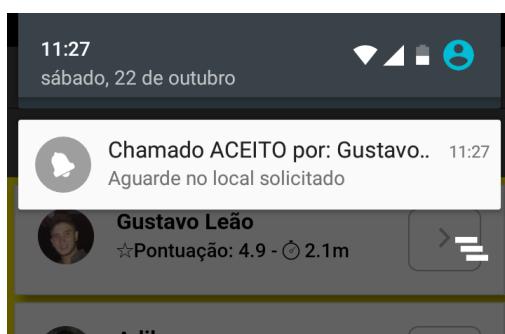


Figura 4.23: Notificação *push* da resposta

Ao confirmar os dados de endereço, o cliente é redirecionado para a tela inicial e recebe uma mensagem para aguardar no local solicitado como mostrado na figura 4.22, neste momento,

a função `$scope.verificaDadosCli = $interval(function() {...}, 3000);` é acionada e fica aguardando a resposta do mototaxista, esta função faz requisições GET para o servidor e retorna as respostas de chamados e corridas a serem avaliadas. Sempre que o piloto responde, o cliente é notificado através função `$cordovaLocalNotification.schedule({...})` que é disponibilizada pela dependência `$cordovaLocalNotification`, esta função exibe uma notificação *push* do Android como mostrado na figura 4.23.

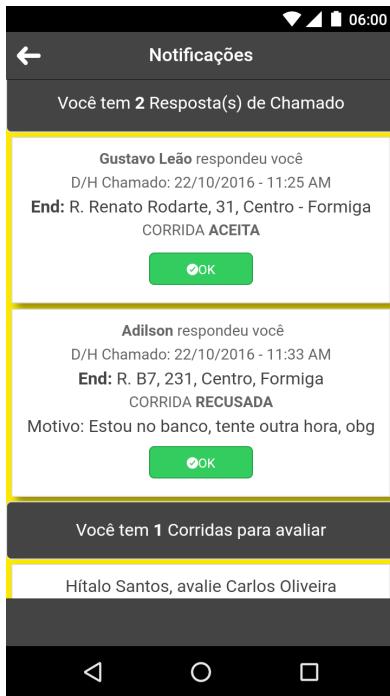


Figura 4.24: Tela de Notificações do Cliente

Além da notificação *push*, o cliente tem a tela *Notificações* para visualizar se as respostas dos mototaxistas foram aceitas ou negadas como mostrado na figura 4.24, nesta mesma tela aparecem também as corridas que foram finalizadas mas ainda não foram avaliadas.

Para avaliar uma corrida é necessário que ela tenha sido finalizada pelo mototaxista. O cliente deve clicar no item da listagem que contém os dados do serviço prestado, ao fazer isso o sistema chama a função `$scope.avaliar = function(corrida) {...}` que aciona o `popup $ionicPopup.show({...})`, este *popup* é exibido na figura 4.25, ele permite que o cliente escolha uma pontuação entre um e cinco.

A tela de histórico do cliente pode ser acessada através do menu lateral da aplicação, para efetuar uma busca é necessário informar a data inicial e data final. Como mostrado na figura 4.26 o histórico exibe os dados do mototaxista, o endereço, a data e hora que o piloto respondeu. Caso a corrida tenha sido aceita, os pontos da avaliação são exibidos, se o chamado foi recusado, no lugar da pontuação aparece um texto indicando que foi negado e o motivo.



Figura 4.25: *Popup* de avaliação de corrida

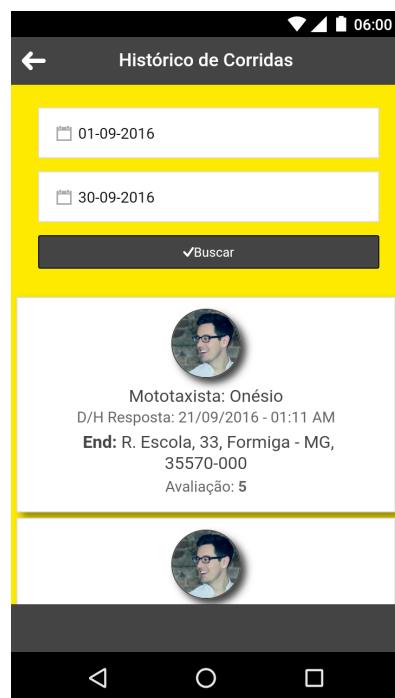


Figura 4.26: Histórico de Corridas do Cliente

Telas do Mototaxista

As telas do mototaxista são encontradas na mesma aplicação do cliente, porém quando o login é efetuado ele é redirecionado e visualiza apenas as telas de sua alcada. O menu lateral do mototaxista tem as informações de sua pontuação, nome, tempo de resposta e as observações que são visualizadas pelos cliente, tem também opção de alterar sua disponibilidade deixando-o visível ou não para os clientes solicitarem corridas como mostrado na figura 4.27.



Figura 4.27: Menu Lateral do Mototaxista



Figura 4.28: Tela inicial do Mototaxista

Como mostrado na figura 4.28 a tela inicial do mototaxista é denominada *Gerenciador de Corridas* e exibe todas as solicitações feitas pelos clientes. Ela é capaz de exibir múltiplas solicitações e permite que mototaxista aceite ou não um chamado. Caso ele queira recusar um chamado de corrida, o aplicativo lhe disponibiliza um campo facultativo para digitar alguma justificativa que será visualizada pelo cliente, quando ele aceita um chamado ele vai para a fila de corridas em aberto que ficam organizadas por data e hora de confirmação. Quando o mototaxista finaliza uma corrida ele tem a opção concluir-la permitindo ao cliente avaliá-la.

Quando a aplicação é iniciada a função `cordova.plugins.backgroundMode.enable()` é chamada para ativar o módulo de *background*, que fica verificando se a aplicação está em segundo plano ou não, quando ela é colocada em segundo plano a função `cordova.plugins.backgroundMode.onactivate=function(){...}` aciona a verificação de dados pendentes para o mototaxista, esta verificação é descrita na função `$scope.verificaDadosMot()` e fica esperando solicitações dos clientes mesmo com a aplicação em segundo plano. Quando um cliente realiza um chamado ao mototaxista ele é informado também pela notificação *push* do comando `$cordovaLocalNotification.schedule({...})` como pode ser observado na figura 4.29.

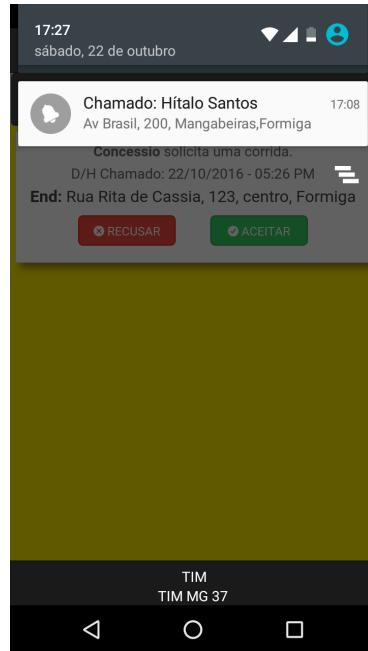


Figura 4.29: Notificação *push* de Corrida



Figura 4.30: Tela de Histórico do Mototaxista

A figura 4.30 mostra a tela do histórico de corridas solicitadas ao mototaxista informando o nome do cliente, a data e hora da resposta, qual foi o endereço indicado pelo cliente e se a corrida foi aceita ou recusada por ele. É válido ressaltar que o mototaxista, em momento algum, tem acesso a pontuação que ele recebeu na corrida que foi efetivada

4.6 Implantação

A implantação foi feita em módulos testando cada nova parte do sistema e observando como essa funcionalidade interagia com o restante do *software* que já foi desenvolvido anteriormente. O Intel XDK reúne todos os códigos escritos em linguagem de aplicações híbridas e gera os instaladores nativos de acordo com cada sistema operacional, no caso deste aplicativo, foi gerado o *Android Package* (APK), esse arquivo é o pacote de instalação de aplicações móveis do Android e permite a execução real do sistema em um dispositivo virtual ou físico. Ele foi instalado via *Universal Serial Bus* (USB) nos dispositivos disponibilizados para os testes.

Testes e Resultados

Os testes foram efetuados em quatro dispositivos físicos usando o sistema operacional Android:

- 1 Motorola Moto G 2^a Geração - Android 5.1
- 1 Motorola Moto G 3^a Geração - Android 6.0
- 2 Motorola Moto G 4^a Geração - Android 6.0.1

Os usuários acessavam o servidor que estava disponível *localhost* via rede *wireless*. Estes testes envolveram chamadas simultâneas a um dos usuários cadastrados como mototaxista verificando as notificações *push* e o gerenciador de corridas. As respostas foram intercaladas entre aceitas e recusadas e foram efetuadas com diferentes espaços de tempo. A busca do endereço via geolocalização foi testada em todos os aparelhos.

Ao término dos testes, os usuários preencheram o questionário no qual puderam classificar as funcionalidades do aplicativo como: desempenho, organização das informações, gerenciamento de corridas, sistema de pontuação dos mototaxistas entre outros. Este questionário se encontra no apêndice A.

Todas as chamadas enviadas ao mototaxista foram entregues corretamente mesmo quando houve múltiplas solicitações de corridas, as notificações *push* exemplificadas na figura 4.29 apresentaram um atraso de aproximadamente quatro segundos para aparecerem após a solicitação de chamado ser recebida pelo gerenciador de corridas do mototaxista.

Houve êxito nas notificações de resposta aos clientes e as notificações *push* apresentaram um atraso equivalente as dos mototaxista. A busca de informações dos históricos apresentou resposta instantânea, a renderização dos dados na tela não prejudicou o desempenho dos aparelhos, todas as telas mostraram uma responsividade eficaz e não ofuscaram a visualização dos dados em diferentes resoluções ou em modo paisagem. O aplicativo continua funcionando

quando é colocado em segundo plano e realiza todas as notificações solicitadas ao cliente ou mototaxista.

A pontuação do mototaxista foi calculada corretamente de acordo com a média das corridas que foram avaliadas, o tempo médio de resposta entre a data e hora da solicitação do cliente e a da resposta do piloto foi extraída corretamente em minutos alterando os dados do piloto. Os filtros de ordenação de mototaxistas por nome, pontuação e tempo de resposta foram eficazes na função proposta. Todos os aparelhos obtiveram êxito na busca do endereço via GPS e no reposicionamento do marcador no mapa recuperando o novo endereço.

Conclusões

Ao finalizar o desenvolvimento das camadas da aplicação desde o banco de dados até o aplicativo, foi possível perceber o quanto o desenvolvimento para aplicações móveis pode contribuir para o desenvolvimento da categoria dos mototaxistas. Os requisitos levantados juntamente com os representantes da categoria e os potenciais passageiros, foram atendidos sempre mantendo o foco na simplicidade de uso. Ao verem o aplicativo realizando todas as ações planejadas, eles afirmaram que o usuário caso ele seja implementado em um ambiente real.

A criação do aplicativo de maneira híbrida simplificou o desenvolvimento deixando mais tempo para elaboração das regras de negócio, apesar de ser uma aplicação executada pelo navegador, o Cordova provê muitos recursos de hardware que podem ser usados de maneira intuitiva dentro do software.

O conceito de servidor RESTful otimizou a troca de informações entre os usuários através dos métodos HTTP. Usando o JSON como transporte das informações pôde-se sustentar muitos dados e transitá-los de maneira leve pela rede. O conceito de JPQL mostrou-se eficaz na busca de informações do banco de dados em conjunto com o gerenciador de entidades do Java.

AngularJS é um eficiente *framework* que reduziu a quantidade de códigos escritos em *backend*, o seu escopo pode efetuar muitas tarefas interligando as *views* ao *controller*, suas diretivas são capazes de realizar diversas funções ainda no código HTML evitando requisições desnecessárias.

As contribuições com a construção deste projeto foram, aprimorar o conhecimento sobre tecnologias móveis e desenvolvimento para aplicações *Web* e o embasamento sobre gerenciamento de softwares na preparação e a organização das etapas. Caso o aplicativo seja implantado em um ambiente real poderá ajudar os clientes a encontrarem mototaxistas de uma maneira unificada, poderá também ampliar os meios que os mototaxistas recebem requisições.

6.1 Trabalhos Futuros

Como trabalhos futuros serão implementadas as funções que possibilitarão o cliente cancelar uma corrida que já foi confirmada, o mototaxista ter a disponibilidade de alterar as suas observações, sua foto e seu nome sem precisar passar pelo administrador do sistema. Um campo que pode ajudar os passageiros a decidirem entre os mototaxistas é o de comentários de outros usuários. Quando a geolocalização for usada pelo passageiro e pelo piloto, calcular um tempo de chegada até o local solicitado. Implementar possibilidade de personalização da aplicação como cores e fontes.

Referências Bibliográficas

ABREU, Artur José Dias de. **MOTOTÁXI: PROPOSTA DE PARÂMETROS E NORMAS GERAIS PARA CONCESSÃO DO SERVIÇO - ESTUDO DE CASO NO MUNICÍPIO DE BETIM-MG.** 2012. 137 f. Dissertação (Mestrado) - Curso de Mestrado em Geotecnia e Transportes, Universidade Federal de Minas Gerais, Escola de Engenharia, Belo Horizonte, 2012. Disponível em: <<https://goo.gl/3ctmz3>>. Acesso em: 30 nov. 2015.

BRANAS, Rodrigo. **AngularJS Essentials:** Design and construct reusable, maintainable, and modular web applications with AngularJS. Birmingham: Packt Publishing, 2014. 164 p.

CORDOVA. **Documentation:** Apache Cordova. Disponível em: <<https://cordova.apache.org/docs/en/latest/>>. Acesso em: 20 nov. 2016.

DATAFOLHA (São Paulo). **Jovens brasileiros.** 2008. Disponível em: <<http://goo.gl/mFTX1W>> Acesso em: 03 nov. 2015.

DEITEL, Harvey M.; DEITEL, Paul J.. **Java:** Como programar. 4. ed. Porto Alegre: Bookman, 2003. Disponível em: <<http://goo.gl/pgV7xF>>. Acesso em: 06 mar. 2016.

FONSECA, Rúben; SIMÕES, Alberto. **Alternativas ao XML: YAML e JSON.** 2007. Disponível em: <<http://goo.gl/5VOl7S>>. Acesso em: 13 mar. 2016.

FONTANA JUNIOR, Sergio Antônio. **PROTÓTIPO DE UM APLICATIVO ANDROID PARA PEDIDOS DE LANCHES E UM PORTAL WEB PARA GESTÃO E MONITORAMENTO.** 2013. 131 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade do Planalto Catarinense, Lages, 2013. Disponível em: <<http://goo.gl/HU87Rq>>. Acesso em: 10 abr. 2016.

GOOGLE (Brasil). **Ajuda do Google Maps.** 2015. Disponível em: <<https://support.google.com/maps/?hl=pt-BR#>>. Acesso em: 25 nov. 2015.

INTEL (Brasil). **Intel Developer Zone:** Visão Geral. 2016. Disponível em: <<https://software.intel.com/pt-br/intel-xdk>>. Acesso em: 03 out. 2016.

JQUERY (New York). **What is jQuery?:** jQuery API. 2016. Disponível em: <<https://jquery.com/>>. Acesso em: 04 out. 2016.

KUHN, Clayton Eduardo. **ELABORAÇÃO DE UM PROTÓTIPO DE APLICATIVO PARA ACOMPANHAMENTO DE REQUISIÇÕES DE TÁXI.** 2012. 69 f. Monografia (Especialização) - Curso de Especialização em Tecnologia Java, Universidade Tecnológica Federal do Paraná, Curitiba, 2012. Disponível em: <<http://goo.gl/JvbcYZ>>. Acesso em: 10 abr. 2016.

LECHETA, Ricardo R.. **Web Services RESTful:** Aprenda a criar web services RESTful em Java na nívem do Google. São Paulo: Novatec, 2015. 432 p.

LOUDON, Kyle. **Desenvolvimento de Grandes Aplicações Web.** São Paulo: Novatec Editora Ltda, 2010. Disponível em: <<http://goo.gl/3peoJf>>. Acesso em: 12 mar. 2016.

MATTJE, Guilherme Lopes. **INTEGRANDO SERVIÇOS CORPORATIVOS EM APPLICATIVO PARA SMARTPHONES SEM USO DE LINGUAGENS NATIVAS:** Estudo de caso na empresa Softplan Planejamento e Sistemas Ltda. 2014. 45 f. TCC (Graduação) - Curso de Análise e Desenvolvimento de Sistemas, Centro Universitário Estácio de Sá de Santa Catarina, São José (sc), 2014. Disponível em: <<https://goo.gl/2IzBKy>>. Acesso em: 20 nov. 2016.

MAZZA, Lucas. **HTML5 e CSS3:** Domine a web do futuro. São Paulo: Casa do Código, 2013. 195 p.

MELCHIOR, Caique Castanho Bolognesi. **CONCEPÇÃO E IMPLEMENTAÇÃO DE UM SERVIÇO BASEADO EM LOCALIZAÇÃO NA PLATAFORMA ANDROID/GOOGLE.** 2011. 63 f. Monografia (Especialização) - Curso de Ciência da Computação, Matemática, Estatística e Computação (dmec), Universidade Estadual Paulista Júlio de Mesquita Filho, Presidente Prudente, 2011. Disponível em: <<http://goo.gl/Yq68lj>>. Acesso em: 11 abr. 2016.

NETBEANS (Brasil). **Docs & Support:** Conceitos Básicos sobre Criação de uma Aplicação Cordova. 2016. Disponível em: <<https://goo.gl/VRSWw1>>. Acesso em: 03 out. 2016.

ORACLE. **JAVA E VOCÊ, FAÇA DOWNLOAD HOJE.** Disponível em: <<http://docs.oracle.com>>. Acesso em: 06 mar. 2016.

PEREIRA, Lúcio Camilo Oliveira; SILVA, Michel Lourenço da. **Android:** Para desenvol-

vedores. Rio de Janeiro: Brasport Livros e Multimídia Ltda, 2009. 223 p.

SASAKI, Elton Eiji. **PROTÓTIPO DE SISTEMA DE ENTREGAS E COLETAS PARA MOTOBOYS**. 2012. 54 f. Monografia (Especialização) - Curso de Especialização em Tecnologia Java Vii, Universidade Tecnológica Federal do Paraná, Curitiba, 2012. Disponível em: <<http://goo.gl/7ldnlP>>. Acesso em: 26 fev. 2016.

SERRA, Ricardo Jorge Maia e. **Interfaces tácteis baseadas em HTML5/ CSS3/ JavaScript**. 2011. 77 f. Dissertação (Mestrado) - Curso de Engenharia Informática e Computação, Faculdade de Engenharia da Universidade do Porto, Porto, 2011. Disponível em: <<https://goo.gl/mswHH8>>. Acesso em: 03 out. 2016.

SILVA, Maurício Samy. **HTML 5: A LINGUAGEM DE MARCAÇÃO QUE REVOLUCIONOU A WEB**. 2. ed. São Paulo: Novatec, 2014. 330 p.

SILVA, Michele. **Mototáxi inova e aceita cartão de crédito na Rocinha**. 2015. Disponível em: <<https://goo.gl/1weUUf>>. Acesso em: 04 out. 2016.

SUEHRING, Steve. **MySQL Bible**. New York: Wiley Publishing, Inc., 2002. 686 p.

VASCONCELLOS, Eduardo A.. **O custo social da motocicleta no Brasil**. 2008. Disponível em: <<http://goo.gl/ksJ5Iy>>. Acesso em: 26 fev. 2016.

VETTORAZZI, F.J.; SCHUTZ, F. **TAXI CALCULATOR: APLICAÇÃO ANDROID PARA CONTROLE DE ROTAS DE TÁXI**. Revista Eletrônica Científica Inovação e Tecnologia, Medianeira, v. 2, n. 2, p.48-52, 07 jan. 2014. Disponível em: <<http://goo.gl/81EOsv>>. Acesso em: 10 abr. 2016.

Questionário de Avaliação

Questionário de avaliação do aplicativo Expert Mototáxi

Nome: _____ Idade: _____

Dispositivo (Marca, Modelo, Versão S.O.): _____



		1	2	3	4	5
1	Chegada das notificações push					
2	Notificações push com aplicativo em segundo plano					
3	Chegada das notificações dentro do aplicativo					
4	Localização do atual no mapa					
5	Retorno do endereço em forma de texto					
6	Pontuação dos mototaxistas					
7	Reorganização da listagem dos mototaxistas					
8	Gerenciamento de corridas					
9	Exibição do histórico					
10	Alertas das telas					
11	Organização dos dados					
12	O designe das telas foi intuitivo					
13	Erros inesperados					
14	Ações podem ser feitas com praticidade					
15	Tempo de entendimento das etapas					
16	As informações foram necessárias					
17	Instruções explícitas					
18	Execução correta ao que o aplicativo propõe fazer					
19	Sequência das etapas para realizar um chamado					
20	Leveza na execução do aplicativo					

Encontrou algum problema na execução ?

()NÃO.

()SIM. Qual ?

Sugere alguma alteração ?

()NÃO.

()SIM. Qual ?

Usará o aplicativo caso seja implantado em ambiente real ?

()SIM.

()NÃO. Motivo: